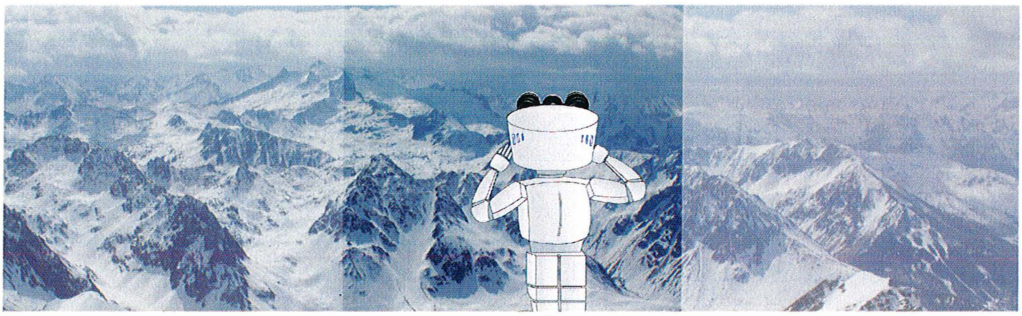
PANORAMA DE L'INTELLIGENCE ARTIFICIELLE



ses bases méthodologiques, ses développements

Algorithmes pour l'intelligence artificielle

* *

Coordinateurs :

Pierre Marquis
Odile Papini
Henri Prade

Préface :

Alain Colmerauer

Cépaduès
ÉDITIONS

PANORAMA DE L'INTELLIGENCE ARTIFICIELLE
ses bases méthodologiques, ses développements

Volume 2

Algorithmes pour l'intelligence artificielle

* *

Coordinateurs :
Pierre Marquis
Odile Papini
Henri Prade

Préface :
Alain Colmerauer

CÉPADUÈS-ÉDITIONS
111, rue Nicolas Vauquelin
31100 Toulouse – France
Tél. : 05 61 40 57 36 – Fax : 05 61 41 79 89
www.cepadues.com
Courriel : cepadues@cepadues.com
Coordonnées GPS en WGS 84
N 43° 34'43,2"
E 001° 24'21,5"

Illustration de couverture réalisée par Émilie Prade, http://emilieprade.com/
Photographie de couverture : Sylvain Erdei

ISBN : 978.2.36493.042.1

Dépôt légal : mai 2014

Présentation de l'ouvrage

L'intelligence artificielle (IA) a cinquante ans révolus. Elle occupe une place singulière dans le champ très vaste de l'informatique. Alors même que l'IA n'a jamais connu autant de développements et d'applications variés, ses résultats restent largement méconnus dans leur ensemble, y compris dans la communauté des chercheurs en informatique.

Au-delà de monographies introductives, il n'existe pas de traité offrant une vue d'ensemble approfondie, et à jour, des recherches dans ce domaine. C'est pourquoi il était important de dresser l'état des lieux des travaux en IA au plan international. Le présent « panorama de l'intelligence artificielle – ses bases méthodologiques, ses développements » vise à répondre à cette demande.

Pour cette entreprise de grande ampleur, il a été fait largement appel à la communauté française en IA. Chaque chapitre est écrit par un ou des spécialiste(s) du domaine abordé.

L'ouvrage est organisé est trois volumes :

— le premier volume regroupe vingt chapitres traitant des fondements de la représentation des connaissances et de la formalisation des raisonnements ;
— le deuxième volume offre une vue de l'IA, en onze chapitres, sous l'angle des algorithmes ;
— le troisième volume, en onze chapitres également, décrit les principales frontières et applications de l'IA.

Si chaque chapitre peut être lu indépendamment des autres, les références croisées entre chapitres sont nombreuses et un index global de l'ouvrage permet d'aborder celui-ci de façon non linéaire.

Quelle que soit la façon dont le lecteur utilisera cet ouvrage, nous espérons que le panorama proposé le réjouira et satisfera sa curiosité.

Sommaire

Volume 1

Volume 2

Volume 3

Volume 2
Algorithmes pour l'intelligence artificielle

La simulation de comportements « intelligents » à laquelle l'intelligence artificielle (IA) vise est une tâche complexe. Elle nécessite l'identification des concepts et processus mis en jeu dans de tels comportements. Ces concepts (croyances, préférences, actions, etc.) et processus (apprentissage, inférence, décision, etc.) doivent être modélisés. Il importe donc de définir des modèles qui sont adaptés à la réalité dont on souhaite rendre compte (en particulier, en permettant l'exploitation des données disponibles) et qui possèdent de « bonnes propriétés » (que l'on se place d'un point de vue normatif ou descriptif). Les concepts, une fois modélisés, doivent être représentés. Ceci requiert de définir et d'étudier des langages de représentation (une même information au sens d'un modèle donné possédant en général plusieurs représentations possibles). Le choix d'un langage de représentation adapté dépend typiquement de plusieurs critères (expressivité, concision, entre autres).

Les processus modélisés doivent également être représentés pour rendre une simulation informatique possible. Cela passe par la conception, l'étude et l'évaluation d'algorithmes dédiés.

Le volume II de ce panorama de l'IA est centré sur cette dernière problématique et a pour objet de présenter en onze chapitres les principales familles d'algorithmes développés ou utilisés en IA pour apprendre, inférer, décider.

Les deux premiers chapitres de ce volume traitent de recherche heuristique ; on y trouve en particulier une présentation de ce paradigme général de résolution de problèmes en IA (chapitre 1) mais aussi l'utilisation d'algorithmes de recherche heuristique pour résoudre des problèmes de jeux et les nouveaux algorithmes que cette utilisation a conduit à développer (chapitre 2).

Les trois chapitres suivants sont centrés sur les algorithmes traitant de représentations logiques de l'information et présentent les problématiques de la déduction automatique (chapitre 3), celles de la programmation logique (chapitre 4), dont la préface de ce volume rappelle toute l'importance, et enfin celles de la logique propositionnelle classique (dont le fameux problème SAT de satisfaisabilité d'un ensemble de formules logiques).

Les trois chapitres qui viennent ensuite sont focalisés sur des algorithmes opérant sur des représentations graphiques de l'information, qu'il s'agisse de réseaux de contraintes booléennes (chapitre 6), de réseaux de contraintes valuées (chapitre 7), ou de réseaux

bayésiens (entre autres) (chapitre 8).

Les trois chapitres finaux font écho aux deux premiers en n'étant pas centrés sur des formalismes de représentation particuliers, mais en se focalisant sur l'algorithmique développée en IA pour simuler des processus particuliers (ou, plus généralement, des familles de tels processus), comme la planification (chapitre 9) ou l'apprentissage (chapitre 10), ou encore pour résoudre des problèmes d'optimisation, une classe générale de problèmes de calcul à laquelle de nombreux problèmes d'IA (mais pas seulement) appartiennent (chapitre 11).

Le volume se clôt par une postface liant et comparant les problématiques de la recherche opérationnelle et de l'IA en matière d'optimisation.

Liste des auteurs du volume 2

Salem Benferhat (Université d'Artois, CRIL-CNRS)
benferhat@cril.fr
Christian Bessiere (Université de Montpellier II, CNRS, LIRMM)
bessiere@lirmm.fr
Bruno Bouzy (Université René Descartes, LIPADE)
bruno.bouzy@parisdescartes.fr
Thierry Boy de la Tour (CNRS, Laboratoire d'Informatique de Grenoble (LIG))
thierry.boy-de-la-tour@imag.fr
Ricardo Caferra (Université Joseph Fourier, LIG-CNRS)
ricardo.caferra@imag.fr
Tristan Cazenave (Université Paris Dauphine, LAMSADE-CNRS)
cazenave@lamsade.dauphine.fr
Martin C. Cooper (Université de Toulouse, IRIT-CNRS)
cooper@irit.fr
Antoine Cornuéjols (AgroParisTech, Paris)
antoine.cornuejols@agroparistech.fr
Vincent Corruble (Université Pierre et Marie Curie, LIP6-CNRS)
vincent.corruble@lip6.fr
Simon de Givry (INRA - Unité de Mathématiques et Informatique Appliqués, Castanet Tolosan)
simon.degivry@toulouse.inra.fr
Henri Farreny(anciennement INP Toulouse, IRIT)
farreny.henri@free.fr
Jin-Kao Hao (Université d'Angers, LERIA)
jin-kao.hao@univ-angers.fr
Arnaud Lallouet (Université de Caen/Basse-Normandie, GREYC-CNRS)
arnaud.lallouet@unicaen.fr
Philippe Leray (Université de Nantes, LINA-CNRS)
philippe.leray@univ-nantes.fr
Yves Moinard (Université de Rennes I, IRISA)
yves.moinard@irisa.fr
Pascal Nicolas (anciennement Université d'Angers, LERIA)
Nicola Olivetti (Université d'Aix-Marseille, LSIS-CNRS)
nicola.olivetti@lsis.org

Nicolas Peltier (CNRS, Laboratoire d'Informatique de Grenoble (LIG))
nicolas.peltier@imag.fr
Régis Sabbadin (INRA - Unité de Mathématiques et Informatique Appliqués, Castanet Tolosan)
regis.sabbadin@toulouse.inra.fr
Thomas Schiex (INRA - Unité de Mathématiques et Informatique Appliqués, Castanet Tolosan)
thomas.schiex@toulouse.inra.fr
Camilla Schwind (anciennement Université d'Aix-Marseille, CNRS, LSIS)
camilla.schwind@lif.univ-mrs.fr
Laurent Simon (Université Paris-Sud, LRI-CNRS)
simon@lri.fr
Christine Solnon (INSA de Lyon, LIRIS-CNRS)
christine.solnon@liris.cnrs.fr
Igor Stéphan (Université d'Angers, LERIA)
igor.stephan@univ-angers.fr
Karim Tabia (Université d'Artois, CRIL-CNRS)
tabia@cril.fr
Florent Teichteil-Königsbuch (ONERA)
florent.teichteil@onera.fr
Olivier Teytaud (Université Paris-Sud, LRI-CNRS)
olivier.teytaud@lri.fr
Vincent Vidal (ONERA)
vincent.vidal@onera.fr
Christel Vrain (Université d'Orléans, LIFO)
christel.vrain@univ-orleans.fr
Jean-Daniel Zucker (Institut de Recherche pour le Développement)
jean-daniel.zucker@ird.fr

Préface

Qu'entend-on exactement par « intelligence artificielle » ? John McCarthy, de l'Université de Stanford, voulait organiser un congrès sur le « traitement d'informations complexes », ce qui englobait aussi bien l'apprentissage, la planification, les jeux d'échecs, le contrôle de robot, la démonstration de théorèmes, la traduction automatique, la démonstration automatique de théorèmes, la reconnaissance de la parole, etc. Ne trouvant pas le sujet suffisamment accrocheur il proposa ce terme plus « *flashy* », me dit-il afin d'obtenir des crédits. Cela se passait durant l'été 1956. En cinquante ans, les choses ont beaucoup évolué, ne serait-ce que parce que les ordinateurs ont beaucoup progressé.

Entre 1971 et 2013, le nombre de transistors à l'intérieur d'un micro-processeur est passé de quelques milliers à quelques milliards. La vitesse et la taille mémoire des ordinateurs ont pratiquement doublé chaque année. La date de 1971 est précisément celle où Nils Nilson dans son livre, « *Problem-solving Methods in Artificial Intelligence* », expose différentes recherches arborescentes dans l'algorithmique des jeux. C'est maintenant, avec des ordinateurs quelques millions de fois plus rapides, que les algorithmes de recherche heuristique prennent leur vrai sens. Par exemple Toma Rokick, Herbert Kociemba *et al.* ont montré en 2010 qu'il est possible de résoudre le cube de Rubik en au plus 20 mouvements, résultat obtenu par une recherche exhaustive du quarantième des 43 252 003 274 489 856 000 positions possibles. Cette algorithmique des jeux est l'objet des deux premiers chapitres de ce volume.

Cette rapidité et cette capacité ont aussi bénéficié à tout ce qui touche de près ou de loin à la logique computationnelle : les Prolog actuels sont des langages de programmation performants, les démonstrateurs propositionnels manipulent des énoncés avec 10^6 propositions.

C'est la démonstration automatique de théorèmes, telle que mise au point par Alan Robinson, qui va nous servir de fil conducteur pour montrer ce qu'ont développé les auteurs dans chaque chapitre de ce volume.

On utilise des formules du genre

$$\forall x(\text{Frère}(Paul, x) \rightarrow (\exists y(Egal(y, \text{Mère}(Paul)) \wedge Egal(y, \text{Mère}(x)))))$$

qui sont faites de formules atomiques de connecteurs et de quantificateurs ; les formules atomiques elles-mêmes étant des symboles de prédicat portant sur des termes formés de variables sans arguments et de symboles fonctionnels. Il s'agit de la logique du

Auteur : ALAIN COLMERAUER.

premier ordre. Il est toujours possible, après avoir introduit des fonctions de Skolem pour éliminer la quantification existentielle, de se ramener à la forme de clause

$$L_1 \vee \cdots \vee L_n$$

où chaque L_i est soit une formule atomique ou la négation d'une telle formule. La seule quantification permise est universelle, elle est sous-entendue. En 1965, Alan Robinson, dans son article « *A Machine-Oriented Logic Based on the Resolution Principle* », montre que l'on n'a plus besoin dans ce cas d'une règle d'inférence :

$$\text{de } A \vee L_1 \vee \cdots \vee L_m \text{ et de } \neg A \vee L_{m+1} \vee \cdots \vee L_n \text{ déduire } L_1 \vee \cdots \vee L_n.$$

En fait, cela n'est vrai que si les littéraux ne contiennent pas de variables. En général il faut d'abord « unifier » les formules atomiques en calculant une substitution et l'appliquer au résultat. Par la même occasion, Alan Robinson constate aussi que l'on peut dans ce cas se limiter à des interprétations d'Herbrand : une variable est interprétée comme un terme sans variable et les symboles fonctionnels sont interprétés par eux-mêmes. La démonstration automatique de théorème s'est beaucoup développée : par exemple William McCune au moyen d'Otter (un des nombreux démonstrateurs) a trouvé en 1998 la preuve que l'algèbre de Robbins est une algèbre de Boole. J'ai trouvé ce résultat au début du chapitre 3. Le reste de ce chapitre est consacré à des logiques non classiques et une méthode par « tableaux ». Cette méthode, introduite par Evert Beth, Raymond Smuyllian et Jaakko Hintikka, travaille directement sur la formule logique initiale, avec tous les quantificateurs, tout en se servant aussi de l'unification.

Une restriction importante de ne servir que des clauses avec un et un seul littéral sans négation

$$A_0 \vee \neg A_1 \vee \cdots \vee \neg A_n \text{ c'est-à-dire } A_0 \leftarrow A_1 \wedge \cdots \wedge A_n,$$

les A_i étant des formules atomiques. Ainsi que l'ont montré Robert Kowalski et Marteen van Emden, un tel ensemble de formules possède un modèle minimal de Herbrand (les prédicats moins souvent vrais que possibles) dans lequel il est possible de calculer, voir le début du chapitre 4. Donc les affirmations ne jouent pas le même rôle que les négations. Le langage Prolog a pu se développer grâce à cette notion, mais avec des ajouts non déclaratifs, tels que les prédicats évaluables « *cut* », « *setof* », d'entrée et sortie, sans lesquels ce ne serait pas un véritable langage de programmation. Saluons au passage David Warren qui par son compilateur a vraiment consacré ce langage de programmation. Des extensions à l'unification, qui est davantage vue comme un système de résolutions de contraintes, m'ont permis de traiter les réels linéaires avec Prolog III et à Mehmet Dincbas, Pascal van Henteryck et Helmut Simonis de traiter les domaines finis avec CHIP. Parallèlement, Joxan Jaffar et Jean-Louis Lassez introduisent la notion de « *Constraint Logic Programming* » qui donne un cadre théorique à tout ceci. Un formalisme de résolution de contraintes dans des domaines finis les CSP est particulièrement étudié aux chapitres 6 et 7. On n'est pas loin de la Recherche Opérationnelle, voir la postface.

Une restriction plus faible est de ne servir que des clauses de la forme

$$A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \neg \wedge A_n,$$

où la négation est interprétée comme une extension de la négation par échec : les modèles « stables » de Michael Gelfond and Vladimir Lifschitz. Ceci est amplement développé dans la deuxième partie du chapitre 4. En particulier, on y voit la relation avec la « *circumscription* » de John McCarthy qui introduit un ensemble de formules supplémentaires pour redonner à ces formules un sens logique habituel.

Une autre restriction où chaque clause est de la forme

$$Q_1 \vee \cdots \vee Q_n,$$

chaque Q_i étant soit un prédicat avec zéro argument ou sa négation. C'est le problème SAT qui est par excellence le problème dans le domaine fini. Il a de nombreuses applications dans l'industrie. Il est traité par l'excellent chapitre 5. Mentionnons aussi la résolution de problèmes combinatoires NP-difficiles, au chapitre 11.

Les chapitres 8, 9, 10 ont trait à l'incertitude, la planification et l'apprentissage. Ces thèmes sont vitaux. L'incertitude parce que nos données sont souvent approximatives. La planification pour fournir la suite d'actions à exécuter pour accomplir une tâche, par exemple conduire un véhicule autonome.

Enfin l'apprentissage qui manque si cruellement à nos réalisations. Si les machines apprenaient tout d'elles-mêmes ce serait un progrès de taille, nous n'aurions plus rien à faire ... ou presque.

Chapitre 1

Recherche heuristiquement ordonnée dans les graphes d'états

Au début des années 1970, les techniques de résolution de problèmes par *recherche heuristique dans des graphes d'états* : (*heuristic graph search*) apparaissent comme une des branches maîtresses de la jeune discipline qui s'affirme alors sous l'appellation « intelligence artificielle » (IA). L'ouvrage de référence dans la période 1970-1980 : « *Problem-solving methods in artificial intelligence* » [Nilsson, 1971] atteste l'importance originelle du sujet. En témoignent aussi les tables des matières des premiers volumes de la série « *Machine Intelligence* » (à partir de 1967), les programmes des premières conférences mondiales « IJCAI » (*International Joint Conference on Artificial Intelligence*; à partir de 1969) et les sommaires des premiers numéros de la revue « *Artificial Intelligence* » (à partir de 1970). Le sujet est demeuré en bonne place dans la plupart des ouvrages qui ont accompagné le développement des enseignements d'intelligence artificielle. Citons : [Winston, 1984], [Nilsson, 1980], [Barr *et al.*, 1981], [Rich, 1983], [Shirai et Tsujii, 1984], [Charniak et McDermott, 1985], [Laurière, 1986], [Farreny et Ghallab, 1987], [Farreny, 1987], [Shapiro, 1987], [Nilsson, 1998], [Russell et Norvig, 2003]. Parmi les ouvrages spécialisés sur le sujet : [Pearl, 1984], [Kanal et Kumar, 1988], [Farreny, 1995].

1.1 Introduction

Esquissons le principe de la recherche heuristique dans les graphes. On suppose connues les notions de *graphe*, *sommet*, *arc* et *chemin* (dans les graphes dits *orientés*), *arête* et *chaîne* (dans les graphes dits *non orientés*), *sous-graphe* d'un graphe. Un sommet N est *fils* d'un sommet M dans un graphe orienté : dès qu'il existe un arc (M, N) d'extrémité initiale M et d'extrémité finale N ; dans un graphe non orienté :

Auteur : HENRI FARRENY.

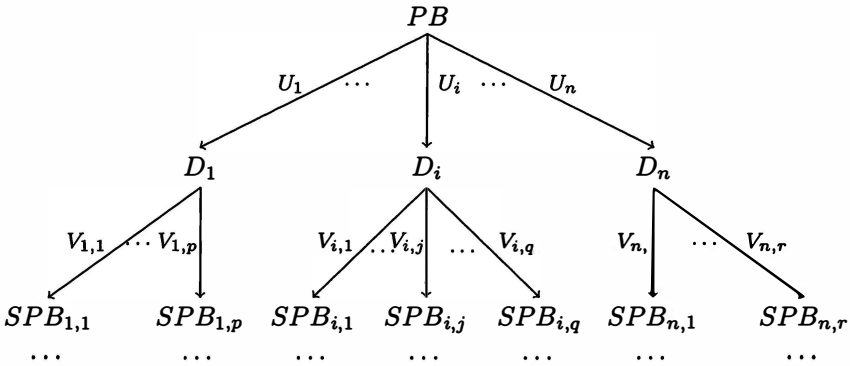


FIGURE 1 – Fragment de graphe de sous-problèmes : on peut résoudre PB en résolvant l'une ou l'autre des n conjonctions de sous-problèmes : $[SPB_{1,1}$ et ... et $SPB_{1,p}]$ ou ... $[SPB_{i,1}$ et ... et $SPB_{i,q}]$ ou ... $[SPB_{n,1}$ et ... et $SPB_{n,r}]$. A son tour, chaque $SPB_{i,j}$ peut être décomposé dès lors qu'il n'est pas un problème-primitif. PB est le sommet-origine de la recherche de solution.

dès qu'il existe une arête $\{M, N\}$. Si N est fils de M, alors M est *père* de N ; et vice-versa.

Un procédé général pour attaquer un problème consiste à tenter de le décomposer en sous-problèmes, puis à tenter de décomposer ceux-ci, etc., jusqu'à n'avoir plus que des *problèmes primitifs* à considérer, i.e. des problèmes dont la solution est considérée immédiatement accessible. Les décompositions que l'on peut envisager peuvent être rassemblées en un *graphe des sous-problèmes* (appelé aussi *graphe et/ou*[1]) tel que le fragment présenté en figure 1.

La partie de ce graphe qui comprend les $p+2$ sommets $PB, D_1, SPB_{1,1} \ldots SPB_{i,p}$ et les $p+1$ arcs $U_1, V_{1,1} \ldots V_{1,p}$ exprime une manière possible d'attaquer le problème représenté par le sommet PB : ce problème peut être résolu si (mais : pas seulement si) on résout la conjonction de p sous-problèmes représentés par les sommets $SPB_{1,1}$ à $SPB_{1,p}$. De même, le problème peut être résolu si on résout la conjonction de q sous-problèmes représentés par les sommets $SPB_{i,1}$ à $SPB_{i,q}$, que i vaille $2, 3 \ldots n$. En définitive, la figure 1 exhibe n manières possibles d'attaquer le problème PB par décomposition.

Les sommets PB et SPB_i sont dits *sommets-ou*, tandis que les sommets tels que D_i sont dits *sommets-et*. D_i est un des n fils de PB ; $SPB_{i,1} \ldots SPB_{i,q}$ sont tous les fils de D_i. Les sommets sans fils représentent des problèmes-primitifs. Produire explicitement le graphe des sous-problèmes est évité ou exclu pour des questions de coût (place et temps). Résoudre le problème représenté par PB consiste à construire un sous-graphe SG du graphe G des sous-problèmes de sorte que :

1. Les graphes et/ou ressortissent de la famille des *hyper-graphes* [Berge, 1970].

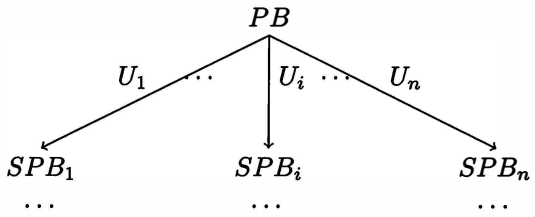


FIGURE 2 – Fragment de graphe d'états : on peut résoudre le problème PB en résolvant l'un ou l'autre des sous-problèmes SPB_1 ou ... ou SPB_i ou ... ou SPB_n, puis en itérant.

1. SG contient PB,
2. chaque sommet-ou de SG qui ne représente pas un problème-primitif conserve un seul des fils qu'il possède dans G ainsi que l'arc qui le joint à ce fils,
3. chaque sommet-et de SG conserve tous les fils (et les arcs qui le joignent à chacun des fils) qu'il possède dans G.

La recherche d'un tel sous-graphe peut être guidée par des connaissances d'origine plus ou moins empirique : on parle alors de *recherche heuristiquement*[2] *ordonnée dans un graphe de sous-problèmes.* Les algorithmes usités diffèrent selon la manière dont il commandent l'expansion et l'exploration du graphe de sous-problèmes (et selon l'efficacité subséquente).

Ordinairement on considère le cas particulier où toute décomposition i applicable à un problème PB ne produit que 2 sous-problèmes $SPB_{i,1}$ et $SPB_{i,2}$ avec, systématiquement, $SPB_{i,1}$ primitif. On peut alors adopter une représentation plus simple (voir figure 2) : l'arc U_i joint directement PB à $SPB_{i,2}$ (noté maintenant SPB_i) ; le sommet D_i ainsi que les arcs $V_{i,1}$ et $V_{i,2}$ disparaissent : U_i représente la résolution immédiate de $SPB_{i,1}$. La nouvelle représentation exprime que résoudre PB revient à résoudre SPB_1 ou SPB_2 ou ... ou SPB_n. En itérant de même sur chaque sous-problème qui n'est pas problème-primitif, on obtient un *graphe d'états*[3]. Produire explicitement le graphe d'états est évité ou exclu pour des questions de coût (place et temps). Résoudre le problème initial consiste à construire un *chemin* depuis PB jusqu'à un problème-primitif, c'est-à-dire une suite de sommets commençant par PB et terminant par un problème-primitif, telle que chaque sommet hormis le premier soit fils de celui qui le précède dans la suite. On parle alors de *recherche heuristiquement ordonnée dans un graphe d'états.*

Formellement, on peut ramener une recherche dans un graphe de sous-problèmes à une recherche dans un graphe d'états. Pratiquement, des procédures spécifiques ont été développées. On ne traitera ici que de *recherche heuristiquement ordonnée dans des*

2. *Heuristique* (adjectif ou nom, du grec *heuriskein* : trouver) : qui a une utilité pour la découverte.
3. Représentable par un simple *graphe* plutôt qu'un *hypergraphe* [Berge, 1970].

graphes d'états. Pour la recherche dans des graphes de sous-problèmes, on peut consulter : [Pearl, 1984], [Farreny et Ghallab, 1987], [Russell, 1992 ; Russell et Norvig, 2003]. Les ressources et résultats de la recherche heuristiquement ordonnée dans les graphes sont souvent ignorés des manuels consacrés à la *théorie des graphes*, l'*algorithmique discrète* ou la *recherche opérationnelle*, malgré des proximités manifestes avec ces disciplines. Pour des rapprochements entre *recherche heuristique dans les graphes*, *programmation dynamique* et procédure *branch-and-bound* introduite en recherche opérationnelle, on peut consulter [Ibaraki, 1978], [Kumar et Kanal, 1988].

La plupart des ouvrages orientés vers l'enseignement de l'intelligence artificielle présentent la recherche dans les graphes d'états, en reprenant une part de l'exposé pionnier de [Nilsson, 1971]. L'énoncé des propriétés des algorithmes est souvent sur-contraint et les justifications lacunaires. La recherche heuristiquement ordonnée est souvent abordée dans les cours d'intelligence artificielle (mais rarement en algorithmique générale) que dispensent universités et écoles d'ingénieurs. L'attention qui lui est accordée résulte pour beaucoup des progrès qu'elle a permis d'accomplir en matière de résolution *informatique* (plutôt qu'*analytique*) d'un vieux mais résistant problème relatif au *jeu de taquin*, que nous introduisons maintenant [4].

1.2 Les taquins, défi conducteur pour la recherche heuristiquement ordonnée

Les taquins sont des jeux devenus populaires à la fin du 19e siècle et encore très familiers aujourd'hui. Dans la forme matérielle la plus répandue (voir figure 3), 15 jetons carrés, de même taille, portant chacun une marque exclusive (ici les nombres de 1 à 15) sont juxtaposés à l'intérieur d'un cadre carré qui pourrait accueillir 16 jetons de ce type, rangés sur 4 lignes et 4 colonnes. L'absence d'un 16e jeton ménage une *case vide*. Les seuls mouvements (*coups*) autorisés sont ceux qui font glisser dans la case vide l'un des 2, 3 ou 4 jetons qui lui sont contigus. Le jeu consiste à déplacer les jetons un par un pour transformer un *état initial* quelconque en un *état but* fixé, *en un nombre minimal de coups*.

NB : On ne s'intéresse ordinairement qu'à la production de solutions *minimales*. Produire une simple solution est beaucoup moins épineux (par exploration informatique ou humaine). Produire une « bonne » solution est un autre problème encore.

Par exemple, il suffit de 9 coups pour transformer l'état initial s_1 en l'état but t ; et on ne peut pas faire moins. Par contre, il faut (et il suffit de) 80 coups pour transformer s_2 en t. Ce genre de taquins est dit *taquin* 4×4 (4×4 *puzzle* ou *fifteen-puzzle*).

4. D'autres puzzles (ou solitaires) offrent un champ d'expérimentation et de démonstration, dont le Rubik's cube. Des problèmes classiques de théorie des graphes ou recherche opérationnelle ont été revisités : détermination d'itinéraires optimaux, problème du voyageur de commerce. Comme autres domaines d'application abordés, on peut mentionner : le guidage de démonstrateurs automatiques de théorèmes, [Kowalski, 1970] [Chang et Lee, 1973] [Minker *et al.*, 1973] [Dubois *et al.*, 1987], le traitement d'images [Martelli, 1976], la navigation en robotique [Chatila, 1982] [Gouzènes, 1984] [Lirov, 1987], la génération automatique de langage naturel [Farreny *et al.*, 1984], la correction d'orthographe [Pérennou *et al.*, 1986], le diagnostic de pannes [Gonella, 1989], la diététique [Buisson, 2008].

s_1

1	5	2	3
4	6		7
8	14	10	11
12	9	13	15

t

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

s_2

15	11	8	12
14	10	9	13
2	6	1	4
3	7	5	

FIGURE 3 – Passer minimalement de s_1 à t : 9 coups. Passer minimalement de s_2 à t : 80 coups.

On peut associer à chaque état d'un taquin 4×4 un sommet de graphe et à chaque coup possible une arête entre 2 sommets. Le graphe d'états compte 16! sommets (soit : 20 922 789 888 000 $\approx 2.10^{13}$). Jouer équivaut à chercher dans le graphe d'états une chaîne (minimale en nombre d'arêtes, exigeons-nous dans ce contexte d'étude) entre les sommets représentant l'état initial et l'état but.

Dès 1879, il a été établi [Johnson et Storey, 1879] un *caractère de résolubilité* facile à mettre en œuvre [5], grâce auquel il apparaît que le graphe du taquin 4×4 comporte deux composantes connexes [6] comptant chacune 16!/2 sommets.

En conséquence, si on tire au sort l'état initial on a exactement 1 chance sur 2 de tomber sur un problème non résoluble et on peut tester immédiatement si tel est le cas. Mais, lorsqu'on sait qu'un problème est résoluble, on ne connaît pas de formule de calcul pour engendrer une chaîne minimale ou seulement fournir la longueur d'une telle chaîne.

On pourrait songer à utiliser un des algorithmes classiques en théorie des graphes, qui peuvent déterminer une chaîne minimale entre deux sommets (ici l'état initial et l'état but [7]) ; mais ce type d'algorithme suppose que l'entier graphe d'états est préconstruit en mémoire ; aujourd'hui encore, les ordinateurs courants ne peuvent représenter simultanément 16!/2 sommets en mémoire centrale. Et s'ils le pouvaient, qu'en serait-il pour le taquin 5×5, dont chaque composante connexe compte 25!/2 sommets ? Etc.

D'autres algorithmes, enseignés en théorie des graphes classique ou en algorithmique générale, sont aptes à construire progressivement en mémoire centrale une partie du graphe d'états du taquin, appelée graphe de recherche GR, jusqu'à discerner dans GR une chaîne minimale joignant le sommet-initial et le sommet-but.

Ainsi, *l'algorithme d'expansion d'un graphe en largeur d'abord* (souvent désigné comme : *breadth first algorithm*), partant de l'état initial s_1, engendre ses quatre états-fils ; puisqu'aucun d'eux n'est le but (t), l'algorithme produit et examine les fils de chacun d'eux, c'est-à-dire l'ensemble EJ_2 des états joignables depuis s_1 en au moins 2 coups ; itérativement, l'algorithme produit l'ensemble EJ_i des états joignables depuis s_1 en au moins i coups, pour $i = 3$ puis 4... puis 8, sans trouver le sommet-but t ; en

5. Chaque état étant représenté par la suite des numéros des jetons (par ex. : de haut en bas et de gauche à droite, case vide numérotée 0), un problème peut-être résolu si et seulement si l'état but est une permutation paire de l'état initial.

6. Deux sommets d'un graphe G satisfont la relation de connexité, si et seulement si, ils sont reliés au moins par une chaîne. Il s'agit d'une relation d'équivalence qui permet de distinguer des classes d'équivalence dans l'ensemble des sommets de G. S'il n'existe qu'une classe d'équivalence, G est un graphe connexe. Le sous-graphe de G constitué par tous les sommets d'une classe d'équivalence et toutes les arêtes de G qui relient deux sommets de la classe, est appelé composante connexe de G.

7. Par exemple l'algorithme de Moore-Dijkstra [Dijkstra, 1959] [Gondran et Minoux, 1979].

construisant EJ_9, l'algorithme rencontre inéluctablement le but t ; le mode de progression (s_1 puis EJ_1 puis $EJ_2, \ldots$ puis EJ_8 puis EJ_9), garantit que la chaîne découverte depuis s_1 jusqu'à t est minimale en nombre de coups. Cet algorithme présente un grave inconvénient : la croissance de l'ensemble des états représentés en mémoire centrale, rapportée à la profondeur d'exploration, est d'ordre exponentiel. Pour la résolution de problèmes de taquin 4×4 qui exigent plusieurs dizaines de coups, le besoin en espace est prohibitif. L'algorithme est aussi coûteux en temps.

Depuis [Ratner et Warmuth, 1990] on sait que trouver une solution minimale pour un problème quelconque de taquin $n \times n$ est NP-dur. Néanmoins, en pratique, on n'affronte pas le taquin $n \times n$ avec n quelconque : quelle que soit la complexité qui pèse a priori sur le problème général, on peut souhaiter résoudre tel cas précis ou telle famille restreinte de cas avec n fixé. De nombreux chercheurs ont décrit et expérimenté des techniques plus ou moins efficaces pour résoudre des problèmes de taquin ; longtemps ce furent des taquins 3×3 ; puis des 4×4 ; rarement des 5×5 (quelquefois des taquins à jetons rectangulaires et différents). Les taquins font partie des *Sliding-Tile Puzzles*, famille de solitaires qui inclut le cube de Rubik. Ont ainsi émergé des bases théoriques et algorithmiques pour la recherche heuristiquement ordonnée, susceptibles d'être utiles pour d'autres domaines d'application.

1.3 Au commencement était A^*

En 1968, Hart, Nilsson et Raphael [1968] [8] ont proposé un modèle d'algorithme communément désigné depuis comme *algorithme* A^*. Ce modèle (voir algorithme 1.1 ci-après) s'applique à des graphes d'états *arcs-valués positivement* [9] : à chaque arc (m, n), qui représente le passage d'un état m à un état n, est associé un *coût* $c(m, n)$ positif ; la longueur d'un chemin est mesurée par la somme des coûts des arcs [10] qui le composent ; l'algorithme cherche à construire un chemin de longueur minimale [11] depuis l'état initial (unique) s jusqu'à l'état but t. On peut considérer un ensemble d'états buts, défini explicitement ou par un faisceau de critères. L'algorithme bâtit une suite de *graphes de recherche* : $GR_0, GR_1 \ldots GR_x \ldots$ qui sont des sous-graphes du graphe d'états. On comprendra un peu plus loin que les GR_i sont des *arborescences* de *racine* s. Dans un graphe orienté un sommet r est racine si et seulement si :

1. r n'a pas de père
2. et quel que soit un sommet n différent de r, il existe un chemin depuis r vers n (évidemment : si une racine existe elle est unique).

Un graphe est une arborescence si et seulement si :

1. il comporte une racine
2. et quel que soit un sommet n différent de r, il n'a qu'un seul père.

8. Voir aussi : [Nilsson, 1971] [Nilsson, 1980] [Nilsson, 1998].

9. On raisonne ici dans le cadre de graphes orientés. Le modèle s'applique de même à des graphes non orientés : substituer alors *arête* à *arc*, *chaîne* à *chemin*. Le graphe d'états du taquin ordinaire peut être aussi bien être représenté par un graphe non orienté que par un graphe orienté (chaque arête correspond à 2 arcs de sens contraire).

10. Dans le cas des taquins, le coût d'un arc (ou arête) est ordinairement considéré égal à 1.

11. Dans le cas des taquins, il s'agit usuellement de minorer le nombre de coups depuis s jusqu'à t.

GR_0 est constitué uniquement de l'état initial s. Schématiquement : GR_x est dérivé de GR_{x-1} en *développant* un sommet de GR_{x-1}, c'est-à-dire en *sélectant* un sommet m de GR_{x-1} puis en *engendrant* tous les fils [12] de m ; à l'issue du x^e développement, GR_x remplace GR_{x-1} en mémoire centrale. La spécificité de A^*, par rapport au *breadth first algorithm* [13], réside dans le *mode de sélection* de m décrit plus loin. Sous des conditions que nous préciserons, A^* s'arrête après d développements, en découvrant dans GR_d (graphe de recherche à l'issue du d^e développement) un chemin [14] C depuis s jusqu'à t, minimal. Soit : la longueur de tout autre chemin, depuis s à t, dans le graphe d'états, est supérieure ou égale. La résolution est estimée d'autant plus efficace que GR_d est « plus petit » (économie de sommets et d'arcs). Le comportement de l'algorithme serait parfait si GR_d était réduit à n'être qu'un chemin depuis s à t.

Les travaux de Nilsson et collègues ont montré qu'à ressources informatiques équivalentes, certains problèmes peuvent être résolus efficacement en tirant parti de connaissances empiriques [15], tandis qu'on n'y parvient pas – ou plus difficilement – avec des algorithmes qui les ignorent. Ces algorithmes sont *non informés* quant aux domaines des problèmes (*uninformed algorithms* ou *blind search algorithms*). Par exemple, le *breadth-first algorithm* procède de la même manière qu'il soit appliqué à un problème de taquin ou à un problème d'itinéraire sur une carte routière ; il n'exploite que les relations de contiguïté entre cases du taquin ou villes de la carte, mais ne prend pas en compte les métriques spécifiques qu'on peut poser sur un taquin ou sur un espace géographique et les estimations heuristiques qui peuvent en découler (par exemple : la distance à vol d'oiseau). Commentons maintenant le code de A^* (un schéma plus simple est présenté au chapitre II.1).

1.3.1 Guider la recherche en recourant à des estimations heuristiques

Chaque cycle de base de l'algorithme A^* (lignes 3-21 de l'algorithme 1.1) assure le développement d'un état, soit m (choisi en ligne 5). L'algorithme maintient un *front* de recherche qui ne comporte initialement que s (ligne 2). La sélection du sommet du front qui va subir le $x+1^e$ développement, exploite une *fonction d'évaluation* f ; à l'issue du x^e développement, à chaque sommet q du front est associée une valeur $f_x(q)$; en vue de procéder au $x+1^e$ développement, A^* choisit un sommet p du front tel que $f_x(p)$ est « meilleure » dans le sens où : $f_x(p) = \min_{q \in front} f_x(q)$. Ce principe de choix est dit *meilleur d'abord* (*best first*). Ordinairement, p est considéré meilleur dans F_i si $f_x(p)$ est minimale.

12. On suppose que chaque état n'a qu'un nombre fini de fils : 4 au plus pour les taquins.

13. Et par rapport à sa généralisation appelée *algorithme du coût uniforme*. Parmi les sommets, jamais développés auparavant, le *breadth first algorithm* privilégie le développement de ceux joignables depuis s par un chemin dont le nombre d'arcs est minimal ; l'algorithme du coût uniforme préfère, lui, les sommets joignables depuis s par un chemin dont la somme des coûts d'arcs est minimale. Tandis que le *breadth first algorithm* est assuré de trouver un chemin de s à t minimal en nombre d'arcs (si un tel chemin existe), l'algorithme du coût uniforme est assuré de trouver un chemin de s à t minimal par rapport à la somme des coûts d'arcs (si un tel chemin existe).

14. On verra que GR_d est une arborescence, donc C sera l'unique chemin joignant s à t dans GR_d.

15. Soit : provenant originellement de l'expérience ou de l'observation (sans théorie préalable).

Par définition, A^* emploie des fonctions d'évaluation de la forme : $f_x(n) = g_x(n) + h(n)$ où $g_x(n)$ est la longueur du chemin [16] qui joint s à n dans GR_x, tandis que $h(n)$ est une *estimation heuristique* à valeurs positives ou nulles, attachée à n (ligne 4) : on dit que h est *statique* pour souligner [17] que, pour tout état n, la valeur $h(n)$ ne dépend que de n, pas du rang des développements. Dans le code de l'algorithme, pour tout n, les valeurs successives de $g_x(n)$ sont capturées par l'élément de tableau à une dimension $g_{noté}(n)$. En pratique, il est bienvenu de programmer A^* (et autres *Heuristic Search Algorithms*) au moyen de langages adaptés à la gestion de listes, auquel cas une information telle que $g_{noté}(n)$ peut être appréhendée comme un élément d'une liste de caractéristiques attachées à l'état n.

Initialement, A^* associe à s la valeur $g_{noté}(s) = 0$. Ensuite, le mode de mise à jour de $g_{noté}$ (lignes 15, 17, 20) assure que, à l'issue du x^e développement, $g_{noté}(n)$ est la longueur minimale des chemins de s à n qui ont été découverts au cours des développements de rangs $1, 2 \ldots x$. De la sorte, pour tout n et pour tout $z > y$, $g_z(n) \leq g_y(n)$: pour tout n, $g_x(n)$ est décroissante large par rapport au rang de développement x. En notant $g^*(n)$ la longueur d'un chemin minimal de s à n dans le graphe d'états, on observe que : pour tout rang de développement x, $g_x(n) \geq g^*(n)$. Donc : $g_x(n)$ est une surestimation de la longueur minimale de chemin depuis s à n, décroissante par rapport au rang de développement x.

Dans le cas particulier où h est identiquement nulle, l'algorithme A^* se confond avec l'*algorithme du coût uniforme* mentionné précédemment. Cet algorithme est baptisé ainsi parce qu'il agit comme si l'estimation de la distance minimale qui sépare du but t n'importe quel état n du front courant était uniformément la même, disons K ; sous cette hypothèse, pour le $x + 1^e$ développement il est naturel de choisir le sommet n qui minimise $g_x(n) + K$, ce qui revient à choisir n qui minimise $g_x(n)$. Nilsson et collègues ont remarqué que pour certains domaines on pouvait faire montre d'un meilleur discernement, d'où la substitution d'une fonction de n, $h(n)$, à la constante K (en première approche h est conçue *statique*).

Ainsi, pour des expérimentations sur des problèmes de taquin, Nilsson et collègues, ont d'abord employé comme estimateur heuristique $h(n)$ d'un état n, le nombre $W(n)$ des jetons de n qui ne sont pas à la place qui leur est assignée dans l'état but t ; par exemple, pour la figure 3, $W(s1) = 6$, $W(s2) = 15$. Puis ils ont utilisé la distance de Manhattan depuis n à t, notée $P(n)$, définie comme suit ; pour chaque jeton j, soit $col(j)$ l'écart en valeur absolue entre les rangs des colonnes où se trouve j dans p et dans t et soit $lig(j)$, l'écart en valeur absolue entre les rangs des lignes où se trouve j dans p et dans t ; $P(n)$ est calculé comme la somme des nombres $col(j)$ et $lig(j)$ pour tous les jetons j ; par exemple, pour la figure 3, $P(s1) = 7$, $P(s2) = 58$. En notant $h^*(n)$ la longueur d'un chemin minimal [18] de n à t dans le graphe d'états, on vérifie aisément que, pour tout n : $W(n) \leq P(n) \leq h^*(n)$. W et P sont donc des sous-estimations de la distance minimale depuis n à t.

16. Pour tout x, GR_x étant une arborescence de racine s, il existe un chemin unique de s à n dans GR_x.

17. Alternativement, on pourrait désirer exploiter des connaissances sur l'état courant n, non réductibles à une fonction de n seul (par exemple dépendant aussi de l'avancement de la recherche en cours).

18. S'il existe, sinon on pose : $h^*(n) = \infty$.

Algorithme 1.1 : Recherche heuristiquement ordonnée, type A^* de Hart, Nilsson et Raphaël

```
début
    front ← {s} ; réserve ← ∅ ; g_noté(s) ← 0
    répéter
        meilleurs ← tous sommets p de front tels que g_noté(p) + h(p) minimal
        dans front
        si il existe un état but dans meilleurs, soit m alors
            répéter
                écrire m ; m ← père(m)
            jusqu'à m = s
            écrire « s » ; stop
        m ← élément quelconque de focal
        front ← front − {m} ; réserve ← réserve + {m}
        pour tout n fils de m faire
            g_nouveau ← g_noté(m) + c(m, n)
            si n ∈ front et g_nouveau < g_noté(n) alors
                père(n) ← m ; g_noté(n) ← g_nouveau
            si n ∈ réserve et g_nouveau < g_noté(n) alors
                père(n) ← m ; g_noté(n) ← g_nouveau
                front ← front + {n} ; réserve ← réserve − {n}
            si n ∉ réserve et n ∉ front alors
                père(n) ← m ; g_noté(n) ← g_nouveau ; front ← front + {n}
    jusqu'à front = ∅
```

Dès qu'un sommet m est choisi pour être développé, il quitte le front pour être rangé en *réserve* (ligne 11) Soit $x+1$ le rang de développement ; tous les fils de m sont considérés (lignes 12-20) ; si l'un d'eux, soit n, est déjà en réserve, il y reste si $g_{nouveau} \geq g_{noté}(n)$, ce qui correspond à $g_{x+1}(n) \geq g_x(n)$ [donc $f_{x+1}(n) \geq f_x(n)$ du fait de la staticité de h] ; mais n quitte la réserve pour le front dès que $g_{nouveau} < g_{noté}(n)$, ce qui correspond à $g_{x+1}(n) < g_x(n)$ [donc : $f_{x+1}(n) < f_x(n)$]. Lorsqu'un état n est engendré pour la première fois il lui est assigné un père (ligne 20) ; les noms des pères sont gérés dans le tableau à une dimension $père(.)$; le père d'un état peut être remplacé (lignes 15, 17, 20) mais il demeure que : tout état engendré (présent dans front ou réserve) garde un père unique. Or s, qui n'a pas de père initialement, ne peut en recevoir par mise à jour (car dès le début, en ligne 2, on affecte à s une longueur nulle de chemin de s à s, qui ne peut être réduite puisque les coûts d'arcs sont supposés positifs) ; clairement, tout état engendré est joignable par un chemin partant de s, donc s est une racine. Donc, les successifs graphes de recherche sont des arborescences. L'algorithme s'arrête soit parce que le front est devenu vide (ligne 21), soit parce que parmi les sommets de front qui présentent la meilleure évaluation, il en est un qui satisfait la définition d'un état but (ligne 5).

1.3.2 Circonstances d'arrêt des A^* avec découverte de chemin minimal

Dans le cadre de graphes d'états arcs-valués quelconques[19], on note $h^*(n)$ la longueur d'un chemin minimal depuis n jusqu'à l'état but (ou, le cas échéant, jusqu'à l'ensemble des états buts) ; si un tel chemin n'existe pas, on pose : $h^*(n) = \infty$; $h^*(s)$ mesure la longueur d'une *solution minimale* : chemin minimal depuis l'état initial s jusqu'à un état but.

Sous le nom de *théorème d'admissibilité*, Nilsson et collègues ont établi [Hart *et al.*, 1968] [Nilsson, 1971, 1980] que l'algorithme A^* appliqué à un graphe d'états arc-valué G s'arrête assurément en trouvant un chemin minimal depuis l'état initial jusqu'à un état but dès lors que

1. G comporte au moins un chemin depuis l'état initial jusqu'à l'état but et
2. tout état de G a un nombre fini de fils et
3. il existe $\delta > 0$ tel que pour tout arc (m, n) de G : $c(m, n) \geq \delta$ et
4. pour tout état n de G : $h(n) \geq 0$ et
5. pour tout état n de G : $h^*(n) \geq h(n)$.

Les fonctions h qui satisfont la condition 5 sont dites *minorantes* (sur G).

Pour les graphes d'états de taquins, les conditions 2 et 3 sont évidemment satisfaites ; la condition 1 est satisfaite si et seulement si l'état but et l'état initial appartiennent à la même composante connexe ; on a indiqué précédemment que le graphe d'états d'un taquin $n \times n$ quelconque comporte 2 composantes connexes et qu'il est aisé de tester si 2 états donnés appartiennent à la même ; les heuristiques W et P satisfont la condition 5 (c'est le cas aussi de l'heuristique identiquement nulle). Dans ce cas, on dérive un théorème d'admissibilité spécifique à l'algorithme de coût uniforme.

Ainsi, en agrégeant un terme – $h(n)$ – qui minore la distance de n au but (i.e. $h(n)$ est une sous-estimation du *coût devant*) et un terme – $g_x(n)$ – qui mesure le chemin le plus court actuellement connu pour atteindre n (i.e. une surestimation du *coût derrière*), la fonction d'évaluation assure à l'algorithme la capacité de découvrir un chemin minimal.

1.3.3 Comparaison entre algorithmes A^* selon les heuristiques minorantes employées

Soient h_1 et h_2 deux heuristiques définies pour un même graphe d'états G, minorantes sur G ; h_2 est dite *mieux informée* que h_1 sur G si et seulement si : pour tout état n de G qui n'est pas état but, on a $h_1(n) < h_2(n)$. Noter : l'inégalité est stricte. Par exemple, pour les taquins, on peut comparer partiellement les fonctions W, P et nulle ; les fonctions W et P sont, chacune, mieux informées que la fonction nulle ; noter : on a bien $W \leq P$, mais il peut exister n tel que $W(n) = P(n)$, sans que n soit un état but.

Sous le nom de *théorème d'optimalité*[20], Nilsson et collègues ont établi [Hart *et al.*, 1968] [Nilsson, 1971, 1980] que si h_1 et h_2 sont 2 heuristiques définies pour un même

19. Nous avions déjà introduit h^*, mais dans le contexte particulier des graphes d'états de taquins.
20. Nous préférons l'appellation : *théorème de comparaison.*

graphe d'états G, minorantes sur G, telles que h_2 est mieux informée que h_1 sur G, tous les états développés par A^* muni de h_2 sont développés par A^* muni de h_1. Noter : ce résultat n'assure pas que l'algorithme A^* doté de h_1 procède à davantage (ou autant) de développements que l'algorithme A^* doté de h_2 ; car il pourrait arriver qu'un même état soit développé davantage de fois par h_2 que par h_1.

1.3.4 Comparaison d'algorithmes A^* lorsque l'heuristique mieux informée est monotone

Une heuristique h définie sur un graphe d'états G est dite *monotone* sur G si et seulement si, pour tout état m de G et tout fils n de G, on a $h(m) \leq c(m, n)+h(n)$. Cette *inégalité triangulaire* exprime une forme de cohérence : l'estimation du chemin le plus court depuis m jusqu'aux buts, $h(m)$, est majorée par l'estimation du chemin le plus court depuis n jusqu'aux buts, $h(n)$, augmentée du coût de l'arc (m, n). L'heuristique nulle est monotone sur tout graphe à coûts d'arcs ≥ 0. On vérifie aisément que les heuristiques W et P définies pour les graphes de taquin sont monotones.

On établit aisément que : si A^* exploite une heuristique monotone, alors tout sommet développé ne peut l'être qu'une fois. Ce résultat permet d'affiner le théorème d'optimalité lorsque h_2 est monotone : l'algorithme A^* doté de h_1 procède à davantage (ou autant) de développements que l'algorithme A^* doté de h_2.

Par exemple, pour des problèmes de taquins, un algorithme A^* employant l'heuristique nulle (c'est-à-dire l'algorithme du coût uniforme) développe un sous-ensemble D^0 des sommets du graphe d'états, une fois chacun seulement, car la fonction nulle est monotone ; l'algorithme A^* employant l'heuristique P (monotone aussi) développe un sous-ensemble D^P des sommets du graphe d'états, une fois chacun seulement aussi, avec $D^P \subseteq D^0$; en pratique, on observe que l'algorithme A^* employant l'heuristique P est beaucoup plus efficace, quant à la mémoire et au temps consommés, que l'algorithme du coût uniforme. Quoique P ne soit pas mieux informée que W, on constate expérimentalement que P est généralement beaucoup plus sobre que W en place et temps.

1.4 Variantes de programmation de A^*

Pendant la quinzaine d'années qui ont suivi son invention, l'algorithme A^* a inspiré diverses variantes de codage et de mise en œuvre visant à améliorer ses performances en termes d'espace occupé et de rapidité de calcul. Les algorithmes B [Martelli, 1977], C [Bagchi et Mahanti, 1983], B' [Mero, 1984] exploitent des fonctions d'évaluation de même forme qu'A^* (avec heuristiques h minorantes) et donnent lieu à des théorèmes d'admissibilité énoncés dans les mêmes termes que celui établi pour A^*. Malgré des contributions ingénieuses, on ne constate pas d'avancée décisive quant à la capacité de résoudre des problèmes de référence (taquins notamment) de taille significative : A^* et ses succédanés présentent le défaut de maintenir en mémoire tous les états engendrés. Par exemple, lorsque A^*, muni de P, résout le problème de la figure 3, quoiqu'il il s'agisse d'un problème dont la solution minimale ne compte que 9 coups, il exécute 18 développements et garde les 45 états en résultant. A la fin, le front compte 27 états, la

réserve en compte 18. Ces nombres sont liés à l'hypothèse suivante : pour développer tout état on considère d'abord le fils obtenu en déplaçant la case vide vers le haut puis la gauche, puis la droite, puis le bas.

Pour éviter la saturation de la mémoire centrale, Chakrabarti et ses collègues ont conçu MA^* (*Memory-bounded* A^*), comme un algorithme A^* qui s'accommode de l'espace-mémoire disponible [Chakrabarti *et al.*, 1989] ; lorsque la mémoire vient à manquer, MA^* libère l'espace occupé par les états qui présentent les pires évaluations, mais conserve certaines caractéristiques des branches élaguées de manière à pouvoir les reconstituer si la suite de l'exploration l'exige ; MA^* présente le très grand avantage de ne pas bloquer la recherche mais le temps d'exécution peut devenir prohibitif.

1.5 Recherche heuristiquement ordonnée A^* bidirectionnelle

Pour chercher un chemin minimal dans un graphe d'états G depuis un état s vers un état t précis, très tôt est apparue l'idée de contrôler les fonctionnements imbriqués [21] de deux algorithmes A^*. L'un travaille depuis s vers t dans le graphe G ; l'autre travaille depuis t vers s dans le graphe $G_{inverse}$ obtenu à partir de G en inversant le sens de tous les arcs. Un module supervise l'allocation de cycles d'expansion à l'un ou l'autre [22] des deux A^* ; il surveille aussi l'occurrence d'une intersection entre les deux graphes de recherche courants.

Pohl a été le principal pionnier de cette approche, avec, notamment, l'algorithme $BHPA$: *Bi-directional Heuristic Path Algorithm* [Pohl, 1971] ; l'algorithme A^* qui travaille de s vers t (respectivement de t vers s) dispose d'une heuristique minorante et monotone notée sh (respectivement : th) et maintient un front de recherche $sfront$ (respectivement : $tfront$) ; pour tout n de $sfront$ (respectivement : $tfront$) l'estimation heuristique $h(n)$ vaut $sh(n)$ [respectivement : $th(n)$]. Au plan formel, Pohl établit [23] que $BHPA$ trouve bien un chemin minimal dès lors qu'il s'arrête ; au plan pratique, il constate que la focalisation commandée par l'heuristique des deux côtés de la recherche, peut conduire à des arrêts tardifs.

De Champeaux et Sint ont proposé l'algorithme $BHFFA$: *Bi-directional Heuristic Front-to-Front Algorithm* [de Champeaux et Sint, 1977] comme un perfectionnement de $BHPA$. $BHFFA$ dispose d'une heuristique E pour estimer chaque fois qu'il en est besoin la distance entre un état p du $sfront$ courant et un état q du $tfront$ courant. Pout tout n de sfront, l'algorithme A^* qui cherche depuis s vers t, calcule l'estimation heuristique $h(n)$ comme le minimum, étendu à tous les états y du $tfront$ courant, de $E(n,y) + tg(y)$, où $tg(y)$ est la longueur du chemin minimal couramment connu depuis t jusqu'à y ; noter : $h(n)$ n'est pas calculé comme $E(n,t)$. Le fonctionnement de l'algorithme A^* qui cherche dans l'autre sens est symétrique. Au plan formel, De Champeaux et Sint établissent [24] que $BHFFA$ s'arrête en trouvant un chemin minimal

21. Le processeur, supposé unique, alloue aux deux A^* des périodes disjointes de fonctionnement.

22. La première idée est d'accorder le contrôle, à chaque cycle de développement, à celui des deux algorithmes dont le front présente un état de meilleure évaluation.

23. Pohl suppose que les coûts d'arcs sont positifs, ainsi que les fonctions sh et th.

24. En respectant les 5 conditions introduites dans l'énoncé du théorème d'admissibilité des A^*.

entre s et t ; au plan pratique, les auteurs indiquent avoir remédié au principal défaut de $BHPA$: les états où se rencontrent les deux graphes de recherche sont maintenant à peu près équidistants de s et t, mais ils soulignent que le temps de calcul de l'estimation heuristique est très élevé.

Kaindl et Khorsand ont développé une version bidirectionnelle d'une version simplifiée de MA^* (voir plus haut) qui présente l'intérêt de ne pouvoir être bloquée par manque d'espace-mémoire [Kaindl et Khorsand, 1994]. La recherche bidirectionnelle (ou multidirectionnelle si plusieurs buts) pourrait connaître un nouvel essor dans un environnement moderne bi ou multi processeur.

1.6 Relaxations de A^* : algorithmes sous-admissibles

On peut songer à assouplir certaines contraintes attachées aux A^* de Nilsson pour plusieurs raisons. Des heuristiques peuvent inspirer confiance sans être strictement minorantes, ou sans qu'on sache prouver qu'elles le sont. Pour nombre d'applications, il importe davantage de trouver sans trop d'effort (espace et temps consommés) une assez bonne solution plutôt que d'obtenir à tout prix une solution optimale. La notion même d'optimalité est à relativiser : elle réfère à un modèle du problème réel considéré qui est plus ou moins schématique ; à quoi bon chercher un chemin de longueur assurément minimale si les coûts des arcs représentatifs des changements d'états (et la distinction même entre les états) ne sont que des approches du problème réel ? Par exemple : un itinéraire estimé minimal sur une carte routière peut ne pas l'être sur le terrain, et vice-versa ; en outre, un chemin moins court que l'optimum peut être avantageux quant à un autre critère.

Tout en suivant le code algorithmique de A^*, Harris a proposé d'employer des heuristiques h qui ne satisfont pas la relation de minorance ($h < h^*$), mais la relation relâchée : $h \leq h^* + e$, où e est une constante positive (Harris 1974). En respectant les 4 premières conditions introduites dans l'énoncé du théorème d'admissibilité, on établit aisément que l'algorithme ainsi piloté découvre un chemin depuis l'état initial s jusqu'à un état but, de longueur inférieure ou égale à $h^*(s) + e$. L'algorithme est dit *sous-admissible* et la solution *sous-minimale*.

Pearl et Kim, ont conservé la contrainte de minorance pour les heuristiques h, mais ont proposé d'assouplir le mode de sélection de l'état à développer [Pearl et Kim, 1982]. Cet assouplissement permet de faire intervenir dans le choix de l'état à développer, un critère de préférence lié à l'application, qui vient seconder l'estimation relative aux longueurs de chemins que donne la fonction d'évaluation. Pour engager le $x+1^e$ développement, ils ont accepté le choix, dans le front courant, de n'importe quel état p tel que : $f_x(p) \leq \min_{q \in front} (1+\varepsilon) f_x(q)$ où ε est une constante positive.

En supposant satisfaites les 4 premières conditions introduites dans l'énoncé du théorème d'admissibilité des A^*, on établit aisément que, dans ces conditions, l'algorithme[25] découvre un chemin depuis l'état initial s jusqu'à un état but, de longueur inférieure ou égale à $(1+\varepsilon)h^*(s)$. C'est une autre forme de sous-admissibilité de l'algorithme et de sous-minimalité de la solution.

25. Qu'on peut noter : A^*_ε. Lorsque ε est nul, on retrouve A^*.

Ghallab et Allard ont autorisé le même mode assoupli de sélection de l'état à développer que Pearl et Kim, mais en admettant simultanément des heuristiques h non minorantes satisfaisant la relation : $h \leq (1+\alpha)h^*$ où α est une constante positive [Ghallab et Allard, 1983]. En supposant satisfaites les 4 premières conditions introduites dans l'énoncé du théorème d'admissibilité des A^* , on établit aisément que l'algorithme résultant [26] découvre un chemin depuis l'état initial s jusqu'à un état but, de longueur inférieure ou égale à $(1+\alpha)(1+\varepsilon)h^*(s)$. C'est encore une forme de sous-admissibilité de l'algorithme et de sous-minimalité de la solution.

On montrera plus loin que la perspective ouverte par Harris, Pearl et Kim, Ghallab et Allard peut être sensiblement élargie.

1.7 Enfin IDA^* vint

L'algorithme A^* et tous les successeurs que nous avons mentionnés jusqu'ici ont été mis à l'épreuve sur des jeux d'essai, limités, de taquin 3×3, en employant l'heuristique P. Mais en 1985 encore, pour aucun d'entre eux, il n'avait pu être montré qu'il était capable de résoudre la totalité des problèmes de taquin 3×3. A l'époque, Korf a rapporté qu'ayant tiré au sort 100 problèmes [27], de taquin 4×4 et tentant de leur appliquer A^*, muni de P, sur un DEC 2060, il ne parvint à résoudre aucun d'entre eux : pour les 100 cas, malgré une programmation soignée, il constata un dépassement de capacité-mémoire après création d'environ 30000 états. C'est ce qui l'amena à concevoir l'algorithme IDA^* : *Iterative-Deepening* A^* [Korf, 1985b,a].

IDA^* utilise des fonctions d'évaluation de même forme que celles employées par A^* : $f_x(n) = g_x(n) + h(n)$, où $g_x(n)$ mesure le chemin minimal connu depuis s à n à l'issue du développement de rang x, tandis que $h(n)$ est une estimation heuristique statique et minorante. Mais IDA^* ne suit plus la stratégie d'expansion de A^*. Pour A^*, développer un état c'est produire et évaluer *tous* ses fils ; à chaque cycle, A^* privilégie le développement d'un état présentant *la plus petite* évaluation parmi tous ceux du front courant ; les graphes de recherche successifs sont des arborescences. Pour IDA^* développer un état c'est produire et évaluer *un seul* de ses fils, différent à chaque nouvelle demande de développement ; IDA^* privilégie le développement du *dernier état* produit, sous réserve que son évaluation ne dépasse pas un certain seuil ; de la sorte IDA^* se comporte localement comme un algorithme *en profondeur d'abord* (*depth-first*), d'où l'expression : *iterative deepening* dans le sigle IDA^* ; les graphes de recherche successifs sont des chemins (arborescences particulières) ; on peut considérer que pour IDA^* le front de A^* est réduit à l'extrémité finale du chemin de recherche courant. IDA^* est décrit à l'algorithme 1.2.

Initialement (ligne 2) le seuil S vaut $h(s)$. Le front de recherche est géré comme une liste de couples ; il ne contient au début que le couple (s 1). IDA^* progresse par cycles (boucle entre les lignes 3 à 16) en cherchant à *développer partiellement* l'état qui figure dans le premier couple du front ; chaque couple est de la forme (n k) où n est un

26. Noté A_ε par les auteurs ; nous préférons : A^α_ε. On retrouve A^*_ε si $\alpha = 0$, A^* si $\alpha = \varepsilon = 0$.

27. Listés dans [Korf, 1985a] ; notons-les : $K4,1$ à $K4,100$. L'état but est standard : case vide en haut à gauche, puis jetons 1 à 15 de gauche à droite et de haut en bas : voir l'état t en figure 3

Algorithme 1.2 : Recherche heuristiquement ordonnée, type IDA^* de Korf

début
  $front \leftarrow ((s\ 1))$; $S \leftarrow h(s)$; *seuil-futur* $\leftarrow \infty$
  répéter
    si $front = \varnothing$ **alors**
      $front \leftarrow ((s\ 1))$; $S \leftarrow$ *seuil-futur* ; *seuil-futur* $\leftarrow \infty$
    $m \leftarrow 1^{er}$ élément de la tête de $front$; $p \leftarrow 2^{e}$ élément de la tête de $front$
    si *m est un état but* **alors**
      edition-chemin-à-rebours ; **stop**
    engendrer le premier fils éventuel n de m, de rang $\geq p$, soit q, non présent dans front
    si *n existe et* $f(n) \leq S$ **alors**
      remplacer $(m\ p)$ par $(m\ q+1)$ en tête de front ; ajouter $(n\ 1)$ en tête de front
    si *n existe et* $f(n) > S$ **alors**
      seuil-futur $\leftarrow \min(seuil\text{-}futur, f(n))$; remplacer $(m\ p)$ par $(m\ q+1)$ en tête de front
    si *n n'existe pas* **alors**
      supprimer la tête de front
  jusqu'à $front = \varnothing$ *et seuil-futur* $= \infty$

Algorithme 1.3 : édition-chemin-à-rebours

début
  répéter
    $m \leftarrow 1^{er}$ élément de la tête de $front$; écrire m ; supprimer la tête de front
  jusqu'à $front = \varnothing$

FIGURE 4 – Pour ce problème, IDA^* n'a besoin que de 18 états en mémoire contre 290 pour A^*

état tandis que k est un *rang de filiation*; k représente le rang du prochain [28] fils de n à engendrer. Le premier couple de front étant (s 1), le premier *développement partiel* concernera s : il s'agira d'engendrer son premier fils. En ligne 6, le premier couple de front est noté (m p); si m est un état but, IDA^* édite à rebours le chemin-solution trouvé (lignes 8 et 1-4 de l'algorithme 1.3); sinon IDA^* tente de créer le k^e fils de m, pourvu qu'il ne soit pas déjà dans le front courant, sinon le $k+1^e$ etc. (ligne 9). Si un fils nouveau n est trouvé et si son évaluation $f(n)$ est inférieure au seuil courant S (ligne 10), IDA^* poursuit les développements en profondeur tant que l'évaluation de l'état le plus profond est inférieure ou égale à S; si cette évaluation dépasse S, IDA^* remet en cause l'ultime production d'état : il remonte jusqu'au premier état précédent qui permet la production d'un nouveau fils et tente à nouveau une expansion en profondeur. S'il ne reste plus de choix, IDA^* lance une nouvelle passe depuis s, mais en assignant maintenant à S le minimum des valeurs d'évaluation ayant dépassé la valeur en vigueur jusqu'ici. IDA^* s'arrête lorsqu'il advient que le sommet à développer est l'état but.

Pour des graphes d'états finis, avec les mêmes hypothèses pour les fonctions d'évaluation et heuristiques que celles retenues par Nilsson pour A^*, Korf montre que IDA^* est admissible et ne développe que des états développés par A^* pour les graphes d'états finis et avec quelques autres restrictions, satisfaites par les taquins. Ces contraintes peuvent être largement relaxées, voir : [Farreny, 1995, 1997b,a, 1999].

L'énorme avantage de $IDA*$ sur A^* c'est qu'à tout instant il ne conserve en mémoire que les états d'un chemin joignant s à l'état le plus profond du moment, chemin dont la longueur est majorée par $h^*(s)$; si, par hypothèse (banalement satisfaite), il existe un minorant $d > 0$ du coût de tous les arcs (par exemple, pour les taquins $n \times n$: $d = 1$), le nombre d'états de ce chemin est majoré par $h^*(s)/d$; IDA^* est donc de complexité linéaire en ce qui concerne l'espace requis. Par exemple, pour trouver la solution minimale (18 coups) du problème représenté en figure 4, A^* maintient en mémoire un graphe de recherche qui compte jusqu'à 290 états (au moment de l'arrêt); tandis que IDA^* ne maintient qu'un chemin de recherche qui comptera au plus 18 états. En contrepartie, chaque fois que le seuil est augmenté, IDA^* entreprend une nouvelle passe d'exploration en profondeur au cours de laquelle il refait tout ou partie du travail accompli à la passe précédente (tout, si la nouvelle passe n'est pas la dernière), ce qui peut prendre beaucoup de temps.

En 1985 donc, grâce à IDA^* muni de $h = P$, Korf détermine une solution minimale pour $K4,1$ à $K4,100$; la longueur moyenne des solutions minimales est environ 53. La

28. Il est supposé que pour tout état, le nombre de fils est fini. A chaque fils d'un état est associé un rang de génération.

résolution sur DEC 2060 consomme environ une demi-heure par problème. En 1993, grâce à IDA^* muni de $h = P$, Reinefeld a pu traiter [29] tous les problèmes de taquin 3×3 à but dit *standard* : la case vide est en haut à gauche, tandis que les jetons 1 à 8 sont disposés dans cet ordre, de gauche à droite et de haut en bas [Reinefeld, 1993]. Pour chacun des $9!/2$ problèmes résolubles il a produit les solutions de longueur minimale (en fait : non seulement une mais toutes). Avec les ressources informatiques dont il disposait alors, il n'aurait pu effectuer ce travail s'il s'était contenté de l'heuristique nulle ou de l'heuristique W. Grâce à cette investigation, il a établi que le maximum des solutions minimales vaut 31 ; en observant que les solutions des problèmes à but non standard ne peuvent être plus longues, on conclut que le *diamètre* du graphe du taquin 3×3 est 31 (le *diamètre* d'un graphe est le maximum du nombre d'arêtes des chaînes minimales entre tous couples de sommets).

L'un des défauts de IDA^*, contrepartie de son extrême sobriété en espace, est de ne pas mettre à profit les capacités en mémoire des ordinateurs disponibles, tandis qu'il répète de nombreux calculs. IDA^* a inspiré une nouvelle vague d'algorithmes, notamment : $MREC$ [Sen et Bagchi, 1989], DFS^* [Rao *et al.*, 1991], $RFBS$ [Korf, 1993], IE [Russell, 1992], $BIDA^*$ [Manzini, 1995], $DBIDA^*$ et $BDBIDA^*$ [Eckerle et Schuierer, 1995]. Des améliorations de performances parfois très importantes ($BIDA^*$ spécialement) ont été obtenues sur le taquin 4×4, avec P comme heuristique.

Néanmoins, quand il s'agit d'attaquer des problèmes de taquin 5×5, IDA^* et ses épigones (toujours guidés par $h = P$) s'avèrent beaucoup trop lents. Il convient de souligner que le pas suivant dans la maîtrise du taquin a été accompli, non pas en remplaçant IDA^*, mais en cherchant à dépasser l'heuristique P, comme expliqué ci-après.

s

17	1	20	9	16
2	22	19	14	5
15	21		3	24
23	18	13	12	7
10	8	6	4	11

t

	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

FIGURE 5 – Problème $K5,1$: passer de s à t minimalement. Solution : 100 coups.

1.8 Inventer des heuristiques en affinant celles déjà connues

[Korf et Taylor, 1996] ont expérimenté IDA^* muni d'une nouvelle heuristique, grâce à laquelle, ils ont réussi à trouver des solutions minimales pour 9 problèmes de taquin 5×5 parmi une série de 10 tirés au hasard, notés ci-après $K5,1$ à $K5,10$ (l'état but est standard). Ils ont utilisé une station Sun Ultra Sparc, 70 fois plus rapide que la DEC 2060 de Reinefeld. Le problème le plus vite résolu (représenté en figure 5) exige 100 coups ; le plus lentement résolu exige 113 coups ; la résolution du 10^e a été interrompue après 1 mois de calcul. La longueur moyenne des 9 solutions minimales trouvées est environ 101.

29. Sur une Sun Sparc Station.

On présente ci-après une version simplifiée, notée h_K, de l'heuristique réellement employée par Korf. On pose : $h_K = P + LC + C$ où P est la distance de Manhattan ; voici les principes de calcul des composantes LC et C.

LC (***Linear-Conflict Heuristic***) est décrite dans [Hansson *et al.*, 1992] mais on trouve déjà l'idée dans [Gardner, 1979]. Si dans l'état p, 2 jetons sont dans la rangée (ligne ou colonne) qui leur est destinée dans t, mais qu'ils sont en ordre inverse, il faudra nécessairement que l'un des 2 jetons quitte la rangée pour « laisser passer » l'autre puis y revienne. On note que ces deux coups supplémentaires ne sont pas considérés lorsqu'on évalue P (ils ne pèsent rien pour P). Par exemple, en figure 6, l'état s_1 présente plusieurs conflits en ligne 2 et colonne 2 ; naturellement il faut gérer les rapports entre conflits pour ne pas surévaluer leur coût.

s_1

8	7	6
5	4	3
2	1	

t

	1	2
3	4	5
6	7	8

s_2

3	1	4
	2	5
8	7	6

FIGURE 6 – Pour illustrer le calcul de LC et C ; t est l'état but ; s_1 et s_2 sont deux états initiaux.

C (***Corner-Tiles Heuristic***) est décrite dans [Korf, 1997]. Dans l'état s_2 de la figure 6, le jeton 4 occupe un coin qui ne lui est pas destiné ; mais le jeton 1 (« voisin de coin »), lui, est à sa place ; pour que le jeton 4 puisse quitter le coin et que le jeton 2 puisse y arriver, il faudra que 1 bouge, ce qui coûtera 2 coups au moins (pour aller et venir). Korf compte de même pour 3 coins sur 4 : il ne tient pas compte du coin haut-gauche (pour éviter des tests complexes). Sur un taquin $n \times n$ avec $n > 3$, ce système peut peser jusqu'à 12 coups ; 6 coups seulement sur un taquin 3×3 car un voisin d'angle ne doit pas intervenir 2 fois. En outre, si on a déjà pris en compte la contribution de LC, il faut éviter de recompter des coups supplémentaires pour les jetons voisins d'angle déjà impliqués dans des conflits linéaires ; ainsi, pour l'état s_2, le jeton 7 est impliqué dans un conflit sur la ligne 3.

Avec quelques précautions quant aux interférences, on définit un peu plus précisément LC et C, donc $h_K = P + LC + C$, de telle sorte que h_K est assurée être minorante. En outre : pour tout état p, $P(p) \leq h_K(p)$. Au sens strict de Nilsson et collègues, h_K n'est pas mieux informée que P : on ne peut donc invoquer le théorème d'optimalité.

Néanmoins, l'expérience confirme l'attente intuitive : de manière générale, h_K conduit à moins de développements et, malgré le surcoût d'évaluation, consomme moins de temps.

1.9 Combiner les estimations heuristiques obtenues pour des sous-problèmes

En vue de disposer d'estimations heuristiques de valeurs plus élevées que P, mais toujours minorantes, on peut appliquer la technique des *Pattern databases* [Culberson

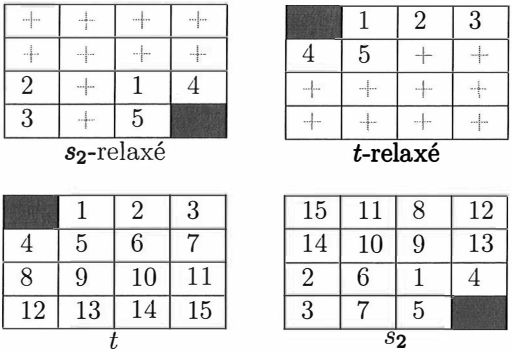


FIGURE 7 – $\mathcal{P}$: passer minimalement de s_2 à t.
$\mathcal{P}_{1-5}$: passer minimalement de s_2 relaxé à t relaxé.

et Schaeffer, 1998], améliorée ensuite sous l'appellation : *Disjoint pattern databases* [Korf, 2000] [Korf et Felner, 2002] [Felner *et al.*, 2004].

Décrivons-en le principe en examinant un des problèmes représentés en figure 3, repris en figure 7 à droite : passer minimalement de s_2 à t. Notons $\mathcal{P}$ ce problème. On a indiqué précédemment que la résolution de $\mathcal{P}$ exige 80 coups. Considérons la relaxation de $\mathcal{P}$ représentée en figure 7 à gauche, notée $\mathcal{P}_{1-5}$: il s'agit de déplacer seulement les jetons 1 à 5, depuis leurs positions dans s_2 jusqu'à celles qu'ils occupent dans t, mais en négligeant les différences d'identité des 10 autres jetons. On a : $P(s_2) = 58$. Notons $P_{1-5}(s2)$ la part de $P(s_2)$ qui concerne les déplacements des jetons 1 à 5, soit $P_{1-5}(s_2) = 20$. On définit de même les problèmes $\mathcal{P}_{6-10}$, $\mathcal{P}_{11-15}$, les estimations $P_{6-10}(s_2)$ et $P_{10-15}(s_2)$. On a : $P_{6-10}(s_2) = 14$, $P_{10-15}(s_2) = 24$, $P(s_2) = P_{1-5}(s2) + P_{6-10}(s2) + P_{10-15}(s2) = 58$.

Puisque 10 jetons sont interchangeables, le graphe d'états de $\mathcal{P}_{1-5}$ compte 10! fois moins d'états que celui de $\mathcal{P}$, soit $N = 2\,882\,880$ états ; $\mathcal{P}_{1-5}$ est donc considérablement plus facile à résoudre que $\mathcal{P}$. On a signalé qu'en 1985, Korf saturait la mémoire de son DEC 2060, avec seulement 30 000 états. Une quinzaine d'années plus tard, des ordinateurs relativement ordinaires peuvent emmagasiner plusieurs milliers de fois plus d'états en mémoire centrale. Ceci autorise d'autres manières de procéder. Supposons cette résolution exécutée[30], notons $h_{1-5}(s_2)$ la distance minimale trouvée entre s_2-relaxé et t-relaxé, puis $\underline{h}_{1-5}(s_2)$ *le nombre de coups qui concernent les seuls jetons* 1 *à* 5 ; nécessairement : $\underline{h}_{1-5}(s_2) \geq P_{1-5}(s_2)$. De même on détermine $\underline{h}_{6-10}(s_2)$ et $\underline{h}_{11-15}(s_2)$. On définit $\underline{h}(s_2) = \underline{h}_{1-5}(s_2) + \underline{h}_{6-10}(s_2) + \underline{h}_{11-15}(s_2)$; on a : $\underline{h}(s_2) \geq P(s_2)$. Vu que chaque coup ne bouge qu'un seul jeton, $\underline{h}$ est manifestement minorante, mais supérieure (au sens large) à P.

Selon le schéma précédent, calculer $\underline{h}(s_2)$ exigerait 3 résolutions subalternes : celles des sous-problèmes $\mathcal{P}_{1-5}$, $\mathcal{P}_{6-10}$ et $\mathcal{P}_{11-15}$. Il est plus judicieux de procéder comme suit. On applique l'algorithme en largeur d'abord à $\mathcal{P}_{1-5}$, mais en partant en arrière, depuis t-relaxé ; quand l'état s_2-relaxé apparaîtra on pourra enregistrer $\underline{h}_{1-5}(s_2)$; en fait, à chaque fois qu'un nouvel état n est engendré on l'enregistre accompagné de la valeur

30. On pourrait recourir à IDA^*, muni de P, ou mieux : $P + LC + C$.

$\underline{h}_{1-5}(n)$; on poursuit jusqu'à ce que tous les états relaxés relatifs aux jetons 1 à 5 aient été engendrés. On obtient ainsi une base de données heuristiques $\mathcal{B}_{1-5}$; on constitue de même $\mathcal{B}_{6-10}$ et $\mathcal{B}_{11-15}$. Ces bases sont constituées et installées en mémoire vive, préalablement à tout traitement de taquin 4×4. Pour ces 3 bases il faut $3N$ entrées, soit 8648640. Ce qui est couramment acceptable aujourd'hui. Par la suite, les consulter équivaut à disposer de l'heuristique minorante $\underline{h}$.

Avec IDA^* doté de la technique des *Pattern databases* (moins puissante [31] que celle des *Disjoint pattern databases* présentée ci-dessus) Korf a pu résoudre 10 problèmes de Rubik's Cube tirés au sort ; les solutions comptaient entre 16 et 18 mouvements de faces [Korf, 1997].

1.10 Formaliser pour ouvrir d'autres champs d'application et voies de résolution

Les algorithmes A^* et IDA^*, ci-dessus présentés en substance, ainsi que ceux brièvement caractérisés (algorithme de Harris, A^*_ε, A^α_ε) [32] peuvent être vus comme des cas particuliers d'une vaste famille définie et analysée dans [Farreny, 1995] sous le nom de *algorithmes* ρ. Pour identifier ces algorithmes cinq voies de généralisation ont été suivies. On introduit ci-après, très succinctement, quatre d'entre elles.

1.10.1 Graphes d'états : on peut être moins exigeant

Dans nombre d'exposés traitant de recherche heuristiquement ordonnée, les contraintes appliquées aux graphes d'états pour assurer que des algorithmes s'arrêtent en trouvant une solution minimale (propriété d'admissibilité) ou sous-minimale (propriété de sous-admissibilité), sont excessives. Par exemple, pour que A^* s'arrête en découvrant une solution minimale, il n'est pas indispensable de satisfaire les conditions 3 et 4 posées dans l'énoncé classique du théorème d'admissibilité, c'est-à-dire l'énoncé publié dans : [Nilsson, 1971, 1980] [Barr *et al.*, 1981] [Rich, 1983] [Winston, 1984] [Pearl, 1984] [Shirai et Tsujii, 1984] [Laurière, 1986] [Farreny et Ghallab, 1987] [Shapiro, 1987] [Nilsson, 1998] [Russell et Norvig, 2003], en vérité, les valeurs d'arcs peuvent être des réels quelconques pourvu que :

a) pour tout réel M, au-delà d'un certain nombre d'arcs, la longueur de tout chemin *élémentaire* (i.e., ne comportant jamais deux occurrences d'un même sommet), issu de s est supérieure à M et

b) il n'existe pas de *circuit* (i.e., de chemin dont le sommet final est identique au sommet initial), de longueur strictement négative.

1.10.2 Longueur d'un chemin : on peut sortir des sentiers battus

Très généralement, *la longueur d'un chemin* est comprise comme la somme des coûts des arcs qui le composent : notons-là L_{add}. Néanmoins, Pearl a envisagé une longueur

31. On emploie alors $h = \max(h_{1-5}, h_{6-10}, h_{11-15})$ plutôt que $h = \underline{h} = \underline{h}_{1-5} + \underline{h}_{6-10} + \underline{h}_{11-15}$.

32. Ainsi que plusieurs des algorithmes cités (B, C, D...) et d'autres non mentionnés jusqu'ici, tels : BF^* [Pearl, 1984], A^{**} [Dechter et Pearl, 1985, 1988], SDW [Köll et Kaindl, 1992].

de chemin, calculée comme le maximum des coûts de ses arcs [1984] ; Yager [1986], ainsi que Dubois, Lang et Prade [1987] ont calculé la longueur d'un chemin comme le minimum des coûts de ses arcs ; Gonella a agrégé les coûts d'arcs par multiplication [1989]. A regarder de plus près, il apparaît que la recherche heuristiquement ordonnée peut trouver de nouveaux champs d'application en substituant à L_{add} une définition de la longueur d'un chemin beaucoup plus générale.

L'opération + et l'ensemble $R =]-\infty, +\infty[$, impliqués dans la définition de L_{add}, peuvent être respectivement remplacés par n'importe quelle opération binaire Θ et n'importe quel sous-ensemble V de R, pourvu que V et Θ constituent un *monoïde.* Rappelons que (V, Θ) est un *monoïde* si et seulement si V est fermé pour Θ, Θ est associative et admet un élément neutre dans V. Soit c une fonction qui associe à chaque arc u un coût $c(u)$ dans R. On définit comme *longueur associée au monoïde* (V, Θ) *et à la fonction* c, toute fonction L qui respecte les contraintes suivantes :

1. pour tout arc u : $L(u) = c(u)$,
2. L(suite vide d'arcs) $= e_n$, élément neutre de (V, Θ),
3. pour toutes suites d'arcs S' et S'', L(concaténation de S' et S'') $= \Theta(L(S'), L(S''))$.

En prenant $V = R$ ou R^+ ou Q ou Q^+ ou Z ou N, et $\Theta = +$, on retrouve les formes courantes de L_{add}. Avec certains sous-ensembles V de R, on peut choisir $\Theta(x, y) = x.y$ ou $\min(x, y)$ ou $\max(x, y)$ ou $\sqrt{x^2 + y^2}$, etc. ; pour d'autres exemples voir [Farreny, 1995]. Les théorèmes d'admissibilité et sous-admissibilité présentés précédemment tiennent lorsqu'une longueur de ce type général est substituée à L_{add}.

1.10.3 États à développer : d'autres choix éclairés

On a vu que A^* choisit dans le front courant le prochain état à développer, en suivant le principe dit du *meilleur d'abord.* Les algorithmes de Pearl-Kim et Ghallab-Allard permettent d'autres choix parmi les sommets du front dont l'évaluation ne dépasse pas de plus de ε % l'évaluation minimale, tout en garantissant que la longueur du chemin trouvé ne dépassera pas de plus de ε % la longueur d'un chemin minimal. On peut considérer un procédé nettement plus général : pour engager le x^e développement, on autorise de choisir dans le front courant n'importe quel état p tel que : $f_{x-1}(p) \leq E(\min_{q \in front} f_{x-1}(q))$.

On établit que, si la fonction E satisfait certaines conditions [Farreny, 1995, 1999], dont la croissance au sens large, la longueur (possiblement généralisée comme précédemment) du chemin trouvé est inférieure ou égale à $E(\mathcal{L})$ où $\mathcal{L}$ est la longueur d'un chemin minimal.

En prenant pour E la fonction identité I, on retrouve le procédé de choix suivi par A^*, ainsi que l'assurance que la longueur du chemin trouvé vaut $\mathcal{L}$. En prenant pour E la fonction $(1+\varepsilon)I$, on retrouve le procédé de choix suivi par les algorithmes de Pearl-Kim et Ghallab-Allard, ainsi que l'assurance que la longueur du chemin trouvé est inférieure ou égale à $(1+\varepsilon)\mathcal{L}$. Mais on peut aussi prendre, par exemple, $E = (1+\varepsilon)I + e$ ou $E = \sqrt{I \times I + e}$.

1.10.4 Heuristiques : d'autres libertés avec garanties

On a noté que A^* emploie des heuristiques h minorantes et statiques. On a noté aussi que l'algorithme de Harris accepte une heuristique statique majorée par $h^* + e$, et que l'écart entre la longueur du chemin trouvé et la longueur d'un chemin minimal ne peut dépasser e. On a noté enfin que l'algorithme de Ghallab-Allard accepte une heuristique statique qui ne dépasse pas h^* de plus de $\alpha\%$, et que la longueur du chemin trouvé ne dépasse pas la longueur d'un chemin minimal de plus de α %. On peut en fait établir une sorte plus générale de *théorème de sous-admissibilité* concernant une large famille d'heuristiques non minorantes (et même : non statiques) pour lesquelles on dispose d'une relation de la forme : $h \leq F$, où F satisfait certaines conditions [Farreny, 1995, 1997b,a, 1999] ; ce théorème énonce une formule de majoration de la longueur du chemin trouvé ; en l'appliquant aux algorithmes de Harris et Ghallab-Allard, on retrouve leurs propriétés de sous-admissibilité comme des cas particuliers. D'autres techniques de résolution, qui prennent davantage de libertés avec les soucis d'admissibilité et de sous-admissibilité, sont exposées au chapitre II.11.

1.11 Conclusion

Métaphoriquement, on pourrait dire que le paradigme de la *recherche heuristiquement ordonnée* consiste à atteler des connaissances empiriques au chariot de la recherche pour qu'elles l'orientent et le tirent efficacement. Chacun est familier d'une mise en œuvre particulière de ce paradigme : celle que l'on réalise pour se mouvoir à tout moment, sans employer une représentation précise de l'environnement ; notre comportement (plus ou moins instinctif ou réfléchi) semble résulter de la prise en compte de plusieurs segments d'informations : direction du but (voire : estimation de son éloignement), histoire de nos propres mouvements jusqu'ici, etc. Aller droit au but, voilà bien une heuristique d'usage courant, faillible certes, mais utile. Que peut-on en attendre ? Comment mieux s'en servir ?

En 1968, un premier cadre formel a été proposé. Le modèle A^* comprend un algorithme et une théorie qui justifie son *admissibilité*. Les notions de *minorance*, d'heuristique minorante *mieux informée* qu'une autre et de *monotonie*, expliquent pourquoi il convient de préférer P ou W à l'heuristique nulle. Effectivement, A^* muni de P parvient à résoudre des problèmes de taquin 3×3 que l'algorithme en largeur d'abord ne pouvait traiter faute de mémoire.

En 1985, IDA^* marque un progrès algorithmique, sustenté par le cadre formel précédent : parce qu'il est *admissible*, comme A^*, mais beaucoup plus économe en mémoire, IDA^* muni de P, parvient à résoudre *tous* les problèmes de taquin 3×3 (les plus difficiles exigeant 31 coups) ; il parvient aussi à résoudre un jeu d'essai de 100 taquins 4×4 tirés au sort (dont les solutions comptent en moyenne 53 coups environ). Malgré l'évolution du matériel, résoudre des problèmes de taquin 5×5 (tirés au sort) reste hors de portée.

En 1996, le remplacement de P par une nouvelle heuristique *minorante* ($\geq P$, mais pas *mieux informée*) permet qu'IDA^* résolve 9 taquins 5×5 sur un jeu de 10 tirés au sort (longueur moyenne des solutions : 101 environ).

En 1997, la combinaison d'arguments analytiques (employant une heuristique *majorante* pour écarter tous problèmes résolubles en moins de 80 coups) et le recours à un puissant réseau de calculateurs parallèles ont permis d'établir que le diamètre du graphe d'états des taquins 4×4 est soit 80 soit 81 [Brungger, 1997]. A notre connaissance, jusqu'à aujourd'hui, le diamètre du graphe d'états des taquins 5×5 demeure inconnu (mais ≥ 113).

En 1997, le recours à la technique des *Pattern databases* permet qu'IDA^* résolve 10 problèmes de Rubik's Cube tirés au sort (solutions : entre 16 et 18 mouvements de faces).

En 2010, grâce à la mobilisation de moyens informatiques colossaux il a été prouvé que le diamètre du graphe d'états du Rubik's Cube (classique : $3 \times 3 \times 3$; en comptant 1 coup pour tout mouvement de face) est de 20 seulement [Rokicki *et al.*, 2010] [Delayae, 2011].

Pour que la maîtrise de tels puzzles progresse, la puissance de calcul joue assurément un grand rôle ; il demeure bienvenu de la conjuguer avec de nouvelles idées concernant les fonctions d'évaluation, les algorithmes, les heuristiques. Noter : pour ces puzzles les coûts d'arcs valent uniformément 1. On peut espérer un certain nombre d'avancées concernant des domaines d'applications plus réalistes (dont : non uniformément arcs-valués), par la prise en compte des voies de formalisation évoquées précédemment ou d'autres à inventer.

Références

BAGCHI, A. et MAHANTI, A. (1983). Search algorithms under different kinds of heuristics–A comparative study. *JACM : Journal of the ACM*, 30(1) :1–21.

BARR, A., FEIGENBAUM, E. et COHEN, P. (1981). *The Handbook of Artificial Intelligence.* Morgan Kaufman, (Los Altos CA), 1982.

BERGE, C. (1970). *Graphes et Hypergraphes.* Dunod, Paris.

BRUNGGER, A. (1997). *Solving hard combinatorial optimization problems in parallel : two case studies.* Thèse de doctorat, ETH Zürich.

BUISSON, J.-C. (2008). Nutri-educ, a nutrition software application for balancing meals, using fuzzy arithmetic and heuristic search algorithms. *Artificial Intelligence in Medicine*, 42(3) :213–227.

CHAKRABARTI, P. P., GHOSE, S., ACHARYA, A. et de SARKAR, S. C. (1989). Heuristic search in restricted memory. *Artificial Intelligence*, 41(2) :197–222.

CHANG, C. L. et LEE, R. C. T. (1973). *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York.

CHARNIAK et MCDERMOTT (1985). *Introduction to Artificial Intelligence.* Addison-Wesley, Reading.

CHATILA, R. (1982). Path planning and environment learning in a mobile robot system. *In Proceedings ECAI-82*, pages 211–215. Orsay.

CULBERSON, J. C. et SCHAEFFER, J. (1998). Pattern databases. *Computational Intelligence*, 14(4) :318–334.

DE CHAMPEAUX, D. et SINT, L. (1977). An improved bidirectional heuristic search algorithm. *JACM : Journal of the ACM*, 24(2) :177–191.

DECHTER, R. et PEARL, J. (1985). Generalized best first search strategies and the optimality of A*. *JACM : Journal of the ACM*, 32(3) :505–536.

DECHTER, R. et PEARL, J. (1988). The optimality of A*. *In* KANAL, L. et KUMAR, V., éditeurs : *Search in Artificial Intelligence*, pages 166–199. Springer-Verlag.

DELAYAE, J. P. (2011). Le Rubik'Cube : pas plus de 20 mouvements ! *Pour la Science*, (400) :98–103.

DIJKSTRA, E. W. (1959). A note on two problems in connexion with graphs. *Numerical Mathematics*, 1 :269–271.

DUBOIS, D., LANG, J. et PRADE, H. (1987). Theorem proving under uncertainty. A possibility theory-based approach. *In Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 984–986, Milan, Italy.

ECKERLE, J. et SCHUIERER, S. (1995). Efficient memory-limited graph search. *Lecture Notes in Computer Science*, 981 :101–112.

FARRENY, H. (1987). *Exercices programmés d'intelligence artificielle.* Méthode + programmes. Masson.

FARRENY, H. (1995). *Recherche Heuristiquement Ordonnée dans les graphes d'états : algorithmes et propriétés.* Manuels informatiques Masson. Masson.

FARRENY, H. (1997a). Recherche Heuristiquement Ordonnée dans les graphes d'états : élargissement des cas d'admissibilité et sous-admissibilité. *Revue d'Intelligence Artificielle*, 11(4) :407–448.

FARRENY, H. (1997b). Recherche heuristiquement ordonnée : Généralisations compatibles avec la complétude et l'admissibilité. *Technique et Science Informatiques*, 16(7) :925–953.

FARRENY, H. (1999). Completeness and admissibility for general heuristic search algorithms-A theoretical study : Basic concepts and proofs. *Journal of Heuristics*, 5(3) :353–376.

FARRENY, H. (2002). Des heuristiques plus performantes si avec la fonction on garde la raison. *In 13ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et d'Intelligence Artificielle (RFIA 2002) , Angers-France, 08/01/02-10/01/02*, pages 607–613. Le Comité d'Organisation.

FARRENY, H. et GHALLAB, M. (1987). *Éléments d'intelligence artificielle.* Hermès.

FARRENY, H., PIQUET-GAUTHIER, S. et PRADE, H. (1984). Méthode de recherche ordonnée pour la désignation d'objets en génération de phrases. *In Proc. 6th Int. Cong. Cyb. and Systems*, pages 647–652.

FARRENY, H. et PRADE, H. (1982). Search methods with imprecise estimates. *In* TRONCALE, L., éditeur : *Proceedings 26th International Symposium on General Systems Methodology*, pages 444–446.

FELNER, A., KORF, R. E. et HANAN, S. (2004). Additive pattern database heuristics. *J. Artif. Intell. Res. (JAIR)*, 22 :279–318.

GARDNER, M. (1979). Jeux mathématiques du "scientific american". *CEDIC.*

GHALLAB, M. et ALLARD, D. G. (1983). A_ε – an efficient near admissible heuristic search algorithm. *In* BUNDY, A., éditeur : *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 789–791, Karlsruhe, FRG. William Kaufmann.

GONDRAN, M. et MINOUX, M. (1979). *Graphes et algorithmes.* Eyrolles, Paris.

GONELLA, R. (1989). *Diagnostic de pannes sur avions : mise en œuvre d'un raisonnement révisable.* Thèse de doctorat, ENSAE, Toulouse, France.

GOUZÈNES, L. (1984). Strategies for solving collision-free trajectories problems for mobile and manipulator robots. *International Journal of Robotics Research, Winter 1984*, 3(4) :51–65.

HANSSON, O., MAYER, A. et YUNG, M. (1992). Criticizing solutions to relaxed models yields powerful admissible heuristics. *Inf. Sci*, 63(3) :207–227.

HART, P. E., NILSSON, N. J. et RAPHAEL, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. and Cybernetics*, SSC-4 (2) :100–107.

IBARAKI, T. (1978). Branch-and-bound procedure and state-space representation of combinatorial optimization problems. *Information and Control*, 36(1) :1–27.

JOHNSON, W. W. et STOREY, W. E. (1879). Notes on the "15" puzzle. *American Journal of Mathematics*, 2 :397–404.

KAINDL, H. et KHORSAND, A. (1994). Memory-bounded bidirectional search. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1359–1364, Seattle, Washington. AAAI Press.

KANAL, L. N. et KUMAR, V., éditeurs (1988). *Search in Artificial Intelligence.* Springer-Verlag, Berlin.

KÖLL, A. L. et KAINDL, H. (1992). A new approach to dynamic weighting. *In* NEUMANN, B., éditeur : *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 16–17, Vienna, Austria. John Wiley and Sons.

KORF, R. E. (1985a). Depth-first iterative deepening : an optimal admissible tree search. *Artificial Intelligence*, 27(1) :97–109.

KORF, R. E. (1985b). Iterative-deepening A* : An optimal admissible tree search. *In Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 1034–1036, Los Angeles, California. Morgan Kaufmann.

KORF, R. E. (1993). Linear-space best-first search. *Artificial Intelligence*, 62(1) :41–78.

KORF, R. E. (1997). Finding optimal solutions to Rubik's Cube using pattern databases. *In Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-96)*, pages 700–705.

KORF, R. E. (2000). Recent progress in the design and analysis of admissible heuristic functions. *In Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000)*, pages 1165–1170, Austin, Texas, USA. AAAI / MIT Press.

KORF, R. E. et FELNER, A. (2002). Disjoint pattern database heuristics. *AIJ : Artificial Intelligence*, 134(1-2) :9–22.

KORF, R. E. et TAYLOR, L. A. (1996). Finding optimal solutions to the twenty-four puzzle. *In Proceedings of the 14th National Conference on Artificial Intelligence*

(AAAI-96), pages 1202–1207, Portland, Oregon, USA. AAAI Press / The MIT Press.

KOWALSKI, R. (1970). Search strategies for theorem proving. *In* MELTZER, B. et MICHIE, D., éditeurs : *Machine Intelligence 5*, pages 181–201. American Elsevier.

KUMAR, V. et KANAL, L. N. (1988). The CDP : A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. *In* KANAL, L. N. et KUMAR, V., éditeurs : *Search in Artificial Intelligence*, chapitre 1, pages 1–27. Springer-Verlag, Berlin.

LAURIÈRE, J.-L. (1986). *Intelligence Artificielle : résolution de problèmes par l'homme et la machine.* Editions Eyrolles, Paris.

LIROV, Y. (1987). A locally optimal fast obstacle-avoiding path-planning algorithm. *Mathematical Modelling*, 9(1) :63–68.

MANZINI, G. (1995). BIDA* : An improved perimeter search algorithm. *Artificial Intelligence*, 75(2) :347–360.

MARTELLI, A. (1976). Application of heuristic search methods to edge and contour detection. *Communications of the ACM*, 19(2) :73–83.

MARTELLI, A. (1977). On the complexity of admissible search algorithms. *Artificial Intelligence*, 8(1) :1–13.

MERO, L. (1984). A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23(1) :13–27.

MINKER, J., FISHMAN, D. H. et MCSKIMIN, J. R. (1973). The Q* algorithm—A search strategy for a deductive question-answering system. *Artificial Intelligence*, 4(3–4) :225–243.

MONTANARI, U. (1970). Heuristically guided search and chromosome matching. *Artificial Intelligence*, 1(3–4) :227–245.

NILSSON, N. J. (1971). *Problem Solving Methods in Artificial Intelligence.* New York : McGraw-Hill.

NILSSON, N. J. (1980). *Principles of Artificial Intelligence.* Tioga, Palo Alto. Édité en français : Principes d'Intelligence Artificielle, Cepadues 1988.

NILSSON, N. J. (1998). *Artificial Intelligence : A New Synthesis.* Morgan Kaufmann Publishers, San Francisco.

PEARL, J. (1984). *Heuristics - intelligent search strategies for computer problem solving.* Addison-Wesley series in artificial intelligence. Addison-Wesley. Édité en français : Heuristiques - Stratégies de recherche intelligentes pour la résolution de problèmes par ordinateur, Cepadues 1990.

PEARL, J. et KIM, J. H. (1982). Studies in semi-admissible heuristics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4(4) :392–399.

POHL, I. (1971). Bi-directional search. *In* MELTZER, B. et MICHIE, D., éditeurs : *Machine Intelligence 6*, pages 127–140. Edinburgh University Press, Edinburgh, Scotland.

PÉRENNOU, G., LAHENS, F., DAUBÈZE, P. et DE CALMES, M. (1986). Rôle et structure du lexique dans le correcteur vortex. *In Séminaire GRECO/GALF Lexiques et traitement automatique des langages*, Université Paul Sabatier, Toulouse.

RAO, N. V., KUMAR, V. et KORF, R. E. (1991). Depth-first versus best-first search. *In* DEAN, T. L. et MCKEOWN, K., éditeurs : *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 1*, pages 434–440. AAAI Press / The MIT Press.

RATNER, D. et WARMUTH, M. (1990). The $(n^2 - 1)$-puzzle and related relocation problems. *JSCOMP : Journal of Symbolic Computation*, 10 :111–137.

REINEFELD, A. (1993). Complete solution of the eight-puzzle and the benefit of node ordering in *IDA**. *In Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 248–253, Chambéry, France.

RICH, E. (1983). *Artificial intelligence.* McGraw-Hill. Édité en français : Intelligence Artificielle, Masson, 1987.

ROKICKI, T., KOCIEMBA, H., DAVIDSON, M. et DETHRIDGE, J. (2010). God's number is 20. http ://www.cube20.org/.

RUSSELL, S. (1992). Efficient memory-bounded search methods. *In* NEUMANN, B., éditeur : *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 1–5, Vienna, Austria. John Wiley and Sons.

RUSSELL, S. J. et NORVIG, P. (2003). *Artificial Intelligence : a modern approach.* Prentice Hall, Upper Saddle River, NJ, 2nd international édition.

SEN, A. K. et BAGCHI, A. (1989). Fast recursive formulations for best-first search that allow controlled use of memory. *In* SRIDHARAN, N. S., éditeur : *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 297–302, Detroit, MI, USA. Morgan Kaufmann.

SHAPIRO, S. (1987). *Encyclopedia of AI.* Wiley-Interscience.

SHIRAI, Y. et TSUJII, J. (1984). *Artificial intelligence : concepts, techniques, and applications.* Wiley series in computing. Wiley.

WINSTON, P. H. (1984). *Artificial Intelligence.* Addison-Wesley, Reading.

YAGER, R. R. (1986). Paths of least resistance in possibilistic production systems. *Fuzzy Sets and Systems*, 19(2) :121–132.

Chapitre 2

Jeux et recherche heuristique

Les algorithmes pour les jeux sont étudiés en intelligence artificielle depuis ses origines. Historiquement, le jeu d'Échecs et l'algorithme Alpha-Beta ont été les plus étudiés. Des algorithmes et des structures de données utilisés initialement pour les jeux comme l'approfondissement itératif et les tables de transpositions ont été ensuite réutilisés pour de nombreux autres problèmes.

2.1 Introduction

Nous présentons dans ce chapitre différents algorithmes utilisés pour les jeux. Nous commençons par deux sections sur les jeux à deux joueurs, la première traite de l'Alpha-Beta et de certaines de ses optimisations, la deuxième porte sur les algorithmes de Monte-Carlo qui ont donné récemment de très bons résultats sur certains jeux et qui ont une portée très générale. On peut noter au passage les contributions essentiellement françaises aux algorithmes de Monte-Carlo, on peut même parler d'une école française du Monte-Carlo dans les jeux. Nous abordons ensuite les puzzles et l'analyse rétrograde. Nous concluons avec une section sur les jeux vidéo.

2.2 Minimax, Alpha-Beta et améliorations

Cette section présente Minimax, Alpha-Beta et ses optimisations. Alpha-Beta est un algorithme de recherche arborescente utilisé dans les jeux à deux joueurs [Schaeffer et van den Herik, 2002], [Allis, 1994]. Il a été développé en même temps que la programmation des Échecs, domaine largement rempli et influencé par ce développement depuis 1950 [Shannon, 1950] jusqu'en 1997 lorsque Deep Blue a battu le champion du monde humain Gary Kasparov [Anantharaman *et al.*, 1989], [Campbell *et al.*, 2002]. L'origine de Alpha-Beta est difficile à déterminer exactement, car il n'existe pas un article fondateur. En revanche, il existe de nombreux articles le décrivant [Campbell et

Auteurs : Bruno Bouzy, Tristan Cazenave, Vincent Corruble et Olivier Teytaud.

Marsland, 1983], [Knuth et Moore, 1975], [Marsland, 1986]. Alpha-Beta est une amélioration de Minimax [von Neumann et Morgenstern, 1944] présenté dans la partie 2.2.1. La particularité de Alpha-Beta par rapport à Minimax d'élaguer certaines branches de l'arbre de recherche est expliquée dans la partie 2.2.2. En pratique, l'efficacité de Alpha-Beta dépend de nombreuses autres améliorations expliquées par les parties suivantes : les tables de transpositions (partie 2.2.3), Iterative Deepening (ID) (partie 2.2.4), la fenêtre de largeur minimale et MTD(f) (partie 2.2.5), ou bien d'autres encore (partie 2.2.6). Toutes les améliorations précédentes se placent dans le cadre d'une recherche à profondeur fixée à l'avance. D'autres algorithmes descendant de Minimax sont également très intéressants (partie 2.2.7). Enfin la détection de menaces permet de résoudre beaucoup plus rapidement certains jeux (partie 2.2.8).

2.2.1 Minimax

Minimax est un algorithme de recherche arborescente utilisé dans les jeux à deux joueurs à somme nulle avec alternance de coups amis et ennemis [von Neumann et Morgenstern, 1944]. Il suppose l'existence d'une fonction d'évaluation appelée à une profondeur donnée et commune aux deux joueurs. Le joueur ami cherche à maximiser l'évaluation et le joueur ennemi cherche à la minimiser. A un nœud ami (respectivement ennemi), la valeur minimax est le maximum (respectivement minimum) des valeurs minimax des nœuds fils. Pour connaître la valeur minimax de la racine d'un arbre de profondeur d et de facteur de branchement b, l'algorithme Minimax explore un nombre de nœuds approximativement égal à b^d.

2.2.2 Alpha-Beta

Alpha-Beta (AB) calcule la valeur minimax du nœud racine en parcourant moins de nœuds que ne le fait Minimax. Pour cela, AB utilise deux valeurs α et β, bornes d'un intervalle dans lequel est située la valeur AB du nœud considéré. Dans la suite, on appelle valeur AB, ou v, la valeur calculée par AB à un nœud de l'arbre. Le joueur ami (respectivement ennemi) cherche à obtenir une valeur v aussi grande (respectivement petite) que possible. Par définition, α est la valeur minimale v que le joueur ami est certain d'obtenir et que le joueur ennemi ne peut contester. β est la valeur maximale que le joueur ennemi est certain d'obtenir, et incontestable par le joueur ami. On a toujours $\alpha \leq v \leq \beta$ et $\alpha < \beta$. Au début de l'exécution de l'algorithme, le nœud courant est le nœud racine et on a $\alpha = -\infty$ et $\beta = +\infty$. La racine est supposée être un nœud ami. A un nœud ami, AB appelle AB récursivement en passant en paramètres α et β sur les nœuds fils, ceci dans un ordre donné généralement par les connaissances du domaine. Soit r la valeur retournée par AB sur un fils. Si $r \geq \beta$, AB s'arrête (par définition de β, il est impossible que r soit supérieur à β) et AB retourne β. Ce cas s'appelle une coupe β car les nœuds fils suivants ne sont pas explorés. Si $\beta > r > \alpha$, AB a amélioré ce qu'il pouvait atteindre donc AB met a jour α avec r. Si $r \leq \alpha$, AB continue en appelant AB sur les fils suivants. Quand tous les nœuds fils ont été explorés, AB retourne α. A un nœud ennemi, AB fait un traitement analogue : si $r \leq \alpha$, AB s'arrête et retourne α. Il s'agit d'une coupe α. Si $\alpha < r < \beta$, l'ennemi a amélioré ce

qu'il pouvait atteindre jusque-là, donc AB met à jour β avec r. Si $r \geq \beta$, AB continue. Quand tous les nœuds fils ont été explorés, AB retourne β.

Si Alpha-Beta est lancé avec des valeurs initiales α et β, et si AB retourne une valeur r, alors AB garantit les résultats suivants : si $\alpha < r < \beta$ alors r est égale à la valeur minimax, si $\alpha = r$ alors la valeur minimax est inférieure ou égale à α, si $r = \beta$ alors la valeur minimax est supérieure ou égale à β.

Alpha-Beta utilise une mémoire linéaire en fonction de la profondeur du nœud courant.

Alpha-Beta est très sensible à l'ordre dans lequel il explore les nœuds. Cet ordre est en général donné par des connaissances du domaine ou par des heuristiques de recherche. Le fait d'explorer le meilleur nœud en premier permet d'augmenter α, ce qui produit des coupes et diminue le nombre de nœuds explorés. [Knuth et Moore, 1975] montre que pour connaître la valeur minimax de la racine d'un arbre de profondeur d et de facteur de branchement b, AB explore un nombre de nœuds au moins égal à approximativement $2b^{d/2}$ et que cette borne inférieure peut être atteinte si l'ordre d'exploration des coups est bon. Cela signifie grossièrement que dans les bons cas AB explore approximativement $2\sqrt{T}$ nœuds où T est le nombre de nœuds explorés par Minimax.

En pratique, Alpha-Beta est utilisée dans sa version NegaMax. Contrairement à AB qui différencie le cas des nœuds amis et des nœuds ennemis, NegaMax gère un seul cas et ne voit qu'un seul type de nœud. Pour cela, pour chaque nœud fils, NegaMax appelle récursivement -NegaMax avec les paramètres $-\beta$ et $-\alpha$.

2.2.3 Tables de transposition (TT)

En pratique, on fait tourner AB avec une table de transpositions (TT). La TT est une table de toutes les positions déjà rencontréee jusque-là. Elle s'appelle table de transpositions car elle n'a d'intérêt que lorsque deux séquences de coups de la recherche aboutissent à la même position, ce qui s'appelle une transposition. Une transposition peut arriver dès qu'une séquence contient deux coups du même joueur. Par exemple, au Go, les séquences de coups A, B, C et C, B, A aboutissent à la même position si aucune capture n'arrive, ce qui est le cas général. En première approximation, quand AB est appelé sur un nœud, il regarde d'abord si la position courante est stockée dans la table. Si oui il utilise les informations disponibles et évite de refaire la recherche située sous cette position. Sinon, il fait la recherche. Quand AB a fini l'exploration d'un nœud, il écrit les résultats de la recherche issue de cette position dans la table. Cette amélioration est la première à apporter à AB.

Pour représenter une position d'un jeu de manière utilisable avec une table, une première idée est de l'associer à un nombre compris entre 0 et $|E|$, la taille de l'espace des états E du jeu. $|E|$ est potentiellement très grand, disons 2^G. (Au Go 9x9, on a approximativement $G = 133$, car $|E| \simeq 3^{81} \simeq 10^{40} \simeq 2^{133}$). Cette première idée n'est donc pas possible telle quelle sur les ordinateurs actuels et il faut représenter une position par un nombre significativement plus petit, au risque d'avoir des collisions (appelées collisions de type 1). Zobrist a inventé un codage permettant de représenter une position avec un nombre entier utilisable avec un ordinateur actuel (un nombre sur 32 ou 64 bits), avec une probabilité de collision arbitrairement faible. Pour cela,

à chaque valeur de propriété de la position est associée un nombre aléatoire, fixé hors ligne. (Par exemple, au Go, à la couleur blanche de l'intersection numéro 4 du damier correspond un nombre aléatoire fixé, à la couleur noire de l'intersection numéro 4 correspond un autre nombre aléatoire fixé, et à la couleur vide de cette intersection correspond le nombre 0, et ainsi de suite pour toutes les intersections du damier). Le nombre de Zobrist d'une position est alors le XOR des nombres aléatoires associés aux valeurs des propriétés de la position. (Par exemple, au Go 9x9, le nombre de zobrist d'une position est le XOR de tous les nombres de Zobrist correspondant aux 81 intersections du damier). Quand un coup est joué, le nombre de Zobrist de la nouvelle position est mis à jour incrémentalement en fonction des changements de valeurs des propriétés de la position (Au Go, si une pierre blanche est posée sur une intersection i, le nombre de Zobrist de la nouvelle position est le XOR de l'ancienne position avec le nombre de Zobrist correspondant à la couleur blanche de l'intersection i). Zobrist montre que la probabilité de collision de type 1 entre deux positions peut être rendue arbitrairement faible [Zobrist, 1990]. Ce mécanisme s'appelle le hâchage de Zobrist.

En pratique, la taille de la TT est limitée, disons égale à 2^L (avec $L = 20$ ou $L = 30$). On prend alors les L premiers bits du nombre de Zobrist pour obtenir l'index d'un enregistrement dans la table. Des collisions peuvent alors se produire si deux positions différentes ont des index identiques (collisions de type 2). Pour éviter une collision de type 2 et l'utilisation des informations sur un enregistrement ne correspondant pas à la position courante, la première chose faite par l'algorithme de recherche est de vérifier que le nombre de Zobrist de la position courante est égal à celui de l'enregistrement trouvé avec l'index. Si c'est le cas, l'algorithme peut utiliser l'information contenue dans l'enregistrement, sinon il fait comme si l'enregistrement n'existait pas et explore la position.

Les premiers programmes d'Échecs [Greenblatt *et al.*, 1967] utilisaient déjà les TT. Le hachage de Zobrist est un mécanisme très général, utilisé dans beaucoup de jeux utilisant une recherche arborescente.

2.2.4 Iterative deepening

AB est un algorithme en profondeur d'abord. Si la solution optimale est courte et située en dessous du second nœud de la racine, Alpha-Beta explore d'abord tous les nœuds situés sous le premier nœud. Il peut alors dépenser un temps inutile à explorer les profondeurs sous ce premier nœud.

Iterative Deepening (ID) est un algorithme itératif appelant AB à profondeur 1, puis 2, etc, tant que du temps de réflexion existe [Korf, 1985a] (cf. chapitre II.1). Le premier avantage de ID est d'être « anytime » (tant que du temps reste, ID explore la profondeur suivante, si le temps est écoulé, ID retourne le coup calculé à l'itération précédente). Un deuxième avantage est de trouver la solution optimale la plus courte. Un troisième avantage est son couplage avec les TT. A une itération donnée et sur une position donnée, ID stocke le meilleur coup trouvé. Aux itérations suivantes, ID explore le meilleur coup d'une position d'abord, ce qui produit des coupes et rend AB efficace. Ce mécanisme a été utilisé dans les premiers programmes d'Échecs [Slate et Atkin, 1977].

2.2.5 MTD(f)

Au lieu de lancer AB avec $\alpha = -\infty$ et $\beta = +\infty$, on peut lancer AB avec des valeurs quelconques pourvu que $\alpha < \beta$. Soit v la valeur AB théorique de la racine. Si α est tel que $v \leq \alpha$, alors AB retournera α et réciproquement. De même, si β est tel que $v \geq \beta$, alors AB retourne β et réciproquement. Enfin, si α et β sont tels que $\alpha < v < \beta$, alors AB retourne v et réciproquement.

L'idée des fenêtres à largeur minimale est de poser $\beta = \alpha + 1$. AB va produire beaucoup de coupes, et son exécution sera très peu coûteuse comparée à une exécution effectuée avec $\alpha = -\infty$ et $\beta = +\infty$. Le résultat d'une exécution de AB sera soit une borne inférieure sur v : $v \geq \alpha + 1$ si AB retourne $\alpha + 1$, soit une borne supérieure sur v : $v \leq \alpha$ si AB retourne α.

La classe d'algorithmes MTD [Plaat *et al.*, 1996] repose sur un mécanisme itéré des fenêtres à largeur minimale. (MTD signifie Memory Test Driver. « *Memory* » car pour être efficace cet algorithme doit utiliser les TT. « *Test* » car un appel de AB avec $\beta = \alpha + 1$ permet de tester si v est plus grand ou plus petit qu'une certaine valeur. « *Driver* » car il s'agit d'une classe d'algorithmes pilotant de manières différentes les appels itérés de AB).

MTD(f) est l'instance la plus simple et la plus utilisée.

MTD(f) appelle itérativement AB avec $\alpha = \gamma$ et $\beta = \gamma + 1$. γ est initialisé avec une valeur quelconque (ou bien donnée par exemple par la précédente exécution de MTD(f)). A chaque itération, si la valeur de retour de AB est égale à γ, alors $v \leq \gamma$ et γ est décrémenté, sinon $v \geq \gamma+1$ et γ est incrémenté. Après un nombre fini d'itérations, la borne inférieure et la borne supérieure de v sont égales, v est connue, MTD(f) s'arrête et le meilleur coup est lu dans la TT. A condition d'être couplée avec TT et ID, MTD(f) est une amélioration sensible de AB utilisée dans les programmes d'Échecs actuels.

2.2.6 Autres améliorations de Alpha-Beta

D'autres améliorations de AB existent. *Principal Variation Search* (PVS) est une recherche AB qui supppose que les nœuds sont déjà bien ordonnés par le générateur de coups et donc que la recherche est une vérification [Pearl, 1980b], [Pearl, 1980a]. Sur un nœud fils, PVS appelle PVS récursivement avec une fenêtre minimale pour vérifier que α n'est pas amélioré. Si c'est le cas, la recherche n'a pas coûté beaucoup, c'est l'intérêt de PVS. Sinon, on relance une recherche en mettant le β normal, ce qui coûte car c'est une seconde recherche. L'heuristique du coup nul [Donninger, 1993] consiste à déterminer une première valeur de α en jouant un coup ne faisant rien, donc donnant le trait à l'adversaire, et en lançant une recherche à profondeur réduite (donc ayant un coût négligeable comparé à celui d'une recherche normale). Cette première valeur de α est significative et permet de lancer une recherche normale avec une bonne valeur initiale. L'heuristique de l'histoire [Schaeffer, 1989] consiste à enregistrer les coups produisant des coupes AB dans une table, et à les essayer en premier dans d'autres positions dans lesquelles ils sont possibles. Cela suppose qu'un coup puisse s'identifier et s'appliquer à des positions différentes. C'est le cas au Go où le lieu du coup est un bon identifiant, et aux Échecs où la nature de la pièce, son origine et sa destination constituent aussi un identifiant utile.

Quiescence search [Beal, 1990] est une variante de AB consistant à n'engendrer que des coups urgents au sens du domaine considéré, et à effectuer la recherche jusqu'à une profondeur où il n'existe plus de coups urgents, et où la position est dite calme, donc évaluable de manière fiable. [Rivest, 1988] est une étude sur la formule de back-up à partir des valeurs des nœuds fils dans la valeur du nœud parent. Enfin, [Junghanns, 1998] est un état de l'art des travaux sur Alpha-Beta.

2.2.7 Meilleurs en premier

D'autres algorithmes améliorent Minimax en explorant les nœuds suivant une stratégie du meilleur en premier. *Proof Number Search* (PNS) [Allis *et al.*, 1994] compte le nombre de nœuds à explorer sous un nœud donné pour prouver sa valeur. PNS explore en premier les nœuds dont ce compteur est le plus faible. *Best-First Search* [Korf et Chickering, 1994] appelle la fonction d'évaluation à tous les nœuds et explore le meilleur nœud en premier. SSS* [Stockman, 1979] explore tous les nœuds en parallèle à la manière de A*. B* [Berliner, 1979] suppose l'existence de deux évaluations, une évaluation optimiste et une évaluation pessimiste, et explore de manière à prouver que la valeur pessimiste du meilleur nœud fils est supérieure à la valeur optimiste du second nœud fils. [McAllester, 1988], [Schaeffer, 1990] définissent les nœuds conspirants comme étant les nœuds feuilles de l'arbre exploré dont l'évaluation influe sur la valeur minimax de la racine, et explorent ces nœuds en premier.

2.2.8 Algorithmes exploitant les menaces

Une menace est un coup pour un joueur qui menace de gagner s'il est suivi par un autre coup du même joueur. Par exemple aux Echecs, un échec est une menace, au Go-Moku aligner quatre pierres est une menace. Lorsqu'on détecte une menace, en analysant les conditions pour lesquelles la menace est vérifiée on peut répertorier le sous-ensemble des coups possibles de l'adversaire qui permettent d'invalider la menace. Ils sont en général bien moins nombreux que tous les coups possibles et ce sont les seuls coups à envisager pour celui qui est menacé puisque tous les autres coups sont perdants. Le facteur de branchement est alors beaucoup plus petit ce qui permet de résoudre des problèmes beaucoup plus rapidement qu'en envisageant tous les coups possibles. L'utilisation de menaces pour sélectionner un petit nombre de coups à envisager a d'abord été faite sur des problèmes d'Echecs [Pitrat, 1976], puis des règles simples de menaces ont permis de résoudre le Go-Moku [Allis *et al.*, 1996], et dans cette lignée des règles ont été automatiquement engendrées pour accélérer la résolution de problèmes de Go [Cazenave, 1998]. L'évolution de ces algorithmes utilisant des connaissances sur les menaces a été de remplacer les connaissances par des recherches détectant les menaces, ce qui est plus simple à mettre en œuvre et plus général [Cazenave, 2001, 2002a, 2003a ; Thomsen, 2002 ; Cazenave, 2004]. Ces techniques ont par exemple permis de résoudre le Phutball 11x11 [Cazenave, 2002b] et l'AtariGo 6x6 [Boissac et Cazenave, 2006].

2.3 Recherche Monte-Carlo

Un changement majeur a eu lieu en matière de jeu de Go ces dernières années. En 1998, Martin Mueller (6ème Dan amateur au Canada) donnait le nombre astronomique de 29 pierres de handicap au programme « Many Faces of Go », au meilleur niveau de l'époque, et l'humain gagnait [Müller, 2002]. A l'inverse, en 2008, le programme MoGo, issu de l'école Française de Monte-Carlo Go, gagnait à 9 pierres de handicap contre Kim Myungwang, pourtant 8ème Dan pro. D'autres succès ont suivi, jusqu'à une victoire à handicap 4 contre un joueur professionnel.

Ces succès sont liés à des progrès algorithmiques majeurs. Les mêmes algorithmes ont eu de nombreuses applications à d'autres domaines : l'apprentissage actif [Rolet *et al.*, 2009], l'optimisation sur des grammaires [De Mesmay *et al.*, 2009], l'optimisation non linéaire [Rolet *et al.*, 2009]. Par ailleurs, dans le même temps, des algorithmes proches se développaient en planification.

2.3.1 Evaluation Monte-Carlo

La première idée était l'utilisation du recuit simulé pour ordonner des listes de coups [Bruegmann, 1993]. En effet, la technique la plus classique pour jouer à un jeu à information parfaite est la technique dite Alpha-Beta ; elle fonctionne par exemple très bien pour les dames ou les échecs, mais sa faiblesse est qu'elle nécessite une fonction d'évaluation, c'est-à-dire une fonction, relativement rapide, qui à une position associe une évaluation de sa valeur pour chacun des joueurs. Typiquement, aux échecs ou aux dames, un expert humain peut facilement écrire une fonction qui estime la probabilité pour noir de gagner la partie à partir d'une position donnée ; rien de tel n'existe au jeu de Go. [Bruegmann, 1993] proposa pour pallier ce problème une idée nouvelle : jouer un certain nombre de parties, aléatoirement, jusqu'à obtention d'une probabilité de gain approchée (cf Alg. 2.1) : le Monte-Carlo Go était né.

Algorithme 2.1 : Evaluation d'une position p par la technique de Monte-Carlo via n simulations.

Entrée : une position p, un nombre n de simulations.
Sortie : $\frac{1}{n}\sum_{i=1}^{n} r_i$ (probabilité estimée de gain pour noir).
for $i = \in \{1, \ldots, n\}$ **do**
  $p' = p$
  while p' n'est pas un état final **do**
    $c =$ coup aléatoire parmi les coups légaux en p'
    $p' = transition(p', c)$
  end while
  if p' est gagnant pour noir **then**
    $r_i = 1$
  else
    $r_i = 0$
  end if
end for

Quoique la technique de Monte-Carlo soit ancienne (on fait en général remonter son

utilisation intensive au projet Manhattan, *i.e.* au projet conduisant à la construction de la bombe atomique aux Etats-Unis pendant la seconde guerre mondiale, mais elle avait auparavant été signalée comme moyen curieux de faire des calculs approchés de π), son utilisation dans le contexte des jeux est alors neuve.

2.3.2 Fouille d'arbre Monte-Carlo

La technique se révèle crédible, elle est reprise [Bouzy et Cazenave, 2001 ; Bouzy et Helmstetter, 2003] et améliorée en la combinant avec des connaissances et des recherches [Bouzy, 2005 ; Cazenave et Helmstetter, 2005]. Néanmoins, le vrai « décollage » des performances aura lieu lorsque la technique sera combinée à une construction incrémentale d'arbre. L'algorithme ainsi obtenu, appelé « *Fouille d'Arbre Monte-Carlo (FAMC)* » (*Monte-Carlo Tree Search* [Coulom, 2006 ; Chaslot *et al.*, 2006]) est présenté en Alg. 2.2. La structure T est un ensemble de situations, augmenté peu à peu (on démarre avec une structure vide à laquelle on ajoute, à chaque simulation aléatoire, la première situation de cette simulation qui n'a pas encore été stockée) ; cette structure possède, pour chacun des nœuds qu'elle contient, un nombre de victoires pour noir et un nombre de victoires pour blanc.

Cette technique est actuellement à l'œuvre dans tous les programmes efficaces de jeu de Go, mais aussi en Havannah, Hex, Phantom Go, Amazons.

Il n'est pas précisé dans l'algorithme 2.2 quelle est la formule dite de « bandit ». Une variante classique, quoique pas la plus utilisée dans le cas du Go, est la formule dite UCT (*Upper Confidence Tree* [Kocsis et Szepesvari, 2006]) qui suit :

$$banditFormula(v, d, n) = v/(v + d) + \sqrt{K \log(n)/(v + d)} \qquad (2.1)$$

(où K est une constante choisie empiriquement, n est le nombre de simulations au nœud considéré, v le nombre de victoires pour le coup considéré et d le nombre de défaites). Le premier terme $v/(v + d)$, dit d'exploitation, favorise les coups qui ont un bon taux de succès ; le second terme favorise les coups encore peu explorés ($v + d$ est petit) et est ainsi appelé terme d'exploration. La formule 2.1 n'est pas proprement définie pour $v + d = 0$; il est fréquent de spécifier $banditFormula(v, d, n) = F$ lorsque $v + d = 0$, pour une certaine constante F [Gelly *et al.*, 2006].

Différentes modifications de cette formule ont été proposées. La formule dite RAVE (*Rapid Action Value Estimates*), basée sur les valeurs dites AMAF (*All Moves As First*), est comme suit :

$$banditFormula(v, d, v', d') = \alpha v/(v + d) + (1 - \alpha)v'/(v' + d').$$

On utilise les mêmes critères v et d, et on utilise en outre :

- v' le nombre de victoires où le coup considéré c est joué par le joueur au trait avant d'être joué par son adversaire [1] ;
- d' le nombre de défaites où le coup considéré c est joué par le joueur au trait avant d'être joué par son adversaire.

1. La victoire est comptée si le coup est joué par le joueur au trait pendant la simulation *avant* d'être éventuellement rejoué par son adversaire ; il n'est pas requis que le coup soit joué dès le premier coup à partir de la situation étudiée.

Algorithme 2.2 : Evaluation d'une position p par la technique de Fouille d'Arbre Monte-Carlo via n simulations. Notations : ¬blanc = noir = 1, ¬noir = blanc = 0 ; *transition*(p, c) est la situation où l'on se retrouve si le joueur au trait joue le coup c en position p.

Entrée : p une position, n un nombre de simulations
Sortie : $\frac{1}{n}\sum_{i=1}^{n} r_i$
T = structure vide
for $i \in \{1, \ldots, n\}$ **do**
  $p' = p$, $q = \varnothing$, *partie* $= \{p\}$
  while p' n'est pas un état final //Faire une partie complète **do**
    if p' est dans T // Si p' est en mémoire **then**
      j = joueur au trait en p' // Algorithme dit « bandit »
      for c =coup légal en p' **do**
        //Calculer le score pour tous les coups
        $p'' = transition(p', c)$
        $Score(c) = banditFormula(T(\neg j, p''), T(j, p''), T(j, p') + T(\neg j, p'))$
      end for
      c =coup parmi les coups légaux en p' maximisant $Score(c)$
    else
      if $q = \varnothing$ // On n'a pas encore trouvé l'état à rajouter
      then
        $q = p'$
      end if
      c =coup aléatoire parmi les coups légaux en p'
    end if
    $p' = transition(p', c)$
    $partie = partie + p'$
  end while
  Ajouter q dans T // si $q \neq \varnothing$
  if p' est gagnant pour noir **then**
    $r_i = 1$
  else
    $r_i = 0$
  end if
  for $p' \in partie \cap T$ **do**
    $T(r_i, p') = T(r_i, p') + 1$ // Incrémenter $T(r_i, p')$
  end for
end for

$\alpha(.)$ est une fonction tendant vers 1, par exemple $\alpha(n) = n/(n+50)$:

- v' et d' étant beaucoup plus grands que 1, on les utilise avec confiance et faute de mieux lorsque v et d sont trop petits pour que le ratio $v/(v+d)$ ait un sens statistique ;
- puis, le nombre de simulations augmentant, on migre vers $v/(v+d)$ qui est plus fiable asymptotiquement que $v'/(v'+d')$.

Une deuxième modification importante [Coulom, 2007 ; Lee *et al.*, 2009 ; Chaslot *et al.*, 2008] est l'utilisation d'heuristiques calées sur des bases de données ; une méthode simple pour utiliser une heuristique $h(p', c)$ (traduisant typiquement la fréquence avec laquelle le coup c est jouée en situation p', au vu de la configuration de la situation p' autour du coup c) est :

$$banditFormula(v, d, p', c) = v/(v+d) + Kh(p', c)/(v+d)$$

pour une certaine constante K. Les meilleurs résultats s'obtiennent en combinant ces différentes approches [Lee *et al.*, 2009].

Les avantages avancés de la méthode FAMC sont les suivants :

- passage à l'échelle : plus on augmente la puissance de calcul et plus le niveau monte ;
- très faible expertise humaine ; l'algorithme présenté en section 2.2 est indépendant du jeu (même l'heuristique $h(.,.)$ peut être calibrée automatiquement sur des bases de données, quoiqu'il soit bien établi que régler empiriquement les constantes correspondant à des motifs connus des experts améliore nettement les résultats).

En raison du passage à l'échelle de la méthode, des parallélisations ont été proposées [Cazenave et Jouandeau, 2007 ; Gelly *et al.*, 2008] ; toutefois, les résultats, s'ils sont numériquement bons au sens où l'algorithme parallèle gagne avec très grande probabilité contre l'algorithme séquentiel, n'en sont pas moins limités : il semble que les performances contre les humains ne montent pas aussi vite que les performances contre le code séquentiel. En effet, la méthode Monte-Carlo est parfois biaisée, car le Monte-Carlo est incapable d'évaluer certaines positions comme les courses de liberté (dites « *semeais* ») ; par exemple, tel *semeai* sera évalué comme gagnant à 50% pour noir, alors que pour un joueur humain il est clair qu'il est gagné à 100% pour noir. Augmenter la puissance de calcul n'empêchera pas l'algorithme de se tromper sur ce *semeai*, et la limite actuelle en matière de Go par Monte-Carlo semble liée à des limitations profondes de ce type.

2.3.3 Pour aller plus loin

Une première direction essentielle de recherche est la réduction des biais, *i.e.* la modification du Monte-Carlo, si possible de manière indépendante du problème (pour gagner en généricité), de façon à ce que le taux de victoire moyen dans les simulations aléatoires soit représentatif de la situation ; il est connu que les *semeais* ou certains problème compliqués de vie et mort, en jeu de Go, donnent de gros biais et il y a fort à parier qu'il en sera de même dans d'autres applications. Dans le cas du jeu de Go, [Gelly *et al.*, 2006] a exhibé un Monte-Carlo modifié beaucoup plus efficace que le

tirage aléatoire simple parmi les coups légaux. [Lee *et al.*, 2009] a montré qu'on pouvait beaucoup gagner en (i) traitant dans le Monte-Carlo des cas particuliers de *Nakade* (ii) dosant mieux la part aléatoire. L'extension de (i) aux *semeais* est une sorte de Graal jamais atteint. Des mélanges du Monte-Carlo avec des solveurs tactiques ont été tentés mais ne sont pas encore pleinement opérationnels [Cazenave et Helmstetter, 2005].

On peut noter que les meilleurs programmes généraux de jeu utilisent aussi la fouille d'arbre Monte-Carlo [Finnsson et Björnsson, 2008]. Le principe de la programmation générale de jeux est que le programme reçoit les règles du jeu juste avant de jouer. Une compétition de programmes a lieu tous les ans à AAAI ou à IJCAI et a été gagnée en 2009 et 2010 par Ary, un programme français. La recherche sur les programmes généraux de jeux remonte par ailleurs aux travaux de Jacques Pitrat [Pitrat, 1968].

Cette section a été entièrement dédiée aux jeux complètement observables ; l'extension au cas non observable a été faite de différentes manières [Cazenave, 2006b ; Rolet *et al.*, 2009] et reste un sujet de recherche.

Enfin, dans certains jeux (*e.g.* le poker), la modélisation de l'adversaire est centrale [Maîtrepierre *et al.*, 2008].

2.4 Puzzles

On appelle ici puzzles des problèmes à un joueur où l'on cherche une suite de coups permettant d'atteindre une solution du problème. Certains algorithmes peuvent optimiser le nombre de coups de la solution ou son score.

2.4.1 A*

L'algorithme A* (cf. chapitre II.1) [Hart *et al.*, 1968] permet de trouver des solutions ayant un minimum de coups à des puzzles variés. Les puzzles résolus par A* sont par exemple le Rubik's Cube [Korf, 1997], le Taquin ou Sokoban [Junghanns et Schaeffer, 2001]. A* est aussi utilisé pour les jeux vidéo [Cazenave, 2006a ; Bulitko *et al.*, 2008 ; Sturtevant *et al.*, 2009].

Pour chaque puzzle auquel on applique A*, il faut définir une heuristique admissible qui sera calculée pour chaque position rencontrée. Une heuristique est admissible si elle donne toujours une valeur qui est plus petite que le nombre réel de coups qu'il faudra jouer pour atteindre une solution à partir de la position.

Heuristique de Manhattan

L'heuristique admissible la plus communément utilisée est l'heuristique de Manhattan. Le principe de l'heuristique de Manhattan est de calculer très rapidement la solution d'un problème simplifié et d'utiliser le coût de cette solution comme minorant du coût de la solution réelle. Elle consiste à considérer pour chaque pièce prise séparément qu'on pourra l'amener vers son emplacement cible sans encombre et sans considérer les interactions avec les autres pièces. On fait ensuite la somme de ces valeurs pour toutes les pièces. Par exemple au Taquin on compte pour chaque pièce le nombre de coups pour la déplacer vers son emplacement cible s'il n'y avait pas d'autres pièces.

Pour le Rubik's cube on calcule la même chose pour chaque cube, toutefois comme chaque coup déplace huit cubes, il faut diviser la somme pour tous les cubes par huit. Concernant les déplacements sur une carte, on calcule l'heuristique de Manhattan en négligeant les obstacles.

Développement de l'arbre de recherche

Le principe de A* est de développer à chaque étape la position qui a le chemin estimé le plus petit. On estime le coût d'un chemin en ajoutant le coût du chemin pris pour arriver à cette position au coût minimum du chemin qu'il reste à parcourir, donné par l'heuristique admissible. Ceci assure que lorsqu'on trouve une solution elle est sur un chemin de coût minimal (toutes les autres positions qu'on pourrait encore développer sont sur un chemin de coût supérieur). Dans les jeux, le coût d'un chemin est souvent le nombre de coups joués sur le chemin.

Algorithme 2.3 : Recherche d'une solution de coût minimal avec A*

```
Entrée : une position p
Sortie : une solution de coût minimal ou échec
Ouverts = {p}
Fermés = {}
g[p] = 0
h[p] = chemin estimé à partir de p
f[p] = g[p] + h[p]
while Ouverts ≠ {} do
  pos = position dans Ouverts qui a le plus petit f
  if pos est la position cherchée then
    retourner le chemin jusqu'à pos
  end if
  retirer pos de Ouverts
  ajouter pos à Fermés
  for c =coup légal de pos do
    pos' = transition(pos, c)
    g' = g[pos]+coût de c
    if pos' n'est pas dans Fermés then
      if pos' n'est pas déjà dans Ouverts avec g[pos'] ≤ g' then
        g[pos'] = g'
        h[pos'] =chemin estimé à partir de pos'
        f[pos'] = g[pos'] + h[pos']
        ajouter pos' dans Ouverts
      end if
    end if
  end for
end while
retourner échec
```

2.4.2 Monte-Carlo

Le succès récent des méthodes de Monte-Carlo pour les jeux à deux joueurs a renouvelé leur intérêt pour les puzzles. L'utilisation d'une méthode de Monte-Carlo pour un puzzle est justifiée lorsqu'on ne dispose pas de bonnes heuristiques pour guider la recherche. C'est par exemple le cas pour des puzzles comme le Morpion Solitaire ou SameGame. Pour ces jeux, mais aussi pour le Sudoku ou le Kakuro, une méthode de Monte-Carlo imbriquée donne de très bons résultats [Cazenave, 2009]. Le principe de cette méthode est de jouer des parties aléatoires au plus bas niveau, et pour les niveaux supérieurs de choisir à chaque fois le coup qui a donné le meilleur score d'une partie du niveau juste inférieur (par exemple chaque coup d'une partie de niveau un est choisi d'après le score d'une partie aléatoire de niveau zéro qui commence par ce coup). Par ailleurs, les méthodes décrites dans la section précédente « Recherche Monte-Carlo » sont générales et applicables aux Puzzles.

2.4.3 Pour aller plus loin

Il est possible pour certains problèmes d'utiliser une version de A* en profondeur d'abord avec approfondissement itératif appelée IDA* [Korf, 1985b] ce qui permet essentiellement d'utiliser A* avec très peu de mémoire.

[Kendall *et al.*, 2008] est une bonne revue des puzzles NP-complets.

2.5 Analyse rétrograde

L'analyse rétrograde permet de calculer à l'avance une solution pour chaque élément d'un sous-ensemble des configurations d'un jeu. Elle a permis de résoudre optimalement plusieurs jeux. Nous présentons d'abord son application aux finales de jeux à deux joueurs, puis aux patterns.

2.5.1 Bases de données de finales

Le principe d'une base de données de finales est de calculer la valeur exacte de chaque position possible d'une finale. Par exemple aux Échecs on peut calculer pour chaque position qui contient cinq pièces ou moins sa valeur exacte, Ken Thompson a ainsi calculé les valeurs de positions contenant jusqu'à six pièces [Thompson, 1996].

L'algorithme utilisé pour calculer les valeurs des positions est un algorithme d'analyse rétrograde. Son principe est de commencer par répertorier toutes les positions de Mat. Pour chaque position gagnée, il déjoue un coup blanc et un coup noir et vérifie en rejouant tous les coups possibles à profondeur deux le statut de la position déjouée. Il trouve ainsi de nouvelles positions gagnées. Il continue ce processus de découverte de positions gagnées tant qu'il en trouve de nouvelles.

Les bases de données de finales sont très utilisées aux Échecs et permettent de jouer certaines finales instantanément et mieux que n'importe quel joueur humain. Elles ont même permis de découvrir des classes de positions gagnées que tous les joueurs croyaient nulles avant leur calcul par Ken Thompson (en particulier la finale Roi-Fou-Fou-Roi-Cavalier).

L'analyse rétrograde ne s'applique pas qu'aux Échecs. Chinook, le programme de dames anglaises (Checkers) qui a résolu ce jeu [Schaeffer *et al.*, 2007], fait aussi un usage intensif des bases de données de finales. Un autre jeu populaire qui a été complètement résolu à l'aide de l'analyse rétrograde est l'Awari [Romein et Bal, 2003]. Suite à l'analyse rétrograde de l'Awari, il existe maintenant un programme capable de jouer parfaitement toutes les positions d'Awari instantanément.

2.5.2 Bases de données de patterns

Lorsqu'on cherche à améliorer A* ou IDA* pour un problème, il est naturel de chercher à améliorer l'heuristique admissible. Améliorer l'heuristique admissible consiste à lui faire trouver des valeurs plus grandes pour un temps de calcul à peu près équivalent. En effet si l'heuristique donne des valeurs plus grandes tout en restant admissible, elle permettra à A* de couper les branches qui ne sont pas sur le plus court chemin plus tôt et donc de trouver une solution plus rapidement.

Une façon efficace d'améliorer l'heuristique admissible est d'effectuer des pré-calculs de nombreuses configurations de sous-problèmes plus simples que le problème original et d'utiliser les valeurs trouvées par ces pré-calculs comme heuristiques admissibles. Si l'on prend l'exemple du Taquin, l'heuristique de Manhattan consiste à considérer chaque pièce comme indépendante des autres. Toutefois si on prend en compte les interactions entre certaines pièces, l'heuristique sera améliorée. On calcule donc pour chaque configuration possible d'un sous-ensemble des pièces le nombre de coups qu'il faudra pour toutes les mettre à leur place en prenant en compte les interactions. Par exemple pour le Taquin 4x4, on peut calculer toutes les configurations possibles des huit premières pièces en ignorant les autres pièces et en mettant donc des vides dans les emplacements non occupés par une des huit premières pièces [Culberson et Schaeffer, 1998]. On calcule alors avec un algorithme d'analyse rétrograde du même type que ceux utilisés pour les Échecs, les Checkers ou l'Awari le nombre de coups minimal pour ranger chaque configuration possible. Pour utiliser cette base de données de *patterns* (i.e. de configurations) on repère pour chaque position la configuration des huit premières pièces et on renvoie la valeur précalculée stockée à l'indice correspondant à la configuration, ce qui est très rapide.

L'utilisation de bases de données de *patterns* n'est pas limitée au Taquin. On peut aussi par exemple calculer le nombre de coups nécessaires pour ranger toutes les configurations des huit cubes de coin au Rubik's cube et s'en servir comme heuristique admissible [Korf, 1997]. Que ce soit au Taquin, au Rubik's cube ou pour d'autre problèmes, les base de données de patterns permettent des gains de rapidité très importants par rapport à l'heuristique de Manhattan (mille fois plus vite pour le Taquin 4x4 et nécessaires pour résoudre optimalement le Rubik's cube).

Pour le Rubik's cube on peut aussi combiner deux bases de données de *patterns*, par exemple en calculant une base pour les huit cubes de coin et une base pour six des douze cubes de bord. On prend alors pour une position le maximum des deux valeurs trouvées sur la position à évaluer. En effet les coups bougent à la fois des cubes de coin et des cubes de bord, on ne peut donc pas faire la somme des deux heuristiques. Pour d'autres problèmes comme le Taquin ce n'est toutefois pas le cas : si on a deux bases pour deux ensembles disjoints de pièces, on peut additionner les deux valeurs

tout en restant admissible puisqu'aucun coup de la première base n'est compté dans la deuxième base (les pièces d'une base ne sont pas dans l'autre base) [Felner *et al.*, 2004].

Pour des problèmes comme les tours de Hanoi avec quatre tiges il peut aussi être intéressant de compresser les bases de données de pattern pour les faire tenir en mémoire. La compression consiste à ne stocker qu'une valeur pour un ensemble de configurations (on stocke le minimum des nombres de coups des éléments de l'ensemble) [Felner *et al.*, 2007].

L'utilisation de pré-calculs pour améliorer l'heuristique admissible et accélérer la recherche n'est pas limitée aux puzzles. Ainsi pour le calcul de plus court chemin sur une carte de jeu vidéo il est possible de pré-calculer la distance d'un seul point à tous les autres points de la carte puis d'utiliser l'inégalité triangulaire sur les distances pour calculer une heuristique admissible pour n'importe quel couple de points [Cazenave, 2006a].

On peut aussi calculer des bases de données de *patterns* pour les jeux à deux joueurs. Au Go par exemple on peut pré-calculer toutes les formes vivantes contenues dans un rectangle de taille donnée [Cazenave, 2003b] et accélérer ainsi la résolution de problèmes de vie et de mort.

2.6 Jeu vidéo

A côté des travaux importants sur les jeux classiques, de nouveaux types de jeux ont ces trente dernières années eux-aussi fait appel au domaine de l'intelligence artificielle. Ces jeux qu'on appelle « jeux vidéo » ont commencé à se distinguer de leurs prédécesseurs en donnant généralement un rôle important à la visualisation comme mode d'interaction, et en privilégiant dans un premier temps le jeu d'action où les réflexes priment sur la réflexion. Bien que visant généralement le grand public, ils nécessitent de l'intelligence artificielle aussi bien pour servir d'aversaire artificiel au joueur humain (comme le fait l'IA des jeux classiques) que pour peupler leurs mondes virtuels d'acteurs ou personnages qui les rendent vivants, crédibles et engageants. Il existe donc une communauté active d'individus en majorité à la frontière de la recherche et de l'industrie s'intéressant à ce domaine, comme en témoigne une série d'ouvrages spécialisés régulièrement mise à jour [Rabin, 2002, 2003, 2006]. Beaucoup de chercheurs en IA se tournent vers le jeu vidéo comme un domaine d'étude très riche [Laird, 2002]. Les jeux vidéo sont des plates-formes d'expérimentation pour plusieurs approches de l'intelligence artificielle : la simulation y est facile, soit en mode IA contre IA soit IA contre humain, les utilisateurs étant souvent plus faciles à trouver que pour d'autres types d'expérimentations. Citons par exemple des travaux sur le jeu Tetris [Thiery et Scherrer, 2009] et sur le problème de la *patrouille multiagent* [Almeida *et al.*, 2004]. Ces dernières années, certaines plates-formes ouvertes, qui facilitent grandement l'entrée dans ce domaine, ont même été proposées aussi bien dans le domaine des jeux d'action que des jeux de stratégies. Enfin nous verrons que des chercheurs venus vers ce domaine pour y tester ou comparer leurs techniques de prédilection y trouvent de nouvelles problématiques de recherche issues de l'industrie ; ces nouvelles problématiques sont susceptibles de renouveler le champ de l'intelligence artificielle.

2.6.1 Introduction

Nous allons voir que, alors que certaines catégories de jeux vidéo peuvent être vues, du point de vue de l'IA, comme des extensions des jeux classiques, d'autres renouvellent complètement la problématique. Dans une première catégorie de jeux, on placera tout d'abord les jeux de stratégie modernes qui, comme leurs aînés, mettent en scène un conflit ou une compétition entre deux ou plusieurs camps, représentant chacun une armée, civilisation, ou autre faction, dans un contexte historique, imaginaire voire fantastique. Des exemples bien connus sont *Age of Empires* (Microsoft), un classique grand public, la série *Total War* (Creative Assembly) qui combine niveaux stratégiques et combats tactiques 3D temps réel à des époques antiques et moyen-âgeuses, la série *Civilisation* de Sid Meier, ou *Hearts of Iron* (Paradox Interactive). Les innovations sont sensibles et vont au-delà de l'immersion visuelle et sonore. Outre le déplacement de pièces (unités combattantes, etc.) dans un environnement 2D ou 3D qui rappelle les jeux classiques et beaucoup de *wargames*, on peut être amené à gérer aussi une économie (collecte de ressources, production, budget,...), une diplomatie (négociations d'alliances,...) voire les priorités de la recherche et de l'innovation (technologies militaires, idées de civilisation, innovations politiques). Ces niveaux multiples de simulation, du plus tactique (déplacement d'unités sur une carte), au plus stratégique (priorités à long terme,...) et leurs interactions complexes, ajoutent de nouveaux niveaux de complexité où l'impact des choix à moyen ou long terme est difficilement prédictible. Le jeu se déroule de manière classique au tour par tour, ou plus souvent en « temps réel » ou à une vitesse choisie par le joueur en fonction de l'intensité du jeu. Du point de vue de la complexité, un élément essentiel qui distingue ces jeux des jeux plus classiques est le parallélisme : que ce soit pour un *wargame* ou pour un jeu de « grande stratégie », toutes les unités peuvent recevoir du joueur des instructions indépendantes à chaque instant ou tour de jeu. L'ensemble des décisions et actions envisageables devient donc problématique à énumérer, et encore plus à évaluer, car sa taille croît exponentiellement avec le nombre d'unités. Les approches présentées plus haut issues de l'intelligence artificielle des jeux classiques, à base d'exploration d'un arbre de jeu, deviennent inopérantes.

Le premier point, lié à la complexité des jeux modernes, explique en partie un phénomène qui pourrait paraître a priori surprenant : l'IA des jeux vidéo utilise peu les techniques issues des recherches en IA sur les jeux classiques. Pourtant, le jeu vidéo est l'un des rares domaines où l'industrie s'est en grande partie appropriée la notion et surtout le vocable d'intelligence artificielle pour en faire un argument commercial. On parle dans l'industrie de l'*IA d'un jeu* comme la partie du logiciel qui gère et produit les comportements automatisés de l'adversaire ou des personnages non joueurs qui peuplent l'environnement. La question de savoir si cette *IA* utilise des techniques issues du domaine de recherche de l'IA proprement dit y est vu, peut-être à juste titre, comme secondaire. Comme nous allons le voir dans la section 2, une majeure partie des jeux vidéos actuels font en effet appel à des techniques qui peuvent sembler assez basiques du point de vue de la recherche en IA, mais qui ont de nombreux avantages du point de vue du concepteur de jeux tout en permettant une certaine complexité. Nous allons aussi voir comment des travaux récents, issus du monde de la recherche autant que de l'industrie, font évoluer ce domaine en partant d'une IA scriptée et figée

vers une IA adaptative.

Cependant, un rapide panorama des problématiques de l'IA se doit d'en brosser les autres aspects. Une dimension importante est la notion de personnage non joueur, ces créatures qui peuplent le monde d'un jeu, en particulier les jeux d'aventure et jeux de rôle, mais on peut généraliser cette notion aux jeux de sports, de simulation, et aux jeux de tir à la première personne (ou FPS pour *First-Person Shooters*), catégorie très populaire. On dépasse ici la notion d'adversaire, mais on s'approche de la notion d'acteur (qui doit suivre les instructions du metteur en scène) voire d'un agent autonome qui doit agir, réagir, interagir de manière crédible et intéressante avec l'histoire et le joueur humain. Tous les travaux récents sur les agents autonomes intelligents trouvent un débouché, et un champ d'expérimentation, particulièrement séduisant dans cette problématique [Corruble et Ramalho, 2009], mais on va voir plus bas qu'on est conduit dans ce cadre à approfondir, redéfinir ou même dépasser certains objectifs des travaux classiques sur les agents rationnels.

2.6.2 IA dans l'industrie du jeu vidéo - scripts et évolutions

Les jeux vidéo modernes doivent être vus comme des œuvres interactives qui s'inscrivent en partie dans une culture cinématographique où un auteur imagine une histoire et met en scène des personnages. Ils y ajoutent une dimension clé qui est l'interactivité : l'évolution de l'histoire est fortement impactée par les actions du joueur, qui vont l'orienter dans une direction ou une autre ; on parle alors de narration interactive [Perlin, 2005 ; Natkin, 2004]. Cet héritage explique en partie la réticence des concepteurs des jeux les plus scénarisés vis-à-vis de l'idée même d'une intelligence artificielle conduisant à des agents autonomes : les PNJ y sont alors vus comme des acteurs avant tout, qui doivent suivre les indications du concepteur et orienter l'évolution de l'histoire dans une des directions voulues. Les jeux les plus scénarisés font parties de la grande catégorie des *roller-coasters* (montagnes russes), ces jeux qui sont conçus précisément pour faire vivre au joueur une suite d'émotions planifiées où s'enchaîne des pics d'intensité, des moments de détente puis de suspense, etc. Ces jeux *roller-coasters* s'opposent aux jeux *sandbox* (bacs à sable), où le joueur construit lui-même l'histoire à partir d'un monde/environnement au contenu et à l'interactivité riche.

Dans la première catégorie, les jeux scénarisés ont le plus souvent une approche scriptée de l'intelligence artificielle. Des comportements, ou séquences d'actions sont déclenchées quand certaines conditions sur l'état du jeu ou de l'avatar du joueur sont réunies. Les PNJ ainsi programmés produisent les comportements atttendus par le concepteur au moment où il les attend et répond donc en grande partie aux besoins du jeu vidéo. Néanmoins, cette approche a ses propres limitations. Le coût de développement est élevé car toutes les situations intéressantes doivent être prévues au moment de la conception ce qui suppose que les comportements des joueurs soient suffisament contraints ... ce qui va à l'encontre du sentiment d'immersion recherché pour la plupart des jeux vidéo. D'autre part, cela tend à conduire vers un jeu figé. La même situation généralement reproduit les mêmes comportements. Ceci peut entraîner une forme de lassitude pour le joueur, mais une autre conséquence est que le joueur peut exploiter ces répétitions pour piéger le jeu : en prévoyant trop aisément la réaction de l'IA à ses propres actions, il obtient un avantage qui nuit rapidement à l'intérêt du jeu.

Du point de vue technique, différentes approches ont été utilisées pour modéliser les comportements des PNJ. On a longtemps utilisé de simples machines à états finis. Les limitations des machines à états finis en terme de richesse de représentation ont conduit à l'utilisation plus récemment de machines à états finis hiérarchiques [Harel, 1987] qui permettent de définir des états et des transitions généralisés et donc de mettre en commun certains aspects d'états similaires. De même, les arbres de comportements (*behaviour trees* [Flórez-Puga *et al.*, 2008]) visent à factoriser un maximum d'informations communes à plusieurs états ; le « comportement » y devient l'entité clé au détriment de l'état. Ces dernières années, pour la mise en œuvre de ces modèles, de véritables langages de scripts puissants (tel LUA[Ierusalimschy *et al.*, 2007], utilisés entre autres dans le célèbre jeu de rôle multijoueur en ligne *World of Warcraft*) ont été proposés en partie pour répondre aux besoins du jeu vidéo. Ils ont l'avantage d'offrir un bon compromis entre vitesse d'exécution et une programmation externe au moteur du jeu, qui permet de gérer l'IA dans les dernières étapes du cycle de développement, et de la raffiner après commercialisation voire même de proposer aux utilisateurs (les joueurs) d'y contribuer par eux-mêmes.

2.6.3 Voies de recherches : IA adaptative et planification

Pour pallier les limitations des approches scriptées citées plus haut, des tentatives ont été faites ces dernières années pour combiner les avantages des techniques de *scripting* appréciées de l'industrie du jeu vidéo et des capacités d'adaptation telles que celles étudiées par des chercheurs en intelligence artificielle, par exemple dans la communauté de l'apprentissage automatique. Un exemple important est le *Dynamic Scripting* [Spronck *et al.*, 2006]. Dans sa version initiale, cette approche vise à apprendre, par l'expérience, à associer un score à chaque script prédéfini par le concepteur du jeu. Cet apprentissage permet ensuite de sélectionner les scripts les plus adaptés à une situation donnée. Les versions ultérieures du *Dynamic Scripting* ont ajouté un niveau d'apprentissage par renforcement qui permet de modifier les scripts eux-mêmes ce qui en fait une véritable méthode d'apprentissage qui garde cependant la possibilité d'utilisation en entrée de scripts pré-existants, proposés par le concepteur.

Dans un contexte majoritairement académique, des recherches proposent depuis quelques années de concevoir l'IA des jeux de manière assez radicalement différente en plaçant à son cœur des techniques d'apprentissage qui ont un rôle triple : éviter les comportements figés des approches très scriptées classiques, offrir une possibilité d'adaptation du jeu (de son IA) au joueur (puisque c'est en jouant que l'IA adaptative va développer sa stratégie), et d'un point de vue de développement, offrir une alternative à l'approche classique qui implique la programmation puis le débogage de nombreux scripts par plusieurs programmeurs (certains studios de jeux vidéo emploient plusieurs dizaines de programmeurs de scripts d'IA en phase de production). La résistance à l'utilisation de ce type d'approche par l'industrie a plusieurs raisons : les techniques d'apprentissage actuelles convergent lentement sur des problèmes complexes, la paramétrisation est ardue, les comportements induits sont difficilement prévisibles et il est difficle de garantir a priori que le résultat obtenu soit acceptable par le concepteur et finalement par le joueur. D'un autre côté, l'industrie reconnaît que les méthodes adaptatives utilisant l'apprentissage sont certainement une voie prometteuse, comme

en témoigne la préface enthousiaste de [Rabin, 2002] qui qualifie l'apprentissage de *Next Big Thing*, pour l'IA des jeux vidéo, avant d'y consacrer dix chapitres de son ouvrage.

De par leur nature, la plupart des jeux vidéo se prêtent bien à une modélisation dans laquelle un ou plusieurs agents (PNJ) interagissent avec leur environnement en jouant. Ils reçoivent le plus souvent une évaluation de leurs actions, par exemple grâce à l'évolution d'un score de jeu. Ils semblent donc adaptés à des approches à base d'apprentissage par renforcement. Ce sont cependant des domaines où les espaces d'états et d'actions se révèlent plus grands que pour la plupart des applications connues de ces techniques, et incitent donc à améliorer ces méthodes, par exemple par des approches de factorisation des Processus Décisionnels Markoviens sous-jacents [Degris *et al.*, 2009] ou par une décomposition hiérarchique de l'apprentissage et la construction de représentations adaptées [Madeira et Corruble, 2009], ce qui a conduit à des solutions viables pour des jeux aussi variés que les FPS ou des jeux de stratégie de type *wargame* historique respectivement. Dans ce dernier cas en particulier, la décomposition hiérarchique de la prise de décision et de l'apprentissage, ainsi que l'adaptation automatique par abstraction des représentations est rendue nécessaire par le parallélisme qui fait d'un *wargame* une véritable simulation multiagent [Corruble et Ramalho, 2009].

A côté de ces approches utilisant l'apprentissage intensément, il ne faut pas négliger des approches qui utilisent la planification. Les travaux présentés dans [Degris *et al.*, 2009] sont d'ailleurs un exemple intéressant combinant l'apprentissage par renforcement et l'utilisation d'un modèle lui-même appris par expérience. Des approches utilisant des techniques sophistiquées de planification, en particulier à base de réseaux de tâches hiérarchiques (HTN), deviennent une direction de recherche importante [Hoang *et al.*, 2005] dont certains jeux commerciaux, en particulier des jeux de stratégie et *wargames* complexes tels que *Total War* (Creative Assembly), *Airborne Assault* (Panther Games), ou *Armed Assault* (Bohemia Interactive), s'inspirent grandement.

2.6.4 Nouvelles problématiques de recherche pour l'IA des jeux vidéo

Nous avons présenté dans les pages précédentes des problématiques d'intelligence artificielle pour le jeu vidéo comme un prolongement des recherches sur les jeux classiques. L'hypothèse sous-jacente y était que l'objectif est de créer une IA qui joue mieux, c'est-à-dire dont le niveau de performances approche, atteint, voire dépasse, celui des joueurs humains. Cet objectif, tellement évident qu'il est souvent tu par les chercheurs en intelligence artificielle, est sérieusement remis en question par le domaine du jeu vidéo. En effet, si pour certains types de jeux, qu'ils soient classiques (échecs ou plus récemment Go) ou « modernes » (jeux de stratégie), la difficulté première pour un concepteur d'IA est de proposer un défi de bon niveau au joueur humain, ce n'est pas le cas pour de nombreux jeux pour lesquels les capacités des machines les rendent d'ores et déjà aptes à dépasser le niveau de la plupart des joueurs. La question pour la recherche en IA se pose alors différemment : le but n'est plus d'atteindre le niveau humain, il est de proposer un adversaire, ou un compagnon de jeu, avec lequel le joueur humain appréciera la confrontation. On touche là à des notions complexes liées à l'idée

de plaisir de jouer et d'amusement qui sont au cœur du jeu mais que les sciences dures ont peu abordé jusqu'à maintenant.

Beaucoup de travaux, à l'intersection de l'intelligence artificielle, de la psychologie et des sciences sociales s'attèlent à la définition et la mesure du plaisir et de l'amusement du joueur. Des théories issues de l'esthétique et du cinéma sont invoquées d'un côté, des approches expérimentales qui observent l'activité du joueur et ses paramètres physiologiques (rythme cardiaque, etc.) pour évaluer son intêret sont appliquées de l'autre. Ceci peut guider la conception de l'IA pour que les comportements qu'elle produit induisent la satisfaction ou l'amusement du joueur. Un cas particulier de cette problématique qui fait l'objet de nombreux travaux ces dernières années est le problème de l'équilibrage du niveau de jeu. Comment faire pour que l'IA joue au bon niveau quel que soit l'adversaire et ses évolutions dans le temps ? C'est souvent en s'inspirant de la théorie psychologique du *flow* [Csikszentmihalyi et Csikszentmihalyi, 1975 ; Csikszentmihalyi et Csikzentmihaly, 1990] qui associe le sentiment de bien-être à un bon équilibre entre niveau de compétence et difficulté de la tâche à accomplir que le jeu vidéo traite de cette question [Chen, 2007]. En particulier, [Andrade *et al.*, 2006] s'intéresse à l'équilibrage dynamique du niveau de jeu, en proposant une méthode détournant l'usage classique de l'apprentissage par renforcement pour que l'action sélectionnée ne soit plus nécessairement celle qui a la meilleure valeur. Tout en apprenant à mieux jouer (en améliorant ainsi leur *compétence*), les agents apprennent aussi à adapter leur niveau de jeu, leur *performance*, en sélectionnant quand nécessaire des actions qu'ils jugent sub-optimales de leur point de vue, parce qu'ils estiment qu'elles seront plus adaptées au niveau de leur adversaire, le joueur humain.

Les travaux sur le niveau de jeu sont facilités par l'existence de mesures objectives, comme le score de jeu. D'autres aspects des jeux quelquefois plus difficilement mesurables ont un impact très fort sur la perception du joueur et à son immersion dans l'histoire. La crédibilité des personnages non joueurs en est un bon exemple, particulièrement important dans les jeux d'aventure et les jeux de rôles qui supposent en général des interactions complexes (dialogues, négociations, ...) entre avatar du joueur et PNJ. Tout un champ de recherche se développe actuellement autour de cette notion assez subtile. En visant la crédibilité plus que le réalisme, on se place dans un cadre ludique plus que de simulation ; on s'autorise à travailler dans des mondes imaginaires où les règles du monde réel ne s'appliquent pas toutes mais où on recherche toujours une forme de cohérence interne qui contribue à ce que le joueur reste « pris » dans l'histoire. Dans ce cadre, il faut parfois dépasser le but d'avoir des PNJ aux comportements performants et rationnels. Pour qu'ils soient crédibles, il faut plutôt qu'ils aient une personnalité reconnaissable qui influence leurs actions sur le long terme, et qu'ils réagissent émotionnellement de manière crédible aux évènements du monde et aux interactions avec les autres personnages. Les PNJ des jeux deviennent donc un champ important pour les travaux sur l'*affective computing* [Rickel *et al.*, 2002]. [Ochs *et al.*, 2009] par exemple propose un modèle computationnel permettant de simuler la dynamique émotionnelle et sociale des PNJ en tenant compte de leur personnalité, dans un contexte donné.

Les quelques problématiques introduites ci-dessus donnent une idée de la richesse des recherches sur l'IA des jeux vidéo. Cette liste est cependant loin d'être exhaustive.

En se déplaçant du simple rôle d'adversaire à celui de PNJ, l'IA a étendu son champ d'action, mais elle se voit déjà sollicitée pour d'autres aspects du jeu vidéo. Peut-elle contribuer à la conception des jeux ? A la mise en scène, par exemple pour le choix du positionnement automatique de la caméra de façon à donner la vue la plus intéressante de l'action au joueur, ou pour adapter ou composer la musique en fonction de l'état du jeu et du joueur ? Tout ces défis constituent des nouvelles frontières aussi bien pour la recherche en IA que pour le jeu vidéo.

2.7 Conclusion

Nous avons présenté dans ce chapitre les algorithmes classiques en programmation des jeux : l'Alpha-Beta et ses améliorations pour les jeux à deux joueurs et à somme nulle, A* pour les puzzles. Nous avons aussi présenté des approches plus récentes et en plein développement comme les algorithmes Monte-Carlo et les applications de l'IA aux jeux vidéo.

Comme nous l'avons vu, l'intelligence artificielle dans les jeux recouvre de nombreux algorithmes et soulève des problèmes variés qui ne sont pas nécéssairement spécifiques aux jeux. Les thèmes de recherche les plus actifs ces dernières années sont liés aux méthodes de Monte-Carlo et aux jeux vidéo.

Références

Allis, L. (1994). *Searching for solutions in games and Artificial Intelligence.* Thèse de doctorat, Vrije Universitat Amsterdam.

Allis, L., van der Meulen, M. et van den Herik, H. (1994). Proof-number search. *Artificial Intelligence*, 66 :91–124.

Allis, L. V., van den Herik, H. J. et Huntjens, M. P. H. (1996). Go-moku solved by new search techniques. *Computational Intelligence*, 12 :7–23.

Almeida, A., Ramalho, G., Santana, H., Tedesco, P. A., Menezes, T., Corruble, V. et Chevaleyre, Y. (2004). Recent advances on multi-agent patrolling. *In SBIA*, volume 3171 de *Lecture Notes in Computer Science*, pages 474–483. Springer.

Anantharaman, T., Campbell, M. et Hsu, F. (1989). Singular extensions : adding selectivity to brute force searching. *Artificial Intelligence*, 43(1) :99–109.

Andrade, G., Ramalho, G., Gomes, A. S. et Corruble, V. (2006). Dynamic game balancing : An evaluation of user satisfaction. *In AAAI conference on Artificial Intelligence and Interactive Digital Entertainement*, pages 3–8.

Beal, D. (1990). A generalised quiescence search algorithm. *Artificial Intelligence*, 43 :85–98.

Berliner, H. (1979). The B* Tree Search Algorithm : a best-first proof procedure. *Artificial Intelligence*, 12 :23–40.

Boissac, F. et Cazenave, T. (2006). De nouvelles heuristiques de recherche appliquées à la résolution d'atarigo. *In Intelligence artificielle et jeux*, pages 127–141. Hermes.

BOUZY, B. (2005). Associating domain-dependent knowledge and monte carlo approaches within a go program. *Inf. Sci.*, 175(4) :247–257.

BOUZY, B. et CAZENAVE, T. (2001). Computer Go : An AI oriented survey. *Artificial Intelligence*, 132(1) :39–103.

BOUZY, B. et HELMSTETTER, B. (2003). Monte-carlo go developments. *In ACG*, volume 263 de *IFIP*, pages 159–174. Kluwer.

BRUEGMANN, B. (1993). Monte carlo go. *Unpublished.*

BULITKO, V., LUSTREK, M., SCHAEFFER, J., BJORNSSON, Y. et SIGMUNDARSON, S. (2008). Dynamic control in real-time heuristic search. *Journal of Artificial Intelligence Research*, 32(1) :419–452.

CAMPBELL, M., HOANE, A. et HSU, F.-H. (2002). Deep Blue. *Artificial Intelligence*, 134 :57–83.

CAMPBELL, M. et MARSLAND, T. (1983). A comparison of minimax tree search algorithms. *Artificial Intelligence*, 20 :347–367.

CAZENAVE, T. (1998). Metaprogramming Forced Moves. *In ECAI 1998*, pages 645–649, Brighton, UK.

CAZENAVE, T. (2001). Iterative Widening. *In Proceedings of IJCAI-01, Vol. 1*, pages 523–528, Seattle.

CAZENAVE, T. (2002a). Abstract Proof Search. *In* MARSLAND, T. A. et FRANK, I., éditeurs : *Computers and Games 2000*, volume 2063 de *Lecture Notes in Computer Science*, pages 39–54. Springer.

CAZENAVE, T. (2002b). Gradual abstract proof search. *ICGA Journal*, 25(1) :3–15.

CAZENAVE, T. (2003a). A Generalized Threats Search Algorithm. *In Computers and Games 2002*, volume 2883 de *Lecture Notes in Computer Science*, pages 75–87, Edmonton, Canada. Springer.

CAZENAVE, T. (2003b). Metarules to improve tactical go knowledge. *Inf. Sci.*, 154(3-4) :173–188.

CAZENAVE, T. (2004). Generalized widening. *In ECAI 2004*, pages 156–160, Valencia, Spain. IOS Press.

CAZENAVE, T. (2006a). Optimizations of data structures, heuristics and algorithms for path-finding on maps. *In CIG*, pages 27–33.

CAZENAVE, T. (2006b). A Phantom-Go program. *In Advances in Computer Games 2005*, volume 4250 de *Lecture Notes in Computer Science*, pages 120–125. Springer.

CAZENAVE, T. (2009). Nested Monte-Carlo search. *In IJCAI 2009*, pages 456–461, Pasadena, USA.

CAZENAVE, T. et HELMSTETTER, B. (2005). Combining tactical search and monte-carlo in the game of go. *IEEE CIG 2005*, pages 171–175.

CAZENAVE, T. et JOUANDEAU, N. (2007). On the parallelization of UCT. *In Proceedings of CGW07*, pages 93–101.

CHASLOT, G., SAITO, J., BOUZY, B., UITERWIJK, J. W. H. M. et van den HERIK, H. J. (2006). Monte-Carlo Strategies for Computer Go. *In* SCHOBBENS, P.-Y., VANHOOF, W. et SCHWANEN, G., éditeurs : *Proceedings of the 18th BeNeLux Conference on*

Artificial Intelligence, Namur, Belgium, pages 83–91.

CHASLOT, G., WINANDS, M., UITERWIJK, J., van den HERIK, H. et BOUZY, B. (2008). Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(3) :343–357.

CHEN, J. (2007). Flow in games (and everything else). *Communications of the ACM*, 50(4) :34.

CORRUBLE, V. et RAMALHO, G. (2009). *Jeux vidéo et Systèmes Multi-Agents*, pages 235–264. IC2 Series. Hermès Lavoisier. isbn : 978-2-7462-1785-0.

COULOM, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy.*

COULOM, R. (2007). Computing "Elo ratings" of move patterns in the game of go. *ICGA Journal*, 4(30) :198–208.

CSIKSZENTMIHALYI, M. et CSIKSZENTMIHALYI, I. (1975). *Beyond boredom and anxiety.* Jossey-Bass San Francisco.

CSIKSZENTMIHALYI, M. et CSIKZENTMIHALY, M. (1990). *Flow : The psychology of optimal experience.* Harper & Row New York.

CULBERSON, J. C. et SCHAEFFER, J. (1998). Pattern Databases. *Computational Intelligence*, 4(14) :318–334.

De MESMAY, F., RIMMEL, A., VORONENKO, Y. et PÜSCHEL, M. (2009). Bandit-Based Optimization on Graphs with Application to Library Performance Tuning. *In ICML*, Montréal Canada.

DEGRIS, T., SIGAUD, O. et WUILLEMIN, P. (2009). Apprentissage par renforcement factorisé pour le comportement de personnages non joueurs . *Revue d'Intelligence Artificielle*, 23(2) :221–251.

DONNINGER, C. (1993). Null move and deep search : selective search heuristics for obtuse chess programs. *ICCA Journal*, 16(3) :137–143.

FELNER, A., KORF, R. E. et HANAN, S. (2004). Additive pattern database heuristics. *J. Artif. Intell. Res. (JAIR)*, 22 :279–318.

FELNER, A., KORF, R. E., MESHULAM, R. et HOLTE, R. C. (2007). Compressed pattern databases. *J. Artif. Intell. Res. (JAIR)*, 30 :213–247.

FINNSSON, H. et BJÖRNSSON, Y. (2008). Simulation-based approach to general game playing. *In AAAI*, pages 259–264.

FLÓREZ-PUGA, G., GÓMEZ-MARTIN, M., DIAZ-AGUDO, B. et GONZÁLEZ-CALERO, P. (2008). Dynamic Expansion of Behaviour Trees. *In AAAI conference on Artificial Intelligence and Interactive Digital Entertainement*, pages 36–41.

GELLY, S., HOOCK, J. B., RIMMEL, A., TEYTAUD, O. et KALEMKARIAN, Y. (2008). The parallelization of monte-carlo planning. *In Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO 2008)*, pages 198–203. To appear.

GELLY, S., WANG, Y., MUNOS, R. et TEYTAUD, O. (2006). Modification of uct with patterns in monte-carlo go. Rapport de recherche INRIA RR-6062.

GREENBLATT, R., EASTLAKE, D. et CROCKER, S. (1967). The Greenblatt chess program. *In Fall Joint Computing Conference*, volume 31, pages 801–810, New york ACM.

HAREL, D. (1987). Statecharts : A visual formalism for complex systems.

HART, P., NILSSON, N. et RAPHAEL, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernet.*, 4(2) :100–107.

HOANG, H., LEE-URBAN, S. et MUÑOZ-AVILA, H. (2005). Hierarchical plan representations for encoding strategic game ai. *In Proc. Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05).*

IERUSALIMSCHY, R., de FIGUEIREDO, L. H. et CELES, W. (2007). The evolution of lua. *In HOPL III : Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 2–1–2–26, New York, NY, USA. ACM.

JUNGHANNS, A. (1998). Are there practical alternatives to Alpha-Beta ? *ICCA Journal*, 21(1) :14–32.

JUNGHANNS, A. et SCHAEFFER, J. (2001). Sokoban : Enhancing general single-agent search methods using domain knowledge. *Artif. Intell.*, 129(1-2) :219–251.

KENDALL, G., PARKES, A. et SPOERER, K. (2008). A survey of NP-complete puzzles. *ICGA Journal*, 31(1) :13–34.

KNUTH, D. et MOORE, R. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6 :293–326.

KOCSIS, L. et SZEPESVARI, C. (2006). Bandit-based monte-carlo planning. *In ECML'06*, pages 282–293.

KORF, R. (1985a). Depth-first Iterative Deepening : an Optimal Admissible Tree Search. *Artificial Intelligence*, 27 :97–109.

KORF, R. et CHICKERING, D. (1994). Best-first search. *Artificial Intelligence*, 84 :299–337.

KORF, R. E. (1985b). Depth-first iterative-deepening : an optimal admissible tree search. *Artificial Intelligence*, 27(1) :97–109.

KORF, R. E. (1997). Finding optimal solutions to rubik's cube using pattern databases. *In AAAI-97*, pages 700–705.

LAIRD, J. E. (2002). Research in human-level ai using computer games. *Commun. ACM*, 45(1) :32–35.

LEE, C.-S., WANG, M.-H., CHASLOT, G., HOOCK, J.-B., RIMMEL, A., TEYTAUD, O., TSAI, S.-R., HSU, S.-C. et HONG, T.-P. (2009). The computational intelligence of mogo revealed in taiwan's computer go tournaments. *IEEE Transactions on Computational Intelligence and AI in Games.*

MADEIRA, C. et CORRUBLE, V. (2009). Strada : une approche adaptative pour les jeux de stratégie modernes. *Revue d'Intelligence Artificielle*, 23(2) :293–326.

MAÎTREPIERRE, R., MARY, J. et MUNOS, R. (2008). Adaptative play in texas hold'em poker. *In European Conference on Artificial Intelligence - ECAI.*

MARSLAND, T. (1986). A review of game-tree pruning. *ICCA Journal*, 9(1) :3–19.

MCALLESTER, D. (1988). Conspiracy Numbers for Min-Max Search. *Artificial Intel-*

ligence, 35 :287–310.

MÜLLER, M. (2002). Computer go. *Artificial Intelligence*, 134(1-2) :145–179.

NATKIN, S. (2004). *Jeux vidéo et médias du XXIe siècle : quels modèles pour les nouveaux loisirs numériques ?* Vuibert.

OCHS, M., SABOURET, N. et CORRUBLE, V. (2009). Simulation de la dynamique des emotions et des relations sociales de personnages virtuels. *Revue d'Intelligence Artificielle*, 23(2) :327–358.

PEARL, J. (1980a). Asymptotic properties of minimax trees and game-searching procedures. *Artificial Intelligence*, 14 :113–138.

PEARL, J. (1980b). SCOUT : a simple game-searching algorithm with proven optimal properties. *In Proceedings of the First Annual National Conference on Artificial Intelligence*, pages 143–145.

PERLIN, K. (2005). Toward interactive narrative. *In International Conference on Virtual Storytelling*, pages 135–147. Springer.

PITRAT, J. (1968). Realization of a general game-playing program. *In IFIP Congress (2)*, pages 1570–1574.

PITRAT, J. (1976). A program for learning to play chess. *In* CHEN, C. H., éditeur : *Pattern Recognition and Artificial Intelligence*, pages 399–419. Academic Press, New York.

PLAAT, A., SCHAEFFER, J., PILS, W. et de BRUIN, A. (1996). Best-first fixed depth minimax algorithms. *Artificial Intelligence*, 87 :255–293.

RABIN, S. (2002). *AI game programming wisdom.* Charles River Media.

RABIN, S. (2003). *AI Game Programming Wisdom, Vol. 2, Charles River Media.* Charles River Media.

RABIN, S. (2006). *AI Game Programming Wisdom 3 (Game Development Series).* Charles River Media.

RICKEL, J., MARSELLA, S., GRATCH, J., HILL, R., TRAUM, D. et SWARTOUT, W. (2002). Toward a new generation of virtual humans for interactive experiences. *IEEE Intelligent Systems*, pages 32–38.

RIVEST, R. (1988). Game-tree searching by min-max approximation. *Artificial Intelligence*, 34(1) :77–96.

ROLET, P., SEBAG, M. et TEYTAUD, O. (2009). Optimal active learning through billiards and upper confidence trees in continous domains. *In Proceedings of the ECML conference.*

ROLET, P., SEBAG, M. et TEYTAUD, O. (2009). Optimal robust expensive optimization is tractable. *In Gecco 2009*, page 8 pages, Montréal Canada. ACM.

ROMEIN, J. W. et BAL, H. E. (2003). Solving awari with parallel retrograde analysis. *IEEE Computer*, 36(10) :26–33.

SCHAEFFER, J. (1989). The history heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transcations on Pattern Analysis and Machine Intelligence*, 11(11) : 1203–1212.

SCHAEFFER, J. (1990). Conspiracy Numbers. *Artificial Intelligence*, 43 :67–84.

SCHAEFFER, J., BURCH, N., BJORNSSON, Y., KISHIMOTO, A., MULLER, M., LAKE, R., LU, P. et SUTPHEN, S. (2007). Checkers is solved. *Science.*

SCHAEFFER, J. et van den HERIK, J. (2002). Games, Computers, and Artificial Intelligence. *Artificial Intelligence*, 134 :1–7.

SHANNON, C. (1950). Programming a computer to play Chess. *Philosoph. Magazine*, 41 :256–275.

SLATE, D. et ATKIN, L. (1977). Chess 4.5 - the northwestern university chess program. *In* FREY, P., éditeur : *Chess Skill in Man and Machine*, pages 82–118. Springer-Verlag.

SPRONCK, P., PONSEN, M., SPRINKHUIZEN-KUYPER, I. et POSTMA, E. (2006). Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3) :217–248.

STOCKMAN, G. (1979). A minimax algorithm better than Alpha-Beta? *Artificial Intelligence*, 12 :179–196.

STURTEVANT, N. R., FELNER, A., BARRER, M., SCHAEFFER, J. et BURCH, N. (2009). Memory-based heuristics for explicit state spaces. *In IJCAI*, pages 609–614.

THIERY, C. et SCHERRER, B. (2009). Construction d'un joueur artificiel pour Tetris. *Revue d'Intelligence Artificielle*, 23(2-3) :387–407.

THOMPSON, K. (1996). 6-piece endgames. *ICCA Journal*, 19(4) :215–226.

THOMSEN, T. (2002). Lambda-search in game trees - with application to Go. *In* MARSLAND, T. A. et FRANK, I., éditeurs : *Computers and Games*, volume 2063 de *Lecture Notes in Computer Science*, pages 19–38. Springer.

von NEUMANN, J. et MORGENSTERN, O. (1944). *Theory of Games and Economic Behavior.* Princeton University Press.

ZOBRIST, A. (1990). A new hashing method with application for game playing. *ICCA Journal*, 13(2) :69–73.

Chapitre 3

Déduction automatique

Après un aperçu historique délimitant les enjeux de la déduction automatique (DA) et une rapide définition de la logique du premier ordre, nous développons les différents aspects de la méthode de résolution puis de la méthode des tableaux sémantiques. Celle-ci s'adapte bien aux logiques non classiques dont certains exemples sont ensuite exposés. Puis nous évoquons les enjeux liés à l'incomplétude inévitable en mathématique. Par manque de place, nous ne pouvons que survoler certaines questions ou développements récents qui sont toutefois indiqués dans une bibliographie succincte ou seulement par leur noms.

3.1 Introduction

La déduction automatique étudie l'utilisation des ordinateurs pour aider les humains à découvrir et écrire des preuves formelles. On peut considérer ce domaine comme l'aboutissement de deux traditions intellectuelles, l'une tendant à étudier le raisonnement humain et dont les origines se trouvent en Inde et dans la Grèce antique, c'est bien sûr la logique, l'autre consistant à construire des outils pour faciliter certaines tâches administratives ou commerciales qui sont de nature mathématique, comme les abaques. L'arrivée de l'ordinateur a eu comme conséquence l'apparition d'une discipline de recherche nouvelle centrée sur la conception de programmes pour le raisonnement (essentiellement déductif). Les liens entre cette discipline et l'intelligence artificielle (IA) sont naturellement très forts. Le programme qui est considéré comme la première réalisation pratique en IA est le démonstrateur de théorèmes « *Logic Theory Machine* » de Newell, Shaw et Simon, en 1956. La proximité de ces deux domaines de recherche, qui fêtent simultanément leur demi-siècle d'existence, est présagée par les assertions de deux grands mathématiciens que l'on peut classer parmi leurs précurseurs :

> L'idée maîtresse de ma théorie de la démonstration n'est rien d'autre que de dépeindre l'activité de notre intelligence, de dresser un inventaire des règles suivant lesquelles notre pensée fonctionne réellement (D. Hilbert).

Auteurs : Thierry Boy de la Tour, Ricardo Caferra, Nicola Olivetti, Nicolas Peltier et Camilla Schwind.

> Je veux mettre en place un système formel qui soit aussi proche que possible du raisonnement réel (G. Gentzen à propos de la déduction naturelle).

Le premier programme publié en déduction automatique semble être une procédure de décision pour l'arithmétique de Presburger par M. Davis, ce qui montre que lors des premières tentatives d'automatisation du raisonnement, on cherchait des programmes qui terminent. Les programmes plus ambitieux pour la logique du premier ordre montreront vite les limites posées par l'explosion combinatoire [Siekmann et Wrightson, 1983].

Dès les années 1970, on peut identifier deux approches, l'une cherchant à imiter les raisonnements du mathématicien [Bledsoe et Loveland, 1984], l'autre tentant d'implémenter des systèmes déductifs (« proof systems » en anglais). La première approche permit de démontrer quelques théorèmes très intéressants principalement en analyse et en topologie, mais rapidement la deuxième s'imposa, en laissant implicitement « pour plus tard » les techniques puissantes, mais trop difficiles à formaliser, considérées dans la première. Certains travaux, comme ceux sur le « robot mathématicien » [Sloman, 2008] et sur « les défis des mathématiques par ordinateur » [Barendregt et Wiedijk, 2005] illustrent les deux approches dans leur maturité actuelle. Outre les progrès théoriques amenés par l'étude des procédures de preuve et l'accroissement de la puissance des machines, le développement d'outils pour l'évaluation pratique des systèmes (bases de benchmarks, compétition de systèmes,...) ont conduit à une amélioration sensible des performances.

On ne peut donner en quelques pages une vue exhaustive d'un domaine de recherche aussi riche. Nous nous contenterons de donner dans ce chapitre un aperçu des techniques les plus couramment utilisées ainsi que quelques repères concernant les thèmes de recherche qui nous semblent les plus importants.

3.2 La logique du premier ordre

La logique propositionnelle (LP) est très utile pour formaliser et vérifier des systèmes finis (voir le chapitre II.5) mais ne suffit plus quand il s'agit d'exprimer des propriétés concernant une infinité d'objets. La logique du premier ordre (L1O) augmente énormément le pouvoir d'expression mais elle n'est plus décidable ; elle est cependant *semi-décidable*, c'est-à-dire qu'il existe un algorithme permettant de détecter toutes les formules valides, mais qui peut ne pas terminer lorsque la formule n'est pas valide.

On se donne une *signature* $\Omega = \{\mathcal{V}, \mathcal{F}, \mathcal{P}\}$ contenant un ensemble dénombrable $\mathcal{V}$ de *variables* et deux ensembles au plus dénombrables $\mathcal{F}$ de *symboles fonctionnels* et $\mathcal{P}$ de *symboles de prédicat.* À chaque symbole de $\mathcal{F} \cup \mathcal{P}$ est associé un unique entier (positif ou nul) appelé son *arité.* L'ensemble des *termes* construits sur $\{\mathcal{V}, \mathcal{F}\}$ est le plus petit ensemble $\Sigma(\mathcal{V}, \mathcal{F})$ contenant $\mathcal{V}$ et tel que si $f \in \mathcal{F}$ est d'arité n et $t_1, \ldots, t_n \in \Sigma(\mathcal{V}, \mathcal{F})$ alors $f(t_1, \ldots, t_n) \in \Sigma(\mathcal{V}, \mathcal{F})$. Si $n = 0$ le terme est une *constante* (noté simplement f). Si t ne contient pas de variable on dit que c'est un terme *fermé* (ou *clos*).

Le langage (ou l'ensemble des *formules*) de la L1O est le plus petit ensemble $\mathcal{L}_\Omega$ tel que :

— si $p \in \mathcal{P}$ est un symbole de prédicat d'arité n et si $t_1, \ldots, t_n$ sont des termes alors $p(t_1, \ldots, t_n) \in \mathcal{L}_\Omega$ (si $n = 0$ alors la formule est simplement notée p), si t et t' sont des termes alors $t \simeq t' \in \mathcal{L}_\Omega$ et $\Box \in \mathcal{L}_\Omega$; ces formules[1] sont dites *atomiques*,

— si $\varphi, \psi \in \mathcal{L}_\Omega$ et $x \in \mathcal{V}$ alors les formules $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$, $\forall x\, \varphi$ et $\exists x\, \varphi$ sont dans $\mathcal{L}_\Omega$.

Un *littéral* est soit une formule atomique soit la négation d'une formule atomique. Un littéral et sa négation sont dits *complémentaires*. Le littéral $\neg\, t \simeq t'$ est noté $t \not\simeq t'$. On note $\mathcal{P}_0$ l'ensemble des $p \in \mathcal{P}$ d'arité nulle ; ce sont les *symboles propositionnels*. La LP est le fragment de la L1O obtenue avec $\mathcal{V} = \mathcal{F} = \varnothing$ et $\mathcal{P} = \mathcal{P}_0$.

En pratique de nombreuses parenthèses sont omises et des règles de priorités sont utilisées pour ordonner les différents connectifs (avec la précédence usuelle : les connectifs $\neg$, $\forall$ et $\exists$ en premier et $\wedge > \vee > \Rightarrow$). Une variable x est dite *liée* dans une formule φ si elle y apparaît sous la portée d'un quantificateur $\forall x$ ou $\exists x$. Ces variables peuvent être renommées ; c'est pourquoi nous supposerons que dans toute formule chaque variable liée n'est quantifiée qu'une seule fois et que les variables non liées, dites *libres*, sont distinctes des variables liées. Une formule sans variable libre est dite *fermée*.

La sémantique est définie comme suit. Soit D un ensemble non vide, une *interprétation* $\mathcal{I}$ de domaine D associe à chaque élément $f \in \mathcal{F}$ d'arité n une fonction (totale) de D^n vers D et à chaque symbole de prédicat p d'arité n une relation dans D^n. Les objets sémantiques assignés aux objets syntaxiques sont identifiés par le nom de ces derniers avec l'exposant $\mathcal{I}$, ainsi $f^{\mathcal{I}}$ dénote la fonction associée au symbole f dans l'interprétation $\mathcal{I}$. Une *valuation* σ d'un ensemble de variables $V \subseteq \mathcal{V}$ dans un domaine D est une fonction de V dans D.

Un couple $(\mathcal{I}, \sigma)$ associe à tout terme $t \in \Sigma(V, \mathcal{F})$ une valeur dans D, notée $[t]_{(\mathcal{I},\sigma)}$ et définie inductivement comme suit : pour tout $x \in V$ on a $[x]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \sigma(x)$ et pour tous $t_1, \ldots, t_n \in \Sigma(V, \mathcal{F})$ et $f \in \mathcal{F}$ d'arité n, $[f(t_1, \ldots, t_n)]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} f^{\mathcal{I}}([t_1]_{(\mathcal{I},\sigma)}, \ldots, [t_n]_{(\mathcal{I},\sigma)})$.

De même si σ est une valuation des variables libres d'une formule φ, alors $(\mathcal{I}, \sigma)$ associe à φ une *valeur de vérité* V ou F, notée $[\varphi]_{(\mathcal{I},\sigma)}$ et inductivement définie de la manière suivante :

1. $[\Box]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{F}$,
2. $[p(t_1, \ldots, t_n)]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi $\langle [t_1]_{(\mathcal{I},\sigma)}, \ldots, [t_n]_{(\mathcal{I},\sigma)} \rangle \in p^{\mathcal{I}}$,
3. $[t \simeq t']_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi $[t]_{(\mathcal{I},\sigma)} = [t]_{(\mathcal{I},\sigma)}$,
4. $[\neg\varphi]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi $[\varphi]_{(\mathcal{I},\sigma)} = \text{F}$,
5. $[\varphi \wedge \psi]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi $[\varphi]_{(\mathcal{I},\sigma)} = \text{V}$ et $[\psi]_{(\mathcal{I},\sigma)} = \text{V}$,
6. $[\varphi \vee \psi]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi $[\varphi]_{(\mathcal{I},\sigma)} = \text{V}$ ou $[\psi]_{(\mathcal{I},\sigma)} = \text{V}$,
7. $[\varphi \Rightarrow \psi]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi $[\varphi]_{(\mathcal{I},\sigma)} = \text{F}$ ou $[\psi]_{(\mathcal{I},\sigma)} = \text{V}$,
8. $[\exists x\, \varphi]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi il existe $a \in D$ tel que $[\varphi]_{(\mathcal{I},\sigma \cup \{x \mapsto a\})} = \text{V}$,
9. $[\forall x\, \varphi]_{(\mathcal{I},\sigma)} \stackrel{\text{def}}{=} \text{V}$ ssi pour tout $a \in D$ on a $[\varphi]_{(\mathcal{I},\sigma \cup \{x \mapsto a\})} = \text{V}$.

1. La formule $\Box$ n'est utile que dans les preuves par réfutation.

La relation de *satisfaction* reliant les interprétations aux formules est alors définie comme suit : $\mathcal{I} \models \varphi$ ssi pour toute valuation σ dans le domaine de $\mathcal{I}$ des variables libres de φ on a $[\varphi]_{(\mathcal{I},\sigma)} = \text{V}$. On dit alors que φ est *satisfaite* par $\mathcal{I}$ et que $\mathcal{I}$ est un *modèle de* φ ; dans le cas contraire (il existe une valuation σ telle que $[\varphi]_{(\mathcal{I},\sigma)} = \text{F}$) on dit que $\mathcal{I}$ est un *contre-modèle* de φ. Si φ est fermée on note $[\varphi]_{(\mathcal{I})}$ pour $[\varphi]_{(\mathcal{I},\varnothing)}$.

Un *modèle de Herbrand* est une interprétation $\mathcal{I}$ de domaine[2] $\Sigma(\varnothing, \mathcal{F})$ telle que pour tout $f \in \mathcal{F}$ et $t_1, \ldots, t_n \in \Sigma(\varnothing, \mathcal{F})$ on a $f^{\mathcal{I}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$. Ces modèles jouent un rôle important en DA, en particulier grâce au théorème de Herbrand qui permet de ramener la preuve de l'insatisfaisabilité d'une formule universelle (de la forme $\forall x_1 \cdots \forall x_n \varphi$ où φ n'a pas de quantificateurs) à celle d'un ensemble fini (qui reste à trouver) d'instances fermées de φ (et d'axiomes pour $\simeq$), ce qui est un problème propositionnel.

3.3 La méthode de résolution

La règle de résolution, due à J.A. Robinson (ainsi que la règle de paramodulation dédiée au traitement de l'égalité) sont depuis leur publication dans les années 1960 les plus connues du domaine. Leur succès vient de leur grande simplicité obtenue grâce à l'utilisation d'une *forme normale* pour les formules ainsi que d'un algorithme fondamental : *l'unification.* Les problèmes essentiels pour ces règles sont, bien entendu, les stratégies et un traitement adéquat de l'égalité. Pour les concepts, définitions de base et liste des résultats obtenus par les démonstrateurs, nous référons le lecteur à l'énorme bibliographie existante (on peut commencer par [Wos *et al.*, 1992 ; Wos, 1988 ; Robinson et Voronkov, 2001]).

3.3.1 Mise sous forme clausale

L'une des formes de raisonnement les plus simples est d'effectuer des *calculs algébriques*, ce qui est possible sur des formules grâce à certaines propriétés comme la commutativité et l'associativité du $\wedge$ et du $\vee$, ou les lois de dualité :

$$\begin{array}{cc} \neg(\varphi \wedge \psi) = \neg\varphi \vee \neg\psi & \neg(\varphi \vee \psi) = \neg\varphi \wedge \neg\psi \\ \neg\forall x\, \varphi = \exists x\, \neg\varphi & \neg\exists x\, \varphi = \forall x\, \neg\varphi \end{array} \qquad \neg\neg\varphi = \varphi \tag{3.1}$$

Ces propriétés ainsi que les autres propriétés des algèbres booléennes sont utilisées de façon quasi réflexe pour simplifier les problèmes mathématiques. Automatiser ces réflexes n'est pas difficile : on voit par exemple qu'en appliquant les identités (3.1) de gauche à droite (et en exprimant l'implication et l'équivalence au moyen de $\wedge$, $\vee$ et $\neg$), on peut déplacer toutes les négations vers les formules atomiques. On déplace les quantificateurs en sens inverse par :

$$\begin{array}{ll} \forall x\, \varphi \wedge \forall y\, \psi = \forall x\, (\varphi \wedge \psi[y/x]) & \varphi \wedge \exists x\, \psi = \exists x\, (\varphi \wedge \psi) \\ \exists x\, \varphi \vee \exists y\, \psi = \exists x\, (\varphi \vee \psi[y/x]) & \varphi \vee \forall x\, \psi = \forall x\, (\varphi \vee \psi) \end{array} \tag{3.2}$$

($\psi[y/x]$ est la formule ψ où y est remplacée par x) et obtenir ainsi une formule *prénexe* où les quantificateurs précèdent tous les autres connectifs. Enfin, la distributivité $\varphi \vee$

2. On suppose qu'il existe une constante dans $\mathcal{F}$ et donc que $\Sigma(\varnothing, \mathcal{F}) \neq \varnothing$.

$(\psi_1 \wedge \psi_2) = (\varphi \vee \psi_1) \wedge (\varphi \vee \psi_2)$ permet d'obtenir une forme normale conjonctive. Plus précisément :

- Une *clause* est une formule C de la forme $L_1 \vee \cdots \vee L_n$ où $n \geq 0$ et les L_i sont des littéraux. Si $n = 0$ on convient que $C = \square$; c'est la *clause vide.*
- Une *forme normale conjonctive* (ou *fnc*) est une formule prénexe dont la partie suivant les quantificateurs est de la forme $C_1 \wedge \cdots \wedge C_m$ où $m \geq 1$ et les C_i sont des clauses.
- une *forme clausale* est une fnc ne contenant que des quantificateurs universels. On peut la considérer comme un *ensemble de clauses* et de même une clause est souvent représentée comme un ensemble de littéraux.

Pour obtenir une forme clausale de ψ il nous faut encore éliminer les quantificateurs existentiels grâce à la *skolemisation* qui consiste à réécrire toute sous-formule $\exists y\, \varphi$ de ψ en $\varphi[y/f(x_1, \ldots, x_n)]$, où $x_1, \ldots, x_n, y$ sont les variables libres de φ et où f est un *nouveau* symbole de fonction n'apparaissant pas dans ψ. Mais nous sortons alors du cadre algébrique puisque la formule obtenue n'est pas équivalente à la précédente ; la valeur $y = f(x_1, \ldots, x_n)$ n'est en général correcte que pour certaines valeurs de f dépendantes de φ ; seule la satisfaisabilité de la formule réécrite est donc préservée.

On peut appliquer la skolemisation avant ou après les règles (3.2) afin de minimiser l'arité ou le nombre de fonctions de Skolem. Une technique similaire consiste à *renommer* dans ψ certains arguments des disjonctions, c'est-à-dire remplacer $\psi[\varphi \vee \varphi']$ par

$$\psi[p(x_1, \ldots, x_n) \vee \varphi'] \wedge \forall x_1 \ldots \forall x_n\, (\varphi \vee \neg p(x_1, \ldots, x_n)),$$

où $x_1, \ldots, x_n$ sont les variables libres de φ et p est un nouveau symbole de prédicat. La sous-formule $\varphi \vee \varphi'$ de ψ est donc remplacée par $p(x_1, \ldots, x_n) \vee \varphi'$, ce qui permet si φ' est une conjonction d'éviter la duplication de φ par distributivité et ainsi d'obtenir une forme clausale de complexité quadratique [Robinson et Voronkov, 2001, chapitre 6], voire minimiser le nombre de clauses [Boy de la Tour, 1992].

3.3.2 L'unification

Une *substitution* est une application σ de $\mathcal{V}$ dans $\Sigma(\mathcal{V}, \mathcal{F})$. Une substitution peut être étendue en une fonction associant à chaque terme $t \in \Sigma(\mathcal{V}, \mathcal{F})$ le terme $\sigma(t)$ obtenu en remplaçant dans t toutes les variables x par leur image par σ ; $\sigma(t)$ (souvent noté $t\sigma$) est une *instance* de t. On peut alors composer des substitutions σ et σ' ; on dit que $\sigma' \circ \sigma$ (souvent notée $\sigma\sigma'$) est une *instance* de σ.

Étant donné un ensemble de n équations entre termes $\{t_i = t'_i \mid 1 \leq i \leq n\}$ le problème de l'*unification* consiste à trouver, si elle existe, une substitution σ (appelée *unificateur*) telle que $\sigma(t_i) = \sigma(t'_i)$ pour $1 \leq i \leq n$. Si un ensemble possède un unificateur, alors il possède également un *unificateur le plus général (upg)*, dont tout unificateur est instance. L'algorithme 3.1 permet de construire un upg, sa complexité est exponentielle dans le pire des cas mais elle peut facilement être réduite à une complexité quadratique grâce au partage de structure [Robinson et Voronkov, 2001, chapitre 8].

Algorithme 3.1 : Un algorithme d'unification

Entrées : $\mathcal{S} = \{t_1 = s_1, \ldots, t_n = s_n\}$
Sorties : soit l'upg σ de $\mathcal{S}$, soit $\perp$ (pas de solution)

$\sigma \leftarrow \varnothing$ /* $\varnothing$ représente la fonction identité de $\mathcal{V}$ sur $\mathcal{V}$ */
tant que $\mathcal{S} \neq \varnothing$ **faire**
  Choisir $t = s \in \mathcal{S}$
  $\mathcal{S} \leftarrow \mathcal{S} \setminus \{t = s\}$
  si $t \neq s$ **alors**
    si $t = f(t_1, \ldots, t_n)$ *et* $s = g(s_1, \ldots, s_m)$ **alors**
      si $f \neq g$ **alors retourner** $\perp$
      sinon $\mathcal{S} := \mathcal{S} \cup \{t_i = s_i \mid i \in [1..n]\}$
    sinon si t *est une variable* **alors**
      si t *apparaît dans* s **alors retourner** $\perp$
      sinon $\sigma \leftarrow \sigma\{t \mapsto s\}$; $\mathcal{S} \leftarrow \mathcal{S}\{t \mapsto s\}$
    sinon
      /* s est une variable */
      si s *apparaît dans* t **alors retourner** $\perp$
      sinon $\sigma \leftarrow \sigma\{s \mapsto t\}$; $\mathcal{S} \leftarrow \mathcal{S}\{s \mapsto t\}$
retourner σ

3.3.3 La méthode de superposition

Plutôt que de présenter la méthode de résolution sous sa forme originelle nous présentons ce qui peut être considéré comme la procédure de preuve la plus performante pour la logique du premier ordre avec égalité : *le calcul de superposition* [Robinson et Voronkov, 2001, chapitre 7]. C'est une extension de la méthode de résolution qui peut aussi être vue comme une généralisation du calcul de complétion (sans échec) de Knuth-Bendix à des clauses quelconques.

Ce calcul s'applique uniquement sur des clauses et utilise une règle d'inférence spécifique à l'égalité, appelée *paramodulation*. Cette règle permet, étant donnée une équation $t \simeq s$, de remplacer n'importe quelle occurrence du terme t apparaissant dans une clause C contenant t (notée $C[t]$) par s. Le remplacement peut-être effectué à n'importe quelle profondeur dans C, par exemple à partir de $a \simeq b$ et $f(g(a), c) \simeq d$ on déduit $f(g(b), c) \simeq d$. Cette idée peut facilement être étendue à des clauses non unitaires : à partir de $t \simeq s \vee D$ et $C[t]$ on déduit une clause $C[t/s] \vee D$ où $C[t/s]$ est obtenue à partir de C en remplaçant t par s. Cette règle s'étend aux clauses non fermées en utilisant l'unification qui permet de trouver les instances les plus générales des clauses sur lesquelles la règle de paramodulation peut être appliquée [Siekmann et Wrightson, 1983].

Afin de réduire l'espace de recherche, des restrictions supplémentaires sont impo-

Paramodulation $C_1 : (t[w] \bowtie s) \vee \alpha,\ C_2 : (u \simeq v) \vee \beta \ \rightarrow \ (t[w/v] \bowtie s \vee \alpha \vee \beta)\sigma$
si $\sigma = upg(u, w)$, $v\sigma \not> u\sigma, s\sigma \not> t\sigma$, w n'est pas une variable,
$(t[w] \bowtie s)\sigma \in \text{sel}(C_1\sigma)$ et $(u \simeq v)\sigma \in \text{sel}(C_2\sigma)$.
Réflexivité $C : (t \not\simeq s) \vee \alpha \ \rightarrow \ \alpha\sigma$
si $\sigma = upg(t, s)$ et $(t \not\simeq s)\sigma \in \text{sel}(C\sigma)$.
Factorisation équationnelle $C : (t \simeq s) \vee (u \simeq v) \vee \alpha \ \rightarrow \ (s \not\simeq v \vee t \simeq s \vee \alpha)\sigma$
si $\sigma = upg(t, u)$, $s\sigma \not> t\sigma$, $v\sigma \not> u\sigma$ et $(t \simeq s)\sigma \in \text{sel}(C\sigma)$.

FIGURE 1 – Le calcul de superposition $\text{SP}^{\succ}_{\text{sel}}$

sées : en particulier le remplacement n'est effectué que si le terme s est « plus simple » que t (suivant un ordre $\succ$ supposé donné a priori). On parle alors de paramodulation orientée. De manière similaire, on peut ne remplacer que les termes apparaissant dans le terme maximal (suivant $\succ$) d'une équation. Par exemple si $f(a) \succ b \succ a$, alors la règle n'est pas applicable sur les clauses $f(a) \simeq b$ et $b \simeq a$. En effet, b ne peut pas être remplacé dans la première équation car il n'est pas maximal ($f(a) \succ b$), et le terme a dans $f(a)$ ne peut pas être remplacé par b puisque $b \succ a$.

L'application des règles est restreinte à certains littéraux des clauses parentes, dits *sélectionnés*. Les règles complètes sont données par la figure 1. Elles sont paramétrées par un ordre $\succ$ sur les termes et une fonction de sélection sel associant à toute clause C un ensemble de littéraux de C.

Par convention, le symbole $\bowtie$ est utilisé pour dénoter indifféremment $\simeq$ et $\not\simeq$. La règle de réflexivité appliquée à une clause unitaire $t \not\simeq s$ infère la clause vide $\square$. La règle de factorisation permet de fusionner deux littéraux $t \simeq s$ et $t \simeq v$ si s et v sont égaux.

Pour éviter d'avoir à utiliser des règles supplémentaires pour traiter les autres prédicats, on suppose que les littéraux non équationnels $P(t_1, \ldots, t_n)$ sont représentés sous forme d'équations $P(t_1, \ldots, t_n) \simeq \text{V}$ (en admettant V pour constante et P pour symbole de fonction). Le calcul de superposition simule la résolution [Leitsch, 1997] lorsque l'ensemble considéré est non équationnel (c'est-à-dire si tous les atomes sont de la forme $P(t) \simeq \text{V}$).

Le calcul de superposition est correct. Il est complet pour la réfutation[3] si $\succ$ et sel satisfont les propriétés suivantes :

(P_1) tous les symboles f sont monotones pour $\succ$

$$t_i \succ s \Rightarrow f(t_1, \ldots, t_n) \succ f(t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots, t_n),$$

et $\succ$ est stable par substitution : $t \succ s \Rightarrow t\sigma \succ s\sigma$; on dit que $\succ$ est un *ordre de réduction*.

(P_2) Pour toute clause C, soit $\text{sel}(C)$ contient un littéral négatif de C, soit $\text{sel}(C)$ contient tous les littéraux maximaux de C ($\succ$ est étendu aux littéraux et aux clauses en utilisant l'extension ensembliste usuelle).

Des variantes du calcul de superposition existent, par exemple la superposition *basique*, qui interdit le remplacement de terme à l'intérieur des termes engendrés par

3. une *réfutation* d'un ensemble H d'hypothèses est une déduction de $\square$ à partir de H.

unification, ou encore la superposition *stricte*, où la factorisation équationnelle est remplacée par la factorisation standard. Ce dernier calcul n'est cependant pas complet si la règle d'élimination de tautologie est appliquée (voir la section suivante).

3.3.4 Élimination de la redondance

La superposition engendre un grand nombre de clauses. Pour réduire l'espace de recherche, des critères ont été définis permettant d'éliminer certaines clauses engendrées.

Définition 53. Une clause C est dite *redondante* par rapport à un ensemble de clauses S et un ordre $\preceq$ si pour toute substitution fermée σ il existe des clauses $C_1, \ldots, C_n \in S$ et des substitutions fermées $\theta_1, \ldots, \theta_n$ telles que $\forall i \in [1..n], C\sigma \succeq C_i\theta_i$ et $\{C_1\theta_1, \ldots, C_n\theta_n\} \models C\sigma$.

Un ensemble S est dit *saturé* si toute clause déductible de S par les règles de $\mathrm{SP}^{\succ}_{\mathbf{sel}}$ est redondante par rapport à S. Le théorème suivant établit la complétude du calcul de superposition en présence de règles d'élimination des clauses redondantes.

Théorème 35. (Bachmair et Ganzinger) On suppose que $\succ$ et sel satisfont les propriétés (P_1) et (P_2) de la section 3.3.3. Pour tout ensemble de clauses S, si S est saturé et ne contient pas $\square$ alors S est satisfaisable.

La relation de redondance étant indécidable, on doit se contenter de critères restreints :

Définition 54.
— (Tautologies) Une clause C est appelée une *tautologie* si elle contient deux littéraux complémentaires ou un littéral de la forme $t \simeq t$.
— (Subsomption) Une clause C *subsume* une clause D ssi il existe une substitution σ telle que $C\sigma \subseteq D$.
— (Simplification équationnelle) Une clause C est appelée une *simplification* d'une clause D dans un ensemble S s'il existe une clause $(t \simeq s) \vee \alpha \in S$ et une substitution σ telle que $C = D[t\sigma/s\sigma]$, $t\sigma \succ s\sigma$ et $\alpha\sigma \subseteq C$.

On vérifie facilement qu'une clause C qui est une tautologie, une clause subsumée par une clause de S ou possédant une simplification dans S est nécessairement redondante par rapport à S (la réciproque n'est pas vraie).

3.3.5 Techniques d'implémentation

Sans technique d'implémentation efficace on n'aurait pas pu attaquer des problèmes complexes et les systèmes actuels ne pourraient pas être utilisés aussi massivement qu'ils le sont. Nous mentionnons deux de ces techniques, le *partage de structure* et l'*indexation*.

Des techniques de partage de structure ont été appliquées dès le début des implémentations de démonstrateurs automatiques (elles avaient déjà été utilisées dans l'implémentation des langages de programmation). Ainsi les travaux pionniers de Boyer et

Moore (publiés en 1972) proposent de ne pas construire effectivement les clauses déduites et de ne pas appliquer effectivement les substitutions mais de les garder comme contexte. L'espace requis est donc indépendant du nombre de littéraux dans les clauses et de la profondeur des termes. Les littéraux et les clauses ainsi que le contexte sont représentés comme des listes. Par exemple, la liste

$$\langle (q(y,y),2),\ (p(x,y),2),\ (\neg p(x,f(y,z)),4)\rangle \tag{3.3}$$

(les entiers dénotent les indices distinguant les variables) dans le contexte

$$\{x_2 \mapsto x_3,\ y_2 \mapsto f(x_4,y_4),\ y_4 \mapsto x_3,\ z_4 \mapsto f(x_2,y_2)\}$$

représente la clause

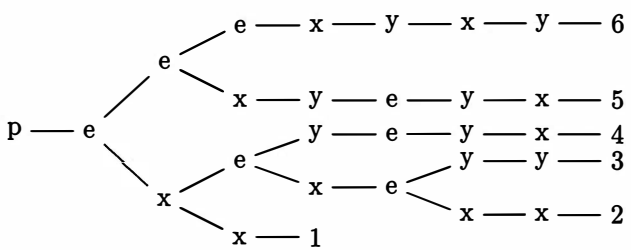
$$q(f(x_4,x_3),f(x_4,x_3)) \vee p(x_3,f(x_4,x_3)) \vee \neg p(x_4,f(x_3,f(x_3,f(x_4,x_3)))).$$

Les littéraux de la clause (3.3) proviennent des clauses prémisses d'une inférence : il suffit donc de conserver des pointeurs vers ces clauses et quelques informations qui permettent de poursuivre la recherche d'une réfutation. Certaines techniques de partage de structure peuvent être intégrées au niveau de la procédure de preuve ce qui permet d'obtenir des calculs plus efficaces ou de réduire la longueur des preuves.

L'indexation est nécessaire à l'application de règles telles que la subsomption « en avant » (qui évite de produire des clauses subsumées par des clauses existantes) et la simplification équationnelle. L'idée est de construire une structure de données (appelé *index*) permettant d'accéder de manière efficace aux termes contenus dans les clauses (sans avoir à parcourir systématiquement l'ensemble des clauses). Ce type de technique est utilisé de manière intensive dans d'autres domaines, comme en programmation logique. Par exemple, l'ensemble de clauses unitaires :

1. $p(e(x,x))$
2. $p(e(x,e(x,e(x,x))))$
3. $p(e(x,e(x,e(y,y))))$
4. $p(e(x,e(y,e(y,x))))$
5. $p(e(e(x,y),e(y,x)))$
6. $p(e(e(e(x,y),x),y))$

est représenté de la manière suivante :



Cette méthode d'indexation (due à W. McCune) a deux propriétés intéressantes : chaque branche de l'arbre représente un littéral et aucun nœud n'a plus d'un fils avec

la même étiquette (partage de structure optimal). Il existe plusieurs techniques d'indexation et il est impossible de les décrire en détail (voir [Robinson et Voronkov, 2001, chapitre 26] pour plus de précisions).

Ces techniques ont permis de réaliser des systèmes très performant utilisant la résolution et la superposition avec lesquels on a pu obtenir (de manière automatique ou semi-automatique) des résultats importants en mathématiques, en particulier des preuves de conjectures. Citons par exemple les preuves de l'indépendance d'axiomes dans la logique ternaire (avec une opération à 3 arguments), des preuves de résultats dans l'algèbre de Robbins, en théorie des groupes, en « equivalential calculus », en logique combinatoire et autres. Le lecteur intéressé peut trouver des détails sur le site www.mcs.anl.gov/research/projects/AR/new_results et dans [Wos, 1988 ; Wos *et al.*, 1992].

Nous mentionnons parmi les systèmes disponibles sur le web quelques-uns des plus connus et des plus performants : Prover9 (le successeur d'OTTER), Vampire (qui a remporté durant neuf années la compétition CASC dans la division principale), le *E*-Prover, SPASS. Tous ces systèmes utilisent la méthode de résolution ou de superposition.

3.3.6 Terminaison : quelques classes décidables

La L1O est indécidable : il n'existe pas d'algorithme permettant de décider en un temps fini si une formule est valide ou non. Il est donc naturel de chercher des sous-classes pour lesquelles ce problème est décidable et en particulier pour lesquelles la procédure de superposition termine [Robinson et Voronkov, 2001, chapitre 25]. Dans un grand nombre de cas, la terminaison peut être assurée par un choix judicieux de la fonction de sélection et de l'ordre.

Par exemple la terminaison peut être assurée pour la classe monadique avec égalité. Une formule est dite *monadique* si elle ne contient aucun symbole de fonction d'arité supérieure à 0 et aucun symbole de prédicat d'arité supérieur à 1. Dans ce cas, les termes apparaissant dans la forme clausale de la formule satisfont certaines propriétés de régularité (préservées par les règles d'inférence) qui permettent de prouver que la profondeur et la longueur des clauses sont bornées, ce qui implique que l'ensemble des clauses engendrées est fini (pour un ordre adéquat). Des résultats similaires peuvent être établis pour la classe Ackermann qui est l'ensemble des formules de la forme $\exists x_1, \ldots, x_n \forall y \exists z_1, \ldots, z_n \varphi$ où φ ne contient pas de symbole de fonction.

Des résultats de terminaison ont aussi été obtenus pour des théories plus spécifiques, notamment celles utilisées en vérification de programmes, comme la théorie des tableaux ou des listes. Ces résultats permettent d'utiliser le calcul de superposition pour tester la satisfaisabilité d'une formule fermée modulo ces théories (ce problème est appelé *problème SMT* pour « *Satisfiability Modulo Theory* »). Des stratégies de résolution ont également été utilisées pour prouver la décidabilité de classes utiles pour la vérification de protocoles cryptographiques.

Il est également naturel de s'intéresser à des sous-classes de la L1O obtenues par traduction à partir de logiques décidables. Il existe une procédure de preuve par superposition pour le fragment gardé avec égalité, une sous-classe de L1O vers laquelle de nombreuses logiques non classiques peuvent être traduites. Une procédure de décision

par résolution a été proposée pour le fragment à deux variables. Certaines procédures de décision pour des logiques de la description sont également fondées sur des raffinements de $\mathrm{SP}^{\succ}_{\mathbf{sel}}$.

Les résultats mentionnés ci-dessus sont fondés essentiellement sur le choix d'un bon ordre de réduction $\succ$. Cependant, des résultats de terminaison peuvent aussi être obtenus par un choix adéquat de la fonction de sélection. On peut s'intéresser en particulier au cas où $\mathrm{sel}(C)$ contient toujours au moins un littéral négatif (pour toute clause C contenant un littéral négatif). Une telle stratégie est dite *positive*. Leitsch [1997] propose une technique générale permettant de prouver la terminaison de la stratégie positive (technique qui a été étendue au cas équationnel). Cette approche peut-être utilisée pour montrer la décidabilité de certaines classes, telle que PVD (une généralisation de DATALOG). Des résultats de terminaison de la stratégie positive ont également été établis pour la classe $\mathcal{BU}$ qui est obtenue par traduction à partir de nombreuses logiques (logiques modales, logique de description).

3.3.7 Les méthodes par instanciation (hyperlinking)

Le théorème de Herbrand permet de ramener la preuve de l'insatisfaisabilité d'un ensemble de clauses de la L1O à celle d'un ensemble fini de clauses de la LP. Ce résultat permet de définir une procédure de semi-décision immédiate pour la L1O. En pratique, cette procédure naïve se révèle inefficace. Cependant, des techniques existent pour réduire le nombre de clauses engendrées ce qui permet d'obtenir des procédures de preuve raisonnablement efficaces.

Elles sont fondées sur une règle d'instanciation appelée *hyperlinking* (cf. figure 2). Les clauses engendrées sont des instances de l'une des clauses parentes. En appliquant de manière systématique et répétée la règle ci-dessus sur l'ensemble de clauses S à réfuter, on obtient un ensemble (en général infini) de clauses $hl(S)$. Afin d'obtenir un ensemble propositionnel, il suffit d'instancier toutes les variables restantes par une constante $\bot$; l'ensemble obtenu est noté $hl(S)\bot$. Évidemment, cet ensemble étant infini il ne peut être explicitement calculé, mais des sous-ensembles finis de plus en plus grands peuvent facilement être engendrés et soumis à un « SAT solver » afin de détecter l'insatisfaisabilité. Si l'un de ces sous-ensembles est insatisfaisable, alors S l'est également et la procédure s'arrête.

La procédure obtenue est complète pour les formules sans $\simeq$: S est insatisfaisable si et seulement si $hl(S)\bot$ est insatisfaisable. Par conséquent, si S est insatisfaisable et si la règle d'hyperlinking est appliquée de manière équitable (sans négliger de clause) alors on obtient après un temps fini un sous-ensemble de $hl(S)\bot$ qui est insatisfaisable.

Il existe divers raffinements de cette technique, par exemple des extensions au cas équationnel ont été considérées. D'autres approches combinent la règle d'hyperlinking avec une procédure de preuve pour la logique propositionnelle, telle que la méthode des tableaux, ou la procédure de Davis et Putnam. Au lieu d'avoir un module générant les instances et appelant ensuite un « *SAT solver* » indépendant vu comme une procédure subordonnée ou une « boîte noire », les deux aspects – instanciation et recherche de preuve – sont fusionnés et interconnectés au sein d'une seule et même procédure. L'avantage est que les instances engendrées sont spécifiques à une branche particulière dans l'arbre de recherche.

$$\frac{L(t_1,\ldots,t_n) \vee C \qquad \neg L(s_1,\ldots,s_n) \vee D}{(L(t_1,\ldots,t_n) \vee C)\sigma,\ (\neg L(s_1,\ldots,s_n) \vee D)\sigma}$$

Si σ est l'unificateur le plus général de $t_1,\ldots,t_n$ et $s_1,\ldots,s_n$.

FIGURE 2 – Règle d'hyperlinking

3.3.8 *E*-unification

Pour tenir compte d'une théorie axiomatique dans le calcul par résolution il suffit d'ajouter aux clauses du problème celles correspondant aux axiomes de cette théorie. Si la théorie est axiomatisée par des équations on doit donc ajouter les équations correspondantes. Cette approche est inélégante (on ne se réfère pas explicitement aux axiomes de l'arithmétique pour décider que $(x + y) + z = (z + y) + x$) et souvent inefficace (les équations ajoutées peuvent inférer de nouvelles équations en très grand nombre). Il peut être beaucoup plus simple de tenir compte d'une théorie équationnelle E *dans* l'algorithme d'unification utilisé par la résolution, ce qui est appelé *E-unification* [Robinson et Voronkov, 2001, chapitre 8].

Par exemple, si f dénote une fonction associative-commutative, l'équation $f(a,\ f(b,z)) = f(x,\ y)$ a comme solutions

$$\{x \mapsto a, y \mapsto f(b,z)\}, \{x \mapsto b, y \mapsto f(a,z)\}, \{x \mapsto z, y \mapsto f(a,b)\}.$$

On peut vérifier facilement que l'on ne peut pas transformer un de ces unificateurs en un autre par application d'une substitution ; il n'existe pas toujours un unificateur plus général unique (ils peuvent même être en nombre infini, parfois finiment représentables par des techniques *ad hoc*).

De nombreux auteurs ont proposé d'encoder une partie des axiomes dans l'algorithme d'unification. Cela n'est possible qui si l'unification est décidable pour la théorie considérée. C'est le cas par exemple des théories de sortes ou encore des termes récurrents qui sont des langages permettant de définir des schémas itérées de termes tels que $f^n(0)$ (où n est une variable dans $\mathbb{N}$ et fait partie intégrante du langage). Ces approches, en étendant le langage de la L1O, permettent parfois d'obtenir des ensembles saturés finis. Des ensembles infinis de clauses peuvent être dénotés de manière finie, par exemple les clauses $\{pair(0), \forall x\,(\neg pair(x) \vee pair(s(s(x))))\}$ peuvent être remplacées par la clause $\forall n\,(pair(s^{2n}(0)))$.

3.3.9 La construction de modèle

La construction de modèle est le problème dual de la recherche d'une réfutation. L'objectif est, étant donnée une formule φ de construire une interprétation satisfaisant φ (donc un contre-modèle de $\neg\varphi$). Cela permet de prouver, par exemple, qu'une théorie est cohérente, mais aussi de détecter ou de corriger des erreurs dans des spécifications logiques.

Diverses techniques ont été développées pour construire automatiquement des modèles de formules logiques. Nous distinguons deux types d'approches.

Méthodes par énumération. L'existence d'un modèle fini est semi-décidable. Si la taille du modèle est finie, les quantifications peuvent être remplacées par des conjonctions ou disjonctions sur l'ensemble des éléments du domaine, ce qui permet de se ramener à un problème propositionnel qui peut alors être traité par un « SAT solver » classique. L'une des premières réalisations concrètes fut le logiciel Finder qui a notamment été utilisé pour restreindre l'espace de recherche de la résolution.

Algorithme 3.2 : recherche de modèle fini par énumération

Notations : L'ensemble des *cellules* sur le domaine D (noté $\mathcal{C}_D$) est l'ensemble des expressions de la forme $f(e_1, \ldots, e_k)$ où f est un symbole d'arité k et $(e_1, \ldots, e_k) \in D^k$. Une *interprétation partielle* sur D est une fonction partielle de $\mathcal{C}_D$ dans D. Une interprétation partielle associe une valeur à certains termes ou formules (les autres ayant une valeur indéfinie).

$FMod(S, n, D, \mathcal{I}) =$
Entrées : un ensemble de clauses S, un entier n et une interprétation partielle $\mathcal{I}$ sur le domaine D avec $|D| \leq n$
Sorties : soit une extension de $\mathcal{I}$ satisfaisant S, soit F
$\mathcal{I} \leftarrow propagation(S, n, D, \mathcal{I})$ /* Propagation des valeurs */
si $\mathcal{I} =$ F *ou* $\mathcal{I}$ *est totale* **alors retourner** $\mathcal{I}$

sinon
  Soit une cellule $c \in \mathcal{C}_D$ non définie dans $\mathcal{I}$
  pour tous les $v \in D$ **faire**
    $\mathcal{J} \leftarrow FMod(S, n, D, \mathcal{I} \cup \{c \mapsto v\})$
    si $\mathcal{J} \neq$ F **alors retourner** $\mathcal{J}$
  si $|D| = n$ **alors retourner** F

  sinon
    /* On ajoute un nouvel élément dans le domaine */
    Soit un nouvel élément $a \notin D$
    retourner $FMod(S, n, D \cup \{a\}, \mathcal{I} \cup \{c \mapsto a\})$

$propagation(S, n, D, \mathcal{I}) =$
Entrées : un ensemble de clauses S, un entier n et une interprétation partielle $\mathcal{I}$ sur D avec $|D| \leq n$
Sorties : soit une extension de $\mathcal{I}$, soit F
si *il existe* $C \in S$ *et une valuation* $\sigma : \mathcal{V} \to D$ *telles que* $[C]_{(\mathcal{I},\sigma)} =$ F **alors**
  /* Détection de contradiction */
  retourner F
sinon si *il existe* $C \vee f(t_1, \ldots, t_n) \simeq s$ *dans* S *et* $\sigma : \mathcal{V} \to D$ *telles que* $[C]_{(\mathcal{I},\sigma)} =$ F *et* $t_1, \ldots, t_n, s$ *ont une valeur dans* $(\mathcal{I}, \sigma)$ *mais pas* $f([t_1]_{(\mathcal{I},\sigma)}, \ldots, [t_n]_{(\mathcal{I},\sigma)})$ **alors**
  retourner $propagation(S, n, D, \mathcal{I} \cup \{f([t_1]_{(\mathcal{I},\sigma)}, \ldots, [t_n]_{(\mathcal{I},\sigma)}) \mapsto [s]_{(\mathcal{I},\sigma)}\})$
sinon retourner $\mathcal{I}$

Cette technique est notamment utilisée par le logiciel MACE2. Des raffinements ont

été proposés pour limiter le nombre de clauses propositionnelles engendrées qui augmente très rapidement avec la cardinalité du domaine. Une idée alternative consiste à traduire le problème en un ensemble de clauses du premier ordre sans symbole de fonction (classe Bernays-Schoenfinkel) pour lesquelles des procédures de décision existent.

L'algorithme 3.2 effectue une énumération directe évitant toute traduction. Pour simplifier, nous supposons que le seul prédicat est l'égalité. Il est utilisé par des systèmes tels que SEM. Des constructeurs de modèles finis ont été utilisés pour résoudre certaines conjectures en mathématiques.

Construction de modèle et déduction. Des méthodes de construction de modèle fondées sur des raffinements de la méthode de superposition ont été proposées. Si un ensemble saturé par superposition ne contient pas la clause vide, alors il est nécessairement satisfaisable. Cependant le modèle ne peut pas en général être construit explicitement car son domaine est infini (termes fermés). Pour certaines sous-classes, il est possible d'en construire une représentation finie, qui doit permettre au moins d'évaluer des atomes fermés dans le modèle (et dans certains cas d'évaluer également les clauses ou formules). De tels algorithmes ont été proposés pour diverses stratégies de recherche de preuve, notamment pour les stratégies positives. Un calcul dédié à la recherche simultanée de réfutation et de modèle a également été proposé, il est fondé sur l'utilisation de contraintes équationnelles [Caferra *et al.*, 2004].

3.4 La méthode des tableaux sémantiques

La méthode des tableaux sémantiques est l'une des méthodes les plus anciennes en DA ; elle a été formulée dans les années 50 par Beth [1955] et Hintikka [1955], développée ensuite par Smullyan [1963 ; 1995] et Fitting [1996] dans les années 60-70. L'idée de base de la méthode est d'essayer de construire un modèle d'un ensemble de formules en les décomposant selon leur structure syntaxique. Ces méthodes de preuve n'utilisant que les sous-formules du théorème à prouver sont dites *analytiques*. Si toutes les tentatives de construire un modèle échouent la formule est insatisfaisable, sinon chaque tentative « non échouée » est un modèle de la formule.

La méthode des tableaux est basée directement sur la sémantique et ne demande aucune transformation sous forme normale. Pour cette raison elle peut être facilement adaptée aux extensions/alternatives de la logique classique, pour lesquelles on peut difficilement obtenir des formes normales intéressantes. Enfin, les calculs des tableaux ont aussi un intérêt théorique puisqu'ils sont strictement liés aux calculs des séquents de Gentzen [Gentzen, 1935 ; Avron, 1993] qui constituent les fondements de la théorie de la preuve.

3.4.1 Tableaux propositionnels

Un *tableau* $\mathcal{T}$ est un arbre, dont les nœuds sont étiquetés par des formules. L'arbre initial comporte un unique nœud étiqueté par la formule (ou les formules) dont on veut tester la satisfaisabilité ; des règles permettent ensuite d'étendre cet arbre initial. En suivant une notation introduite par Smullyan [1963] et devenue standard, nous

α	α_1	α_2
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \Rightarrow \psi)$	φ	$\neg\psi$
$\neg\neg\varphi$	φ	

β	β_1	β_2
$\varphi \vee \psi$	φ	ψ
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi \Rightarrow \psi$	$\neg\varphi$	ψ

(a) Formules α et β

$$\frac{\alpha}{\alpha_1 \\ \alpha_2}\ (r\alpha) \qquad \frac{\beta}{\beta_1 \mid \beta_2}\ (r\beta)$$

$$\frac{p \quad \neg p}{\times}\ (r\times)$$

(b) Règles pour les tableaux propositionnels

FIGURE 3 – Règles pour les tableaux propositionnels

regroupons ces règles selon le type d'extension : prolongation (formules α) ou branchement (formules β). La figure 3(a) contient la répartition des formules selon les deux types et la figure 3(b) contient les règles d'expansions. Pour simplifier l'exposition nous identifierons souvent une formule avec son type.

Définition 55. Un tableau pour un ensemble de formules S est défini inductivement comme suit. Base : un arbre $\mathcal{T}$ constitué par un seul nœud étiqueté avec $\varphi \in S$ est un tableau pour S. Étapes inductives : soit $\mathcal{T}$ un tableau pour S et $\mathcal{B}$ une branche de $\mathcal{T}$, alors

(i) si $\varphi \in S$, l'arbre $\mathcal{T}'$ obtenu en prolongeant $\mathcal{B}$ avec un nœud étiqueté par φ est un tableau pour S,

(ii) si $\alpha \in \mathcal{B}$ l'arbre $\mathcal{T}'$ obtenu en prolongeant $\mathcal{B}$ avec un nœud n étiqueté par α_1 et (si nécessaire) un successeur direct n' de n étiqueté par α_2 est un tableau pour S,

(iii) si $\beta \in \mathcal{B}$ l'arbre $\mathcal{T}'$ obtenu en prolongeant $\mathcal{B}$ avec deux nœuds n_1 et n_2 étiquetés respectivement par β_1 et β_2 est un tableau pour S.

Si $S = \{\varphi\}$, on écrira que $\mathcal{T}$ est un tableau pour φ. La règle $(r\times)$ s'applique uniquement à des littéraux p et $\neg p$ d'une même branche $\mathcal{B}$.

Définition 56. Étant donné un tableau $\mathcal{T}$, une branche $\mathcal{B}$ est *fermée* si la règle $(r\times)$ est applicable à $\mathcal{B}$, sinon elle est *ouverte*. Un tableau est *fermé* si toutes ses branches sont fermées et *ouvert* dans le cas contraire. Une branche $\mathcal{B}$ ouverte est *saturée* si, pour chaque formule φ dans la branche, elle contient au moins une des conséquences de la règle (unique) qui peut être appliquée à φ.

Le tableau fermé de la figure 4 montre l'échec des tentatives de construction de modèles pour la négation de la formule $(p \vee q) \Rightarrow ((p \Rightarrow q) \Rightarrow q)$.

La méthode est *correcte* et *complète* : *une formule propositionnelle φ est valide si et seulement s'il existe un tableau fermé pour $\neg\varphi$.*

On peut observer que la construction d'un tableau initialisé avec une formule (ou un ensemble fini de formules) se termine toujours en produisant soit un tableau *fermé*, soit un tableau contenant une branche *saturée* et *ouverte*. Pour cette raison, la correction et la complétude sont prouvées normalement dans la forme contraposée : la formule $\neg\varphi$ est satisfaisable si et seulement s'il y a un tableau $\mathcal{T}$ pour $\neg\varphi$ contenant une branche

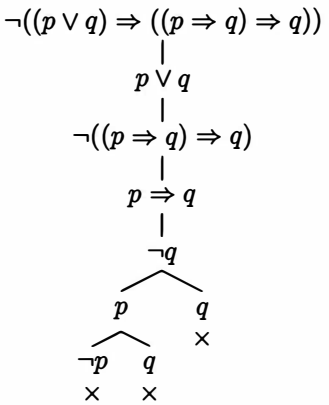


FIGURE 4 – Tableau prouvant la validité de $(p \vee q) \Rightarrow ((p \Rightarrow q) \Rightarrow q)$

saturée et *ouverte.*

Pour la correction il s'agit de montrer que les règles préservent la satisfaisabilité des formules d'une branche : si l'ensemble des formules présentes sur une branche est satisfaisable alors un des ensembles obtenus en étendant la branche par le conséquent d'une règle applicable est encore satisfaisable. Donc, elle ne peut pas être fermée, car une branche fermée n'est pas satisfaisable. On peut conclure que le tableau final contiendra une branche ouverte (et saturée).

Pour la complétude on montre que si un tableau contient une branche saturée et ouverte $\mathcal{B}$, alors il y a une interprétation $\mathcal{I}$ qui satisfait toutes les formules dans $\mathcal{B}$ (et en particulier la formule à la racine). Pour prouver cela, nous introduisons la notion suivante : un *ensemble d'Hintikka* H est un ensemble de formules satisfaisant les conditions suivantes : (i) il n'y a pas de symbole propositionnel p tel que $p \in H$ et $\neg p \in H$, (ii) si $\neg\neg\varphi \in H$ alors $\varphi \in H$ (iii) si $\alpha \in H$ alors $\alpha_1 \in H$ et $\alpha_2 \in S$, (iv) si $\beta \in H$ alors $\beta_1 \in H$ ou $\beta_2 \in H$.

Un ensemble d'Hintikka H fournit une interprétation $\mathcal{I}_H$ qui satisfait toutes les formules lui appartenant, cette interprétation est définie comme suit : $[p]_{(\mathcal{I}_H)} = \text{V}$ si $p \in H$ et $[p]_{(\mathcal{I}_H)} = \text{F}$ sinon. On peut alors prouver facilement que si $\varphi \in H$ alors $[\varphi]_{(\mathcal{I}_H)} = \text{V}$ et si $\neg\varphi \in H$ alors $[\varphi]_{(\mathcal{I}_H)} = \text{F}$. Pour terminer la preuve, on vérifie simplement que l'ensemble des formules présentes dans une branche saturée et ouverte est un ensemble d'Hintikka.

La formulation sous forme d'arbre n'est pas la seule possible. On peut donner un calcul équivalent dans lequel des règles sont locales et ne font plus référence aux branches : les branches sont remplacées par les ensembles des formules qu'elles contiennent. Ci-dessous nous donnons la formulation ensembliste des règles propositionnelles. Ici Γ dénote un ensemble arbitraire de formules et p un élément quelconque de $\mathcal{P}_0$.

$$\frac{\Gamma,\ \alpha}{\Gamma,\ \alpha_1,\ \alpha_2}\ (r\alpha) \qquad \frac{\Gamma,\ \beta}{\Gamma,\ \beta_1 \mid \Gamma,\ \beta_2}\ (r\beta) \qquad \frac{\Gamma,\ p,\ \neg p}{\times}\ (r\times) \tag{3.4}$$

Un tableau ensembliste pour Γ est un arbre avec Γ à la racine, construit en appliquant les règles et dans lesquels toutes le feuilles sont des applications de la règle $(r\times)$.

γ	$\gamma_1(t)$
$\forall x\,\varphi$	$\varphi[x/t]$
$\neg\exists x\,\varphi$	$\neg\varphi[x/t]$

(a) Formules γ : t est un terme fermé qui a une occurrence dans la branche

δ	$\delta_1(t)$
$\exists x\,\varphi$	$\varphi[x/c]$
$\neg\forall x\,\varphi$	$\neg\varphi[x/c]$

(b) Formules δ : c est une nouvelle constante qui n'a pas d'occurrence dans la branche

$$\frac{\gamma}{\gamma_1(t)}\ (r\gamma) \qquad \frac{\delta}{\delta_1(c)}\ (r\delta)$$

(c) Règles γ et δ

FIGURE 5 – Tableaux pour L1O

Cette formulation est très proche de celle des *calculs des séquents* sans coupures (ou calcul à la Gentzen). Ce sont les principaux systèmes de la théorie de la preuve, en fait il existe une correspondance précise entre les tableaux ensemblistes et le calcul des séquents [Avron, 1993].

3.4.2 Tableaux pour L1O

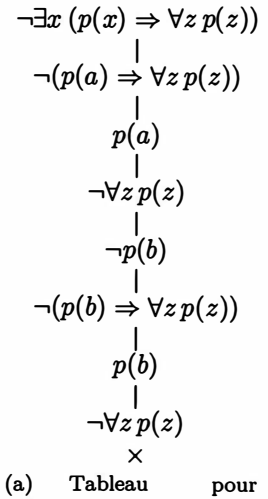
Pour étendre la méthode des tableaux à la logique du premier ordre nous ajoutons les règles pour les quantificateurs (figure 5). Reprenant la notation de Smullyan, nous avons la règle γ et la règle δ. La première, universelle, permet de prolonger la branche avec toutes les instances de la formule par un terme t fermé. La deuxième, existentielle, permet d'instancier la formule avec une *nouvelle constante.*

Deux observations sur la règle γ : si une branche ne contient pas de constante, la règle elle-même va en introduire une. Plus important, la règle γ peut et doit être appliquée *plusieurs fois* à la même formule avec des termes fermés différents. L'exemple de la figure 6 (a) montre ces deux caractéristiques. Le tableau de la figure 6(b) fournit un contre-exemple à la validité de la formule $\forall x\,(p(x) \vee q(x)) \Rightarrow (\forall x\,p(x) \vee \forall x\,q(x))$. On peut observer que la branche ouverte (et saturée) fournit un contre-modèle $\mathcal{I}$ de cette formule sur le domaine $\{a, b\}$ défini par $p^{\mathcal{I}} = \{b\}$, $q^{\mathcal{I}} = \{a\}$.

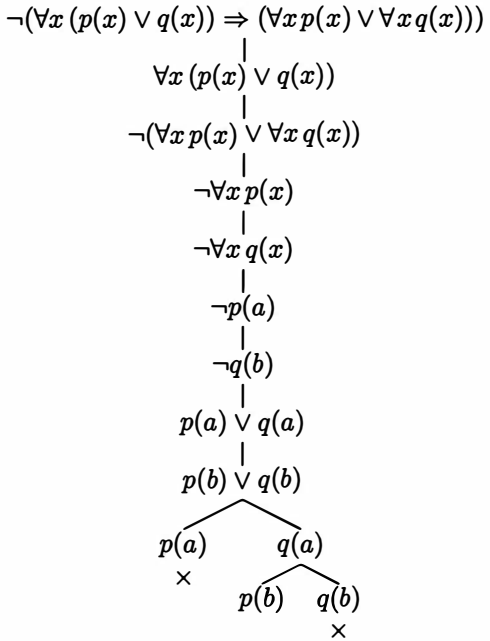
Si pour la logique propositionnelle la construction d'un tableau se termine toujours après un *nombre fini d'étapes* avec toutes les branches saturées ou fermées, ceci n'est évidemment pas le cas pour la logique du premier ordre, qui n'est pas décidable. Pour s'en rendre compte, il suffit de développer un tableau pour la formule $\forall x\,\exists y\,p(x, y)$. Du moment que l'ensemble des termes fermés peut être infini, une branche saturée (et ouverte) sera forcément infinie. Cependant, on peut montrer que la méthode est correcte et complète : *une formule φ de L1O est valide si et seulement s'il existe un tableau fermé pour $\neg\varphi$.*

Pour la correction, comme dans la logique propositionnelle, il s'agit de montrer que les règles préservent la satisfaisabilité.

Pour la complétude, la question est plus compliquée : il faut construire une interprétation satisfaisant toutes les formules d'une branche *saturée et ouverte* et donc la formule initiale. Nous étendons la notion d'ensemble d'Hintikka par rapport à un ensemble Σ de termes fermés ; cet ensemble contient les termes construits avec les constantes introduites par les applications des règles δ (en plus de celles déjà présentes). Nous disons qu'un ensemble de formules H est un ensemble d'Hintikka par



(a) Tableau pour $\neg\exists x\,(p(x) \Rightarrow \forall z\,p(z))$



(b) Tableau pour $\neg(\forall x\,(p(x) \vee q(x)) \Rightarrow (\forall x\,p(x) \vee \forall x\,q(x)))$.

FIGURE 6 – Exemples de tableaux pour L1O

rapport à Σ s'il satisfait les conditions (i)-(iv) indiquées dans la section 3.4.1 et les conditions additionnelles suivantes : (v) si $\gamma \in H$ et $t \in \Sigma$, alors $\gamma_1(t) \in H$; (vi) si $\delta \in H$ alors il existe un terme $t \in \Sigma$ tel que $\delta_1(t) \in H$. Pour démontrer qu'un ensemble de Hintikka H est satisfaisable, on définit un modèle de Herbrand $\mathcal{I}$ qui va satisfaire toutes les formules de H : $p^{\mathcal{I}} = \{\langle t_1, \ldots, t_n\rangle \mid p(t_1, \ldots, t_n) \in H\}$ pour tout $n \in \mathbb{N}$ et tout symbole de prédicat p d'arité n. On démontre que si $\varphi \in H$ alors $\mathcal{I} \models \varphi$. Il est facile de voir que les formules d'une branche $\mathcal{B}$, saturée et ouverte, forment un ensemble d'Hintikka par rapport à l'ensemble des termes fermés de $\mathcal{B}$.

On obtient donc que $\neg\varphi$ est satisfaisable s'il y a un tableau pour $\neg\varphi$ contenant au moins une branche *saturée et ouverte*.

Pour conclure la preuve de complétude il faut donner une stratégie (ou un algorithme) qui garantit la génération d'une branche (infinie) saturée et ouverte si le tableau n'est fermé à aucune étape finie. Il faut donc une *stratégie systématique* qui utilise un critère *équitable* d'application des règles : aucune formule ne sera « infiniment négligée ». Une telle stratégie systématique fonctionne car les règles sont *non destructives* : toutes applications des règles à un tableau $\mathcal{T}$ produisent un tableau qui contient $\mathcal{T}$ comme sous-arbre. Cela implique que l'application d'une règle ne peut jamais rendre impossible la construction d'un tableau fermé s'il en existe un. En conséquence, ce calcul a la propriété dite de *confluence* : tout tableau pour un ensemble insatisfaisable de formules peut être étendu à un tableau fermé.

3.4.3 Tableaux avec variables libres

L'implantation de cette idée comporte un certain nombre des modifications aux règles et à la construction du tableau afin de préserver la correction et la complétude de la méthode : la règle γ remplace la variable quantifiée avec une nouvelle *variable libre* interprétée comme *rigide* (elle dénote le même élément dans toutes les branches). Ces variables libres sont éventuellement instanciées lors de *l'unification* (cf. section 3.3.2) de deux littéraux complémentaires dans une branche [4], ce qui entraîne la fermeture de la branche. Du moment que les variables libres sont *rigides*, l'unificateur le plus général qui ferme une branche doit être appliqué globalement à *tout* le tableau.

Définition 57. Un tableau avec variables libres pour un ensemble des formules S est un arbre généré (dans le sens de la définition 55) par les règles α et β et par les règles suivantes :

1. $\dfrac{\gamma}{\gamma_1(x')}\ (r\gamma')$ où x' est une variable qui n'apparaît pas dans le tableau. On appelle $\gamma_1(x')$ une nouvelle instance de γ.
2. $\dfrac{\delta}{\delta_1(f(x_1,\ldots,x_n))}\ (r\delta^+)$
3. La règle (*Subst*) : si σ est un *unificateur le plus général* (upg) de deux littéraux complémentaires L et L' dans une branche $\mathcal{B}$ d'un tableau $\mathcal{T}$, on applique σ à $\mathcal{T}$ (noté $\mathcal{T}\sigma$) et on ajoute $\times$ à la branche.

La figure 7 montre un tableau avec variables libres fermé pour le même exemple de la figure 6.

$$\begin{array}{c}
\neg\exists x\,(p(x) \Rightarrow \forall z\, p(z)) \\
| \\
\neg(p(x') \Rightarrow \forall z\, p(z)) \\
| \\
p(x') \\
| \\
\neg\forall z\, p(z) \\
| \\
\neg p(b) \\
| \\
\times\ \{x' \mapsto b\}
\end{array}$$

FIGURE 7 – Tableau avec variables libres pour $\neg\exists x\,(p(x) \Rightarrow \forall z\, p(z))$

On peut prouver que la méthode est correcte. Cependant, donner un algorithme qui garantit la génération d'un tableau fermé (s'il existe) est plus difficile que pour la méthode de base. La stratégie systématique ne fonctionne plus car la règle de fermeture (Subst) est *destructive* : en conséquence une mauvaise application de cette règle peut empêcher la construction d'un tableau fermé, même s'il en existe un. Considérons l'exemple (trivial) d'un tableau pour $\{\forall x\,(p(x) \vee q(x)), \neg p(a), \neg p(b), \neg q(b)\}$, si l'on fait le mauvais choix unifiant $p(x')$ avec $p(a)$, on ferme une des deux branches, mais on ne peut plus obtenir un tableau fermé, bien qu'il en existe un.

4. On peut montrer que la règle de fermeture peut être limitée aux littéraux.

En d'autres termes, le calcul présenté avec variables libres et la règle (Subst) n'a plus la propriété de *confluence*. Pour cette raison les algorithmes pour les tableaux avec variables libres sont basés soit sur l'idée de construire tous les tableaux possibles en parallèle (par une exploration de l'arbre des tableaux possibles par « iterative deepening » [Stickel, 1992]), soit sur l'idée de différer l'application de la règle (Subst) jusqu'à ce qu'on trouve un unificateur le plus général en mesure de fermer d'un *seul coup tout* le tableau [Fitting, 1996].

On peut formuler des calculs plus efficaces que les tableaux avec variables libres si l'on suppose que les formules ont été transformées en forme clausale de Skolem (voir section 3.3.1). Pour ce type de formules la méthode se réduit en fait à une règle d'expansion qui comporte à la fois le branchement n-aire (β) et l'instanciation par une nouvelle variable libre (règle γ) et la règle de fermeture (Subst). Cette variante est connue sous le nom de *tableaux de clauses*. Un raffinement ultérieur (*tableaux avec connexions*) consiste à imposer qu'un des littéraux introduits par la règle d'expansion doit être unifiable avec le complémentaire du littéral qui est l'étiquette de son prédécesseur.

3.5 Les logiques non classiques

Dans cette partie, on se réfère au chapitre I.2 qui introduit les logiques modales et la sémantique de Kripke. Un *langage modal* est construit sur un ensemble $\mathcal{P}_0$.

Les méthodes les plus répandues de démonstration automatique sont des méthodes de résolution, qui nécessitent une forme clausale des formules. Cela ne pose pas de problèmes pour la logique classique (propositionnelle ou du premier ordre), mais en pose pour les logiques modales et conditionnelles qui, en général, n'admettent pas de forme clausale.

A partir de ce constat, deux approches pour la démonstration automatique en logique modale sont envisageables : soit on utilise une méthode qui ne nécessite pas de forme clausale des formules à prouver, soit on traduit les formules modales en L1O en introduisant des variables qui nomment les mondes et on utilise ensuite un démonstrateur classique. Mais la L1O n'est pas décidable alors que beaucoup de logiques modales le sont, la décidabilité peut donc être perdue par traduction. D'autre part, il y a des systèmes modaux dont la caractérisation n'est pas exprimable en L1O mais seulement en logique d'ordre supérieur. Par exemple, le système (S.4.3.1) est défini par les axiomes **K**, **T**, **4** et « Dum », ce dernier exprime l'inexistence de « clusters » non finaux. Les méthodes de traduction en L1O ne peuvent donc pas s'appliquer à ces logiques.

Les méthodes basées sur les tableaux sémantiques, qui ne nécessitent pas la mise sous forme clausale des formules à prouver, relèvent de la première approche. Elles sont devenues les méthodes préférées pour la démonstration automatique en logique modale. Il y a deux types de calculs basés sur les tableaux sémantiques : les calculs de tableaux *implicites* et les calculs de tableaux *explicites*.

Les règles pour le calcul de tableaux implicites codent le passage d'un monde à l'autre implicitement dans les règles de tableaux [Goré, 1999]. Le calcul de tableaux explicites [Viganò, 2000] utilise des formules précédées par un nom de monde (une *étiquette*). On démontre donc qu'une formule est satisfaisable (ou non) dans un monde x, dont le nom précède la formule. Ceci permet de formuler directement et très facilement

les règles de tableaux qui représentent cette relation.

Il y a un grand nombre de logiques modales définies par des axiomes. Chaque axiome est caractérisé sémantiquement par une propriété de la relation de transition R. Par exemple, pour mentionner les plus connus, pour l'axiome **T** la relation R est réflexive, pour l'axiome **4** elle est transitive et pour l'axiome **B** elle est symétrique. Le calcul de tableaux explicites peut directement exprimer ces propriétés pour chaque axiome. Les méthodes implicites les codent indirectement. Les propriétés principales des tableaux modaux sont définies de façon analogue aux tableaux classiques : branches et tableaux fermés, saturés.

3.5.1 Calcul de tableaux implicites

Nous donnons ici d'abord un calcul implicite ensembliste pour les logiques modales **K**, **T** et **S4**. Chaque calcul de tableaux contient les règles pour la logique propositionnelle et une ou plusieurs règles de la figure 8.

$$\frac{\Box\Gamma,\ \neg\Box\varphi}{\Gamma,\ \neg\varphi}\ (K) \qquad \frac{\Gamma,\ \Box\varphi}{\Gamma,\ \Box\varphi,\ \varphi}\ (T) \qquad \frac{\Box\Gamma, \neg\Box\varphi}{\Box\Gamma,\ \neg\varphi}\ (S4)$$

FIGURE 8 – Règles de tableaux implicites pour **K**, **T**, **S4**

Γ représente un ensemble fini de formules, $\Box\Gamma = \{\Box\psi : \psi \in \Gamma\}$ et φ représente une formule. Chaque règle utilise implicitement la sémantique de l'opérateur modal. Si, dans un ensemble Γ de formules, une formule modale $\Diamond\varphi$ est vraie, alors il y a un monde possible dans lequel φ est vraie et aussi toutes les formules ψ telles que $\Box\psi$ était vraie. Ce monde possible est construit par les formules du dénominateur de la règle (K). La règle (T) prend en compte que chaque monde est un monde possible pour lui-même (réflexivité). Elle ajoute donc simplement la formule φ à un monde qui contient $\Box\varphi$. La règle (S4) « code » la transitivité de la relation de transition : si un nouveau monde possible est introduit il contient toutes les formules de la forme $\Box\psi$, qui étaient contenues dans le monde « actuel ». Un tableau pour un ensemble de formules est construit comme décrit en section 3.4 avec les règles ensemblistes (3.4). On appelle $\mathcal{C}$**X**-tableau un tableau pour la logique X. La figure 9 montre un $\mathcal{C}$**K**-tableau pour la formule $\neg(\Box(p \Rightarrow q) \wedge \Diamond p \Rightarrow \Diamond q)$.

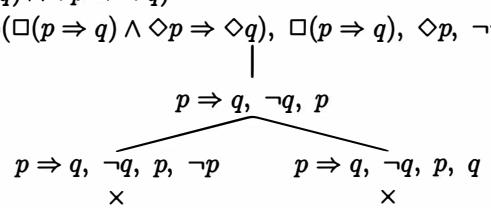


FIGURE 9 – $\mathcal{C}$**K**-Tableau pour $\neg(\Box(p \Rightarrow q) \wedge \Diamond p \Rightarrow \Diamond q)$

Un ensemble de formules Γ est appelé $\mathcal{C}$**K**-inconsistant (resp. $\mathcal{C}$**T**-inconsistant, resp.

$$\frac{x : \Box\varphi,\ x\ R\ y}{y : \varphi}\ (T\Box) \qquad \frac{x : \neg\Box\varphi}{x\ R\ y,\ y : \neg\varphi}\ (F\Box)(*)$$

(*) y est une étiquette nouvelle dans la branche.

FIGURE 10 – Règles de tableaux explicites pour **K**

$\mathcal{CS4}$-inconsistant) si chaque $\mathcal{CK}$-tableau (resp. $\mathcal{CT}$-tableau, resp. $\mathcal{CS4}$-tableau) pour Γ est fermé. Donc, il est consistant ssi chaque tableaux pour Γ est ouvert. A partir d'un tableau ouvert pour un ensemble Γ on peut construire un modèle pour Γ. Une formule φ est donc un théorème de **K** (resp. **T**, resp. **S4**) ssi il y a un $\mathcal{CK}$-tableau (resp. $\mathcal{CT}$-tableau, resp. $\mathcal{CS4}$-tableau) fermé pour $\neg\varphi$.

3.5.2 Calcul de tableaux explicites

Nous introduisons ici des calculs de tableaux où la relation d'accessibilité est représentée explicitement. On peut « dériver » une règle de tableau pour un schéma de formule à partir de la sémantique de la logique sous-jacente. Considérons la formule modale $\Box\varphi$. Étant donnés une interprétation modale $M = (S, R, v)$ et $x \in S$ on a $M, x \models \Box\varphi$ ssi $\forall y \in S$ si $x\ R\ y$ alors $M, y \models \varphi$. On peut donc déduire de $M, x \models \Box\varphi$ et $x\ R\ y$ que $M, y \models \varphi$; c'est la règle $(T\Box)$ de la figure 10 qui contient les règles de tableaux pour la logique modale de base **K**. Leurs prémisses sont des *formules étiquetées* $x : \varphi$ et des *formules de transition* $x\ R\ y$.

La figure 11 montre le tableau étiqueté pour la formule $\neg(\Box(p \Rightarrow q) \wedge \Diamond p \Rightarrow \Diamond q)$:

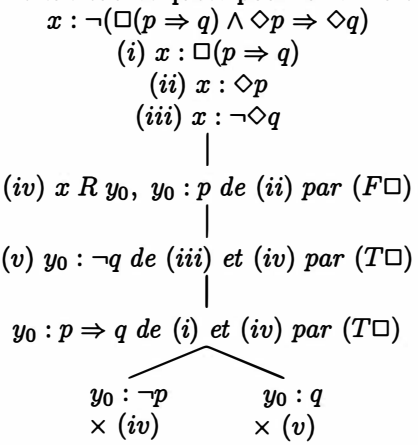


FIGURE 11 – Tableau explicite pour $\neg(\Box(p \Rightarrow q) \wedge \Diamond p \Rightarrow \Diamond q)$

La notion de fermeture est définie comme pour la logique classique en prenant en compte les formules étiquetées, ainsi que la notion de saturation d'une branche $\mathcal{B}$ (avec

l'ensemble des étiquettes $\mathcal{B}_E$) en ajoutant les règles qui correspondent aux règles de tableaux modaux : si $x : \Box\varphi \in \mathcal{B}$ et $x\ R\ y \in \mathcal{B}$ alors $y : \varphi \in \mathcal{B}$; et si $x : \neg\varphi \in \mathcal{B}$ alors il y a $y \in \mathcal{B}_E$ et $y : \neg\varphi \in \mathcal{B}$ et $x\ R\ y \in \mathcal{B}$.

Pour démontrer la correction et la complétude des règles de tableaux, on associe une interprétation à une branche ouverte et saturée. Étant donnée une branche $\mathcal{B}$ on définit l'ensemble des étiquettes de $\mathcal{B}$ par $\mathcal{B}_E = \{x \,|\, x : \varphi \in \mathcal{B}\}$ [5] et $\mathcal{P}(\mathcal{B}) = \{p \in \mathcal{P}_0 \,|\, x : p \in B\}$ est l'ensemble des formules atomiques qui ont une occurrence dans $\mathcal{B}$.

On dit qu'une interprétation $M = (S, R, v)$ est *associée* à une branche $\mathcal{B}$ *sous* la fonction $f : \mathcal{B}_E \longrightarrow S$ si pour toute formule de transition $x\ R\ y$ dans $\mathcal{B}$ on a $f(x)\ R\ f(y)$. Une branche $\mathcal{B}$ est *satisfaite par une interprétation* $M = (S, R, v)$ *sous la fonction* f ssi M est associée à $\mathcal{B}$ sous f et si pour toute formule étiquetée $x : \varphi$ dans $\mathcal{B}$ on a $M, f(x) \models \varphi$. Un tableau est *satisfaisable* s'il admet une branche satisfaite par une interprétation sous une fonction.

On montre facilement que le calcul est correct :

Lemme 1. Soit $\mathcal{T}$ un tableau et $\mathcal{T}'$ le tableau obtenu à partir de $\mathcal{T}$ en appliquant une règle de tableau. Alors $\mathcal{T}'$ est satisfaisable si $\mathcal{T}$ est satisfaisable.

Pour la complétude, on construit une interprétation canonique $MC = (S, R, v)$ à partir d'une branche ouverte et saturée $\mathcal{B}$, avec $S = \mathcal{B}_E$, $R = \{\langle s, s'\rangle \mid s\ R\ s' \in \mathcal{B}\}$ et $v(s, p) = \mathrm{V}$ si $s : p \in \mathcal{B}$ pour tout $p \in \mathcal{P}(\mathcal{B})$. On montre alors que MC satisfait toutes les formules de $\mathcal{B}$.

Théorème 36. Si une formule φ est valide alors tout tableau pour $x : \neg\varphi$ est fermé.

Les extensions « standard » de la logique modale de base **K** sont caractérisées par des axiomes additionnels et au niveau sémantique par une propriété de la relation de transition. Nous donnons ici les règles de tableaux pour les axiomes **T**, **S4** et **S5**, qui s'expliquent directement à partir de la relation sémantique (table 1).

En présence des règles S4 et S5, on n'est pas assuré de la terminaison du calcul. Par exemple, le tableau pour la formule $\neg(\Box\Diamond p \Rightarrow \Diamond\Box p)$ génère une séquence infinie de mondes possibles. Mais ces boucles peuvent être détectées. Elles sont identifiables parce que chaque étiquette nouvellement générée va précéder le même ensemble de formules. On introduit donc des tests de boucles dans le calcul afin d'obtenir un calcul qui termine [Heuerding *et al.*, 1996]. On peut aussi développer des stratégies sophistiquées avec tests de terminaison [Massacci, 2000].

L'idée d'utiliser des étiquettes se trouve déjà dans les travaux de Kripke, qui donne un calcul de tableaux pour logiques modales, Fitting [Kripke, 1963 ; Fitting, 1983]. Schütte dans [1978] introduit des arbres étiquetés où les étiquettes sont des séquences finies d'entiers. Ce codage des étiquettes a aussi été proposé par Massacci [2000], qui donne un calcul de tableaux systématique et modulaire pour un grand nombre de systèmes connus. Dans ce calcul (Schütte, Massacci), les fils d'un nœud étiqueté σ sont étiquetés $\sigma.1, \sigma.2, \ldots$, ce qui permet de « déduire » directement la relation de transition entre deux nœuds en prenant éventuellement en compte la transitivité, la symétrie, etc.

5. Si $x\ R\ y \in \mathcal{B}$, on a aussi $x \in \mathcal{B}_E$ et $y \in \mathcal{B}_E$, selon les règles.

Axiome	Propriété de R	Règle de tableau
T $\Box\varphi \Rightarrow \varphi$	réflexive	$\dfrac{x : \varphi}{x\ R\ x,}$
S4 $\Box\varphi \Rightarrow \Box\Box\varphi$	transitive	$\dfrac{x\ R\ y, y\ R\ z}{x\ R\ z}$
S5 $\neg\Box\neg\Box\varphi \Rightarrow \varphi$	symétrique	$\dfrac{x\ R\ y}{y\ R\ x}$

TABLE 1 – Axiomes et règles de tableau pour **T**, **S4**, **S5**

3.5.3 Logiques conditionnelles

Les techniques de tableaux explicites ont été largement utilisés en démonstration automatique pour les logiques conditionnelles. Mais le problème de la définition et de la construction un démonstrateur pour ces logiques est d'une très grande complexité. Notamment, les logiques conditionnelles ne sont pas caractérisées par une sémantique standard comme les logiques modales. La sémantique la plus générale, celle des fonctions de sélection peut caractériser toutes les logiques conditionnelles. Des systèmes de démonstration basés sur le calcul des séquents ont été élaborés pour quelques variantes de ces logiques (système **CK, ID, CEM, CS**) [Olivetti *et al.*, 2007]. Pour d'autres systèmes (*AC*, *CV*, *MOD*, *CUT*, ...), caractérisés par la sémantique préférentielle, un système de tableaux a été présenté dans [Giordano *et al.*, 2009]. Dans ce travail, on utilise des techniques de calcul explicite pour décrire des relations préférentielles et pour représenter des éléments minimaux.

3.6 Gérer l'incomplétude

Les logiques utilisées dans les sections précédentes sont toutes semi-décidables, elles admettent donc des systèmes d'inférence complets qui sont un élément essentiel de leur mécanisation. On sait cependant qu'elles ne recouvrent qu'un ensemble limité du raisonnement mathématique, il leur manque l'ingrédient essentiel qu'est le raisonnement inductif. Or on sait depuis Gödel qu'une induction aussi simple que la récurrence sur les entiers rend l'arithmétique incomplète. La DA ne peut que pallier cet inconvénient par des méthodes très différentes des précédentes et que nous ne pouvons qu'évoquer brièvement.

3.6.1 L'induction

Tout raisonnement inductif s'appuie sur un *principe d'induction*, c'est-à-dire un théorème qui dit que, pour toute propriété P, si on a $\forall y\,(\chi(P,y) \Rightarrow P(y))$ alors on a $\forall y\,P(y)$. La formule $\chi(P,y)$ est l'*hypothèse d'induction* et y est la *variable d'induction*. Le raisonnement inductif consiste alors à déduire une formule φ (ayant y pour variable libre) à partir de l'hypothèse $\chi(\varphi,y)$, puis à en déduire que $\forall y\,\varphi$ est un théorème selon ce principe d'induction. On conçoit facilement qu'il est plus simple de prouver φ avec une hypothèse que sans ; c'est ce qui fait l'intérêt du raisonnement inductif.

Un principe d'induction dépend évidemment de l'ensemble E dans lequel y varie et surtout de la formule $\chi(\varphi,y)$. On peut distinguer deux formes générales du raisonnement inductif selon que la formule $\chi(\varphi,y)$ est universelle ou existentielle. La forme existentielle est l'*induction structurelle*, elle dépend d'un ensemble $\mathcal{C}$ dont les éléments sont appelés *constructeurs* ; un constructeur est une fonction de E^n vers E pour un $n \in \mathbb{N}$. L'hypothèse d'induction $\chi(\varphi,y)$ est alors

$$\exists n \in \mathbb{N},\ \exists f \in \mathcal{C},\ \exists x_1,\ldots,x_n \in E,\ y = f(x_1,\ldots,x_n) \wedge \varphi[y/x_1] \wedge \cdots \wedge \varphi[y/x_n].$$

Il est facile de montrer que $\forall y \in E,\ (\chi(\varphi,y) \Rightarrow \varphi)$ implique $\forall y \in E,\ \varphi$ si E est l'ensemble des éléments finiment engendrés par les constructeurs ; c'est le *principe d'induction structurelle*. Il est alors naturel de dénoter les éléments de E par des termes fermés, en prenant $\mathcal{F} = \mathcal{C}$ (avec les mêmes arités). Mais en L1O on ne peut en général exclure des modèles contenant des éléments non dénotés par des termes clos ; le raisonnement par induction structurelle y est donc incorrect puisqu'il assignerait à ces éléments des propriétés vraies uniquement sur E (donc vraies seulement dans les modèles de Herbrand construits avec $\mathcal{F}$). C'est pourquoi la L1O devient vite insuffisante lorsqu'il faut raisonner sur des objets mathématiques ou informatiques.

L'*induction bien fondée* utilise une relation binaire R sur E telle qu'il n'existe pas de suite infinie $(x_i)_{i\in\mathbb{N}}$ vérifiant $\forall i \in \mathbb{N},\ x_{i+1}\ R\ x_i$ (autrement dit, *tous* les chemins descendants par R sont finis). Elle autorise alors une hypothèse d'induction universelle :

$$\forall x \in E,\ x\ R\ y \Rightarrow \varphi[y/x].$$

L'induction bien fondée vient de la théorie des ordinaux où R est la relation d'appartenance $\in$ (on l'appelle parfois *induction transfinie*). Un exemple très connu d'induction bien fondée est la *récurrence forte* où $E = \mathbb{N}$ et R est l'ordre strict $<$ sur $\mathbb{N}$.

Il est facile de voir que la récurrence classique est à la fois une induction structurelle (avec l'atome 0 et la fonction successeur $\mathrm{s}(n) = n+1$ comme constructeur) et une induction bien fondée (avec $R = \mathrm{s}$) ; cela est dû au fait que pour tout n il y a un chemin unique par s reliant 0 à n. Le théorème d'incomplétude de Gödel montre donc que les deux principes d'induction sont sources d'incomplétude.

En théorie de la preuve cela se traduit par le fait que certaines formules φ ne peuvent être obtenues qu'en prouvant par induction un lemme qu'il faut découvrir (autrement dit, on ne peut pas éliminer les « coupures »). Par exemple, certains théorèmes doivent d'abord être généralisés pour qu'il soit possible de les prouver par induction. Pour les théorèmes contenant plusieurs variables universelles, il est également essentiel de choisir judicieusement la variable d'induction.

C'est pourquoi les systèmes de démonstration automatique par induction restent ouverts à l'intervention humaine. Le plus célèbre est probablement le démonstrateur de Boyer et Moore (Nqthm, puis ACL2) ; mais des systèmes comme RRL, INKA ou Oyster/Clam ont également servi de base à de nombreux travaux visant à proposer automatiquement des schémas d'induction, des généralisations inductives ou simplement la variable d'induction [Robinson et Voronkov, 2001, chapitre 13].

D'autres techniques, inspirées de la méthode de complétion de Knuth-Bendix en réécriture, permettent d'automatiser plus simplement l'induction dans des théories ou pour des formules particulières. Elles utilisent des ensembles de termes ayant certaines propriétés de complétude (les « cover sets » et les « test sets », comme dans les systèmes RRL et SPIKE). Dans certains cas on dispose d'axiomes qui permettent de prouver une conjecture *par consistance*, c'est-à-dire en montrant simplement que la conjecture ajoutée aux axiomes n'entraîne aucune contradiction. On peut alors utiliser un démonstrateur pour la L1O, mais fonctionnant par saturation afin d'établir la consistance, pour prouver des propriétés inductives (c'est-à-dire valides dans l'ensemble des modèles de Herbrand) ; c'est l'*induction sans induction*, voir [Robinson et Voronkov, 2001, chapitre 14].

3.6.2 Les cadres logiques ou assistants de preuve

On peut considérer que les ordinateurs sont dès leur origine des assistants mathématiciens : encore aujourd'hui le calcul numérique constitue probablement l'activité principale des ordinateurs dans le monde. Mais lorsqu'on envisage de résoudre des problèmes numériques difficiles le calcul doit se faire symbolique et sa mécanisation relève alors de l'IA.

Mais le calcul symbolique ne possède pas les propriétés algorithmiques du calcul purement numérique, et sa complexité augmente avec celle des problèmes qu'il permet de résoudre. Une approche pragmatique consiste donc à automatiser ce qui peut l'être, comme le calcul sur les polynômes et les fractions rationnelles. Les *systèmes de calcul formel* furent d'abord développés pour la physique, où un système comme Schoonschip contribua même à des travaux qui valurent un prix Nobel à leur auteur. Ils sont aujourd'hui très utilisés également en mathématiques.

Cependant, ces systèmes de plus en plus complexes présentent certains inconvénients : la correction des algorithmes utilisés est rarement prouvée, et certaines méthodes ne sont correctes que sous certaines hypothèses, mais sont appliquées sans les vérifier. Ces problèmes deviennent cruciaux lorsqu'on veut non seulement calculer mais garantir la correction des résultats, ou plus généralement lorsqu'il s'agit de prouver des propriétés. Pour cela il faut concevoir des systèmes suffisamment généraux pour pouvoir y développer l'essentiel des mathématiques ; ce sont les *assistants de preuve*, aussi appelés *cadres logiques* car ils reposent sur un langage permettant d'y exprimer de façon plus ou moins directe des mathématiques, ce qui inclut des définitions, des théorèmes et des preuves [Robinson et Voronkov, 2001, chapitre 17].

Ces systèmes offrent la possibilité de développer et de vérifier des preuves. La vérification peut n'utiliser qu'un noyau relativement simple qui n'est autre qu'une version mécanisée d'un système formel (axiomes et règles d'inférence). Un système vérifie le

critère de de Bruijn si toutes les preuves sont vérifiées par ce noyau. La correction des théorèmes (relativement au cadre logique) repose alors uniquement sur celle du noyau.

Cette réduction semble d'ailleurs être nécessaire aux mathématiques, dans la mesure où certaines preuves extrêmement longues et complexes empêchent leur relecture par un nombre suffisant de spécialistes compétents. C'est le cas par exemple de la démonstration par Wiles du théorème de Fermat, ou de la classification des groupes finis : la plupart des mathématiciens doivent se contenter de *croire* que ces résultats sont effectivement prouvés. Certains sont même tentés de refuser le statut de preuve à un texte trop long pour qu'un être humain puisse en vérifier toutes les étapes. C'est pourquoi la preuve du théorème des quatre couleurs, basée en partie sur une énumération automatisée d'une très longue liste de cas particuliers, est peu satisfaisante pour l'intuition.

Mais avant qu'un ordinateur soit capable de remédier à ces problèmes, il faut d'abord trouver un cadre logique suffisamment puissant pour y développer ces preuves et assez simple pour satisfaire l'intuition ; il faut ensuite fournir à un utilisateur les outils nécessaires pour y développer de plus longues preuves qu'il n'en pourrait lire.

Le premier problème a été essentiellement résolu par les recherches sur les fondations des mathématiques. On sait depuis Cantor que la théorie des ensembles, exprimée en logique du premier ordre, permet de formaliser les mathématiques, bien que parfois au prix de certaines lourdeurs. La théorie des types est un autre candidat, souvent plus proche de la pratique mathématique (et informatique). C'est ce dernier choix qui a été retenu dans le premier cadre logique, le système AUTOMATH dû à N. de Bruijn dès 1968. Mais malgré son nom, ce système n'offre pas de possibilités d'automatiser la construction d'une preuve.

Deux systèmes plus spécialisés pour la preuve de programme apparurent en 1971-72 : Nqthm (ou démonstrateur de Boyer et Moore, devenu ACL2) et LCF (de Milner). Le premier est assez largement automatisé, le second offre des possibilités d'automatisation au moyen des *tactiques* ; ce sont des programmes dont le but est d'appliquer des règles d'inférence « inversées », avec la conjecture courante comme conclusion afin d'obtenir les prémisses correspondantes, qui deviennent alors de nouvelles conjectures. En cas d'échec, un mécanisme de rattrapage d'exceptions permet de revenir à l'état initial.

L'intérêt des tactiques est surtout la possibilité de les composer entre elles, pour en obtenir de plus complexes, au moyen d'opérateurs appelés *tacticals*. On peut par exemple répéter jusqu'à échec une tactique sur toutes les conjectures qu'elle produit successivement. C'est d'ailleurs pour permettre à l'utilisateur d'écrire ses propres tactiques que Milner a conçu le langage ML (pour Méta-Langage), où le typage permet d'assurer que seules les formules prouvées ont le type `thm`.

Un autre assistant de preuve pionnier est le système Mizar, développé par Trybulek dès 1973. Il est basé sur une théorie des ensembles en logique classique, mais offre des possibilités de typages tels qu'employés en mathématiques. Le but de ce système est de vérifier automatiquement des textes (ou *articles*) rédigés dans un langage le plus proche possible d'un texte mathématique, et donc lisible par un humain. L'auteur d'un article est censé ajouter des détails intermédiaires (grâce à des indications provenant du système) jusqu'à ce que Mizar soit capable de le vérifier.

Les cadres logiques plus récents offrent souvent une possibilité que nous n'avons pas encore évoquée, mais qui semble irrésistible à un informaticien ; celle de développer formellement des *mathématiques constructives* (en abandonnant le principe du tiers exclu et donc la logique classique). En effet, on peut extraire d'une preuve constructive d'une proposition de la forme $\forall x_1 \ldots x_n \exists y\, p(x_1, \ldots, x_n, y)$ un programme fonctionnel f tel que

$$\forall x_1 \ldots x_n\, p(x_1, \ldots, x_n, f(x_1, \ldots, x_n)).$$

On peut alors exécuter ce programme sur des données particulières. Cela n'est pas seulement anecdotique, les mathématiques ont un aspect calculatoire depuis l'antiquité (on peut penser à l'algorithme d'Euclide pour calculer le pgcd), et il semble indispensable que la notion de calcul soit intégrée dans un système où des mathématiques sont développées. Les cadres logiques diffèrent selon qu'ils permettent d'effectuer des calculs numériques ou symboliques avec plus ou moins de contrôle ou de concision, selon la façon dont le raisonnement équationnel y est représenté.

Parmi les systèmes récents, le plus connu est PVS. Il offre à l'utilisateur de nombreuses fonctionnalités (procédures de décision, solveur SMT, etc.), en partie du fait qu'il ne satisfait pas au critère de de Bruijn. Dans le système Isabelle, ce critère est respecté mais les preuves formelles (trop longues en présence de calculs) ne sont pas conservées, ce qui interdit l'extraction de programme. Le système Coq conserve les preuves sous forme de lambda-termes typés (dans le calcul des constructions inductives), ainsi que le script qui a permis de la produire. On trouve dans [Barendregt et Wiedijk, 2005] une comparaison plus détaillée de ces systèmes et de quelques autres (NuPRL, Agda, HOL, Lego...)

Ces systèmes disposent de librairies plus ou moins étendues, contenant des résultats mathématiques standard dont les preuves y ont été développées souvent au prix de longs et fastidieux efforts. Mais ces résultats peuvent alors être utilisés pour prouver une conjecture, et l'expérience montre que cela est d'autant plus rapide que l'on dispose de bibliothèques adéquates. C'est l'une des supériorité évidente des assistants de preuve sur les démonstrateurs automatiques, dont la « culture » mathématique est réduite à leurs règles d'inférence et leurs stratégies.

On peut donc déjà attribuer aux assistants de preuve le mérite d'avoir aidé à vérifier formellement de nombreux résultats fondamentaux des mathématiques, dont certains sont très élaborés (comme le premier théorème d'incomplétude de Gödel). Le résultat le plus intéressant est sans doute la preuve du théorème des quatre couleurs, obtenue par Georges Gonthier avec le système Coq. Cette preuve formelle a été obtenue en partie grâce à des techniques de démonstration automatique. Elle est bien sûr trop longue pour être lue par un mathématicien, mais le critère de de Bruijn lève les doutes qui pouvaient subsister quant à sa validité.

3.7 Conclusion

Malgré ces évidents succès de nombreux progrès restent à réaliser avant que les assistants de preuve deviennent des outils aussi utiles aux scientifiques et ingénieurs que le sont les systèmes de calcul formel. Ils réclament souvent beaucoup d'efforts à leur utilisateur et leur offrent peu d'inspiration. À moins d'être particulièrement pointilleux,

un mathématicien ne ressent pas le besoin de transcrire ses démonstrations dans un formalisme excessif qui seul permettra d'obtenir des preuves formelles.

Il faut en particulier améliorer la lisibilité et la concision des informations fournies par l'utilisateur et par le système l'un à l'autre, afin qu'une communication fructueuse puisse s'établir. Cela nécessite de meilleures interfaces, des capacités de traductions entre différents formalismes, et surtout un plus haut degré d'automatisation, afin que ce qui paraît évident à l'utilisateur ne rencontre pas un blocage de l'assistant de preuve (sauf si l'évidence s'avère fausse). En attendant que des capacités aussi contradictoires soient réunies, il semble raisonnable de développer les cadres formels vers les preuves de programmes. Non seulement par un évident intérêt économique, mais aussi parce que les preuves y sont souvent moins profondes du point de vue mathématique, et suffisamment fastidieuses pour que l'aide d'un assistant soit la bienvenue.

Références

AVRON, A. (1993). Gentzen-type systems, resolution and tableaux. *Journal of Automated Reasoning*, 10(2) :265–281.

BARENDREGT, H. et WIEDIJK, F. (2005). The challenge of computer mathematics. *Philosophical Transactions of the Royal Society A*, 363 :2351–2375.

BETH, E. W. (1955). Semantic entailment and formal derivability. *Mededelingen der Koninklijke Nederlandse Akademie van Wetenschappen. Afd. Letterkunde. Nieuwe reeks*, 18(13) :309–342.

BLEDSOE, W. W. et LOVELAND, D. W., éditeurs (1984). *Automated Theorem Proving : After 25 Years*, volume 29 de *Contemporary Mathematics*. American Mathematical Society.

BOY DE LA TOUR, T. (1992). An optimality result for clause form translation. *Journal of Symbolic Computation*, 14 :283–301.

CAFERRA, R., LEITSCH, A. et PELTIER, N. (2004). *Automated Model Building*, volume 31 de *Applied Logic (Kluwer)*. Springer (Kluwer).

FITTING, M. C. (1983). *Proof methods for modal and Intuitionistic Logics*, volume 169 de *Synthese Library*. D. Reidel, Dordrecht, Holland.

FITTING, M. C. (1996). *First-Order Logic and Automated Theorem Proving*. Springer, New York, second édition.

GENTZEN, G. (1935). Untersuchungen über das logische schlieSen. *Mathematische Zeitschrift*, 39 :176–210 and 405–431.

GIORDANO, L., GLIOZZI, V., OLIVETTI, N. et SCHWIND, C. (2009). Tableau calculus for preference-based conditional logics : PCL and its extensions. *ACM Trans. Comput. Log.*, 10(3).

GORÉ, R. (1999). Tableau methods for modal and temporal logics. *In Handbook of Tableau Methods*, pages 297–396. Kluwer Academic Publishers.

HEUERDING, A., SEYFRIED, M. et ZIMMERMANN, H. (1996). Efficient loop-check for backward proof search in some non-classical propositional logics. *In* MIGLIOLI, P.,

MOSCATO, U., MUNDICI, D. et ORNAGHI, M., éditeurs : *TABLEAUX*, volume 1071 de *Lecture Notes in Computer Science*, pages 210–225. Springer.

HINTIKKA, K. J. J. (1955). Form and content in quantification theory. *Acta Philosohica Fennica*, 8 :7–55.

KRIPKE, S. A. (1963). Semantical analysis of modal logic I, normal propositional calculi. *Zeitschr. f. math. Logik u. Grundl. d. Math.*, 9 :67–96.

LEITSCH, A. (1997). *The Resolution Calculus.* Texts in Theoretical Computer Science. Springer.

MASSACCI, F. (2000). Single step tableaux for modal logics. *J. Autom. Reasoning*, 24(3) :319–364.

OLIVETTI, N., POZZATO, G. L. et SCHWIND, C. (2007). A sequent calculus and a theorem prover for standard conditional logics. *ACM Trans. Comput. Log.*, 8(4).

ROBINSON, J. A. et VORONKOV, A. (2001). *Handbook of Automated Reasoning (2 volumes).* Elsevier and MIT Press.

SCHÜTTE, P. (1978). *Vollständige Systeme modaler und intuitionistischer Logik.* Springer Verlag.

SIEKMANN, J. et WRIGHTSON, G., éditeurs (1983). *Automation of Reasoning. Classical Papers on Computational Logic 1957-1967 and 1967-1970*, volume 1, 2. Springer-Verlag.

SLOMAN, A. (2008). The well-designed young mathematician. *Artificial Intelligence*, 172(18) :2015–2034.

SMULLYAN, R. M. (1963). A unifying principle in quantification theory. *Proceedings of the National Academy of Sciences of the U.S.A.*, 49(6) :828–832.

SMULLYAN, R. M. (1995). *First-Order Logic.* Dover Publications. Second edition.

STICKEL, M. E. (1992). A prolog technology theorem prover : A new exposition and implementation in prolog. *TCS : Theoretical Computer Science*, 104.

VIGANÒ, L. (2000). *Labelled Non-Classical Logics.* Kluwer Academic Publishers, Dordrecht.

WOS, L. (1988). *Automated Reasoning : 33 Basic Research Problems.* Prentice Hall.

WOS, L., OVERBEEK, R., LUSK, E. et BOYLE, J. (1992). *Automated Reasoning : Introduction and Applications.* McGraw-Hill.

Chapitre 4

Programmation logique

Ce chapitre présente la famille des langages de la programmation logique qui considèrent le calcul en tant que déduction dans un formalisme logique. Nous présentons d'abord les fondements et propriétés de la Programmation Logique en clauses de Horn et son instance la plus répandue, Prolog. De ce premier langage logique, initiateur du paradigme, sont nées de nombreuses extensions ; deux sont abordées en détail : la Programmation en Logique avec Contraintes, qui permet un traitement plus élégant d'autres domaines que celui des termes finis, et l'*Answer Set Programming*, qui apparaît comme une implantation effective du raisonnement non monotone et offre un traitement plus satisfaisant de la négation.

4.1 Introduction

Programmation logique [1], programmation logique avec contraintes et programmation logique non monotone sont différents paradigmes qui accompagnent l'histoire de l'intelligence artificielle. Aujourd'hui, sous couvert de syntaxes très proches, voire identiques, on trouve deux familles de programmation en logique. La plus ancienne que l'on qualifiera de « *programmation logique* » s'appréhende plutôt via une sémantique opérationnelle, elle est basée sur une théorie de la preuve et est représentée par Prolog. En ce qui concerne ce langage, on peut dire que l'aspect programmation l'emporte sur l'aspect logique. Afin de permettre d'aborder des domaines différents, notamment les domaines numériques, la programmation logique a rapidement évolué vers ce que l'on nomme la « *programmation logique avec contraintes* ». Parallèlement, on a vu l'émergence de l' « *answer set programming* » (ASP) ou « *programmation par ensembles réponses* », qui est une programmation logique non monotone avec une sémantique purement déclarative, basée sur une théorie des modèles. Ici, c'est l'aspect logique (notamment non monotone) qui l'emporte sur l'aspect programmation.

Auteurs : Arnaud Lallouet, Yves Moinard, Pascal Nicolas et Igor Stéphan.

1. Pour être précis, on devrait parler de « Programmation en logique » à la place de « Programmation logique », la plupart des activités de programmation faisant appel à de la logique.

Dans les sections suivantes, nous présentons les fondamentaux théoriques de chacune des trois approches, les logiciels permettant d'utiliser de manière effective ces formalismes, les champs d'applications de l'IA concernés par « les programmations logiques » et leurs extensions. La section 4.2 présente un survol de la programmation logique et de ses bases théoriques, la section 4.3 présente quelques aspects de la programmation logique avec contraintes tandis que la section 4.4 s'attache à décrire les dernières avancées de la recherche dans le domaine de l'ASP.

4.2 Programmation logique

L'invention du paradigme de la programmation logique [Giannesini *et al.*, 1985 ; Colmerauer *et al.*, 1973 ; Kowalski, 1974 ; Colmerauer, 1983] a été une révolution dans le domaine de l'intelligence artificielle car il a montré que la logique, jusqu'alors cantonnée à la représentation de la connaissance, pouvait aussi être un moyen efficace de calcul. Nous présentons ce paradigme dans son cadre général pour ensuite nous focaliser sur l'instance la plus répandue, la programmation logique en clauses de Horn, et enfin son implantation, le langage Prolog.

4.2.1 De la logique à la programmation logique

Le paradigme de la programmation logique considère le calcul en tant que déduction dans un formalisme logique. Ce paradigme est à mettre en perspective avec celui de la programmation impérative qui considère le calcul comme la modification d'un état global par des instructions et celui de la programmation fonctionnelle qui considère le calcul comme le résultat de l'évaluation d'une fonction. Un langage de la programmation logique est constitué d'un langage des données, d'un langage des programmes et d'un langage des questions ; ces deux derniers étant des fragments plus ou moins grands d'un formalisme logique. Les langages de la programmation logique sont des langages déclaratifs (dits « de cinquième génération ») : le programme déclare la structure du problème mais ne fournit pas explicitement de méthode opérationnelle pour le résoudre. Le formalisme logique ne doit pas seulement pouvoir représenter la connaissance mais il doit aussi être suffisamment puissant pour pouvoir définir l'ensemble des fonctions (et procédures) calculables.

Pour réaliser le calcul, un démonstrateur de théorèmes restreint aux fragments choisis est nécessaire, il doit contenir un principe de déduction. L'ensemble des hypothèses du théorème est décrit dans le langage des programmes associé au langage des données tandis que la formule qui doit être déduite des hypothèses est décrite dans le langage des questions associé au langage des données. Un programme logique exprime la connaissance du domaine en des formules dont les variables sont implicitement universellement quantifiées tandis que les questions sur cette connaissance s'appuient sur des variables existentiellement quantifiées. Une réponse attendue d'un tel programme est un ensemble d'instances pour les variables existentiellement quantifiées. L'activité de programmation pouvant être vue comme la somme d'une composante « logique », d'une composante « structure de donnée » et d'une composante « contrôle » [Wirth, 1978 ; Kowalski, 1979], cette dernière disparaît si le paradigme est poussé jusqu'à l'extrême,

c'est-à-dire jusqu'à utiliser un démonstrateur de théorèmes dont le fonctionnement reste opérationnellement opaque. Un premier élément qui discrimine un démonstrateur de théorèmes quelconque d'un mécanisme sous-jacent à l'exécution d'un langage de la programmation logique est que le premier établit une décision si « oui » ou « non » le théorème est vrai tandis que le second, à partir d'une même connaissance, calcule des réponses pour des questions existentielles. Un second élément est que le démonstrateur de théorèmes qui sert de mécanisme à l'exécution du programme devant être basé sur de la déduction, il offre une sémantique procédurale au paradigme de la programmation logique.

Les langages de la programmation logique sont des langages relationnels : le calcul est défini en terme de relations comme dans les langages de base de données ; ainsi a contrario des fonctions qui ont des paramètres en entrée et un résultat en sortie, les paramètres d'une relation suivent un principe de réversibilité et ne sont statiquement ni en entrée ni en sortie mais dynamiquement soit l'un, soit l'autre, voire les deux. Une conséquence au caractère relationnel des langages de programmation logique est qu'ils sont indéterministes, c'est-à-dire que la réponse à la question posée n'est pas nécessairement unique et (une partie suffisante de) l'ensemble de ces réponses est calculé(e).

Au départ, les langages de la programmation logique sont des langages symboliques : le langage des données est un ensemble construit sur des symboles sans sémantique propre dont la signification est celle accordée par le programmeur (a contrario des valeurs numériques), ainsi le calcul est-il purement syntaxique et, hors l'ajout de domaines spécifiques tels que les entiers, des éléments du langage des données ne sont jamais mis en rapport sémantiquement mais uniquement syntaxiquement.

4.2.2 La programmation logique en clauses de Horn

La programmation logique en clauses de Horn est sans doute l'instance la plus connue et répandue du paradigme de la programmation logique [Lloyd, 1987]. Le langage des données est ici le langage des termes construit inductivement sur un ensemble de symboles de fonction (les symboles de fonction d'arité 0 sont appelés symboles de constante) et un ensemble de variables. Ainsi : toute variable et tout symbole de constante est un terme ; si t_1, ..., t_n sont des termes et f est un symbole de fonction d'arité n alors $f(t_1, \ldots, t_n)$ est un terme. Les langages des programmes et des questions s'appuient sur la notion d'atome, de littéral et de clause, construits à partir d'un ensemble de symboles de prédicat : si p est un symbole de prédicat d'arité n et t_1, ..., t_n des termes alors $p(t_1, \ldots, t_n)$ est un atome ; un atome (resp. sa négation) est un littéral positif (resp. négatif) ; une disjonction universellement quantifiée de littéraux forme une clause. Une clause contenant exactement un littéral positif est une clause définie ; si une telle clause contient un unique littéral alors c'est un fait sinon elle est notée $A \leftarrow A_1 \ldots, A_n$ avec $A_1, \ldots, A_n$ les atomes des littéraux négatifs et A l'unique littéral positif de la clause. Le langage des programmes est celui de l'ensemble des clauses définies. Cet ensemble a pour sémantique la conjonction des clauses qui le constituent. Le langage des questions est celui des conjonctions existentiellement quantifiées de littéraux positifs.

En logique mathématique, un modèle est une interprétation des symboles d'une formule qui la rend vraie. Le mécanisme de déduction sous-jacent pour la programmation

logique en clauses de Horn est celui de la conséquence logique : une question est conséquence logique d'un programme si toute interprétation qui est modèle du programme (c'est-à-dire simultanément modèle pour toutes les clauses), est aussi un modèle pour la question. Démontrer en logique qu'une formule est conséquence logique d'un ensemble de formules est équivalent à démontrer que la conjonction des formules de l'ensemble augmenté de la négation de la formule, qui doit être conséquence, est insatisfaisable, c'est-à-dire n'admet pas de modèle. La négation d'une conjonction existentiellement quantifiée de littéraux positifs est équivalente à une clause constituée uniquement de littéraux négatifs. La réunion d'un programme et de la négation d'une question est donc un ensemble de clauses telles que chacune d'entre-elles contiendra au plus un littéral positif, ce qui est la définition de la clause de Horn et ce qui justifie le vocable de « programmation logique en clauses de Horn ».

Démontrer qu'un ensemble de formules n'admet pas de modèle, et ce dans n'importe quelle interprétation, est une tâche insurmontable dans sa grande généralité mais ne l'est pas dans le cas très particulier d'un ensemble de clauses car il suffit alors de démontrer qu'il n'y a pas de modèle de Herbrand. Une interprétation de Herbrand est une interprétation très simple qui fait le pont entre la syntaxe et la sémantique : les symboles de constante et de fonction sont interprétés en eux-mêmes. L'univers de Herbrand pour un programme est l'ensemble des termes qui peuvent être construits sans variable (et en ajoutant un nouveau symbole de constante, s'il n'y en a pas). La base de Herbrand pour un programme est l'ensemble des atomes qui peuvent être construits à partir des éléments de l'univers de Herbrand et des symboles de prédicat. Celle-ci est nécessairement un modèle pour un programme défini puisqu'il est constitué uniquement de clauses définies. Une interprétation de Herbrand est simplement un sous-ensemble de la base de Herbrand. L'intersection de tous les modèles de Herbrand est lui-même un modèle, le plus petit modèle de Herbrand qui correspond exactement à l'ensemble des éléments de la base de Herbrand conséquence logique du programme. Ainsi la sémantique déclarative d'un programme défini est double, elle aussi : plus petit modèle, en tant qu'intersection des modèles, de Herbrand et ensemble des conséquences logiques du programme.

La structure des interprétations de Herbrand munie de l'inclusion des ensembles forme un treillis complet pour tout programme défini. Sur cette structure peut être définie une sémantique procédurale dite « en chaînage avant » (car partant des faits, c'est-à-dire des clauses ne contenant aucun littéral négatif) comme point fixe d'un opérateur de conséquence immédiate T_P de l'ensemble des interprétations de Herbrand dans lui-même. Pour un programme défini P cette fonction T_P se définit comme suit : si I est une interprétation de Herbrand, si I contient $A_1, \ldots, A_n$ et si $A \leftarrow A_1, \ldots, A_n$ est une instance sans variable d'une clause définie de P et alors $A \in T_P(I)$. Cette fonction est non seulement monotone ($T_P(I)$ contient I) mais elle est aussi (sup-)continue, c'est-à-dire elle préserve les bornes supérieures. Le plus petit point fixe de cette fonction pour un programme P coïncide avec le plus petit modèle de Herbrand. Pour décrire la résolution SLD, ultime sémantique procédurale (dite « en chaînage arrière » car partant de la question), l'unification, qui est l'unique mécanisme de passage de paramètres de la programmation logique en clauses de Horn, doit être définie. Un unificateur pour deux atomes est une substitution, c'est-à-dire une fonction des variables dans les termes, qui

a pour objectif de les rendre, une fois appliquée, égaux. Si deux termes sont unifiables alors il existe nécessairement un unificateur le plus général, substitution qui instancie le moins possible les deux termes, unique au renommage des variables près. La résolution SLD applique de manière indéterministe en une dérivation SLD la règle de résolution SLD suivante : si $Q_1, \ldots, Q_{m-1}, Q_m, Q_{m+1}, \ldots, Q_r$ est une conjonction d'atomes et $A \leftarrow A_1, \ldots, A_n$ est (une copie avec des variables nouvellement renommées de manière cohérente d') une clause définie telle que θ est l'unificateur le plus général de Q_m et A alors $\theta(Q_1, \ldots, Q_{m-1}, A_1, \ldots, A_n, Q_{m+1}, \ldots, Q_r)$ est une résolvante selon la clause définie et l'atome sélectionné Q_m. La résolution SLD [Loveland, 1968 ; Luckham, 1968] est une restriction linéaire avec fonction de sélection pour des clauses définies de la résolution de Robinson [Robinson, 1965]. Une dérivation SLD d'une question mène à un échec si au moins un des atomes ne peut être éliminé et à un succès si la règle de résolution SLD parvient à éliminer tous les atomes et atteindre une résolvante vide, un succès. Le cumul des unificateurs appliqué à la question initiale est alors la réponse calculée. Une fois la stratégie de sélection de l'atome dans la résolvante fixée, l'ensemble des dérivations pour une question donnée forme un arbre SLD ; la stratégie de recherche étant la manière de parcourir cet arbre. Si la stratégie de recherche est équitable, c'est-à-dire que sur chaque résolvante sera appliquée, si possible, la règle de résolution SLD alors la stratégie de sélection peut être quelconque. L'ensemble des éléments de la base de Herbrand d'un programme défini P tels qu'il existe pour chaque élément une dérivation SLD menant à un succès coïncide avec le plus petit modèle de Herbrand de P. Ainsi la sémantique procédurale d'un programme défini est-elle aussi double et coïncide avec les deux sémantiques déclaratives offrant à tout programme une double lecture, une double approche de la programmation. De plus, la sélection de l'atome dans la résolvante et la sélection de la clause à unifier avec l'atome ouvrent la possibilité à deux formes de parallélisme.

4.2.3 Le langage Prolog

Le langage Prolog (pour « Programmation en logique ») est une instance particulière de la programmation logique en clauses de Horn, plus « programmation » que « logique ». La programmation logique en clauses de Horn est un outil théorique qui induit par l'indéterminisme, potentiellement une infinité de dérivation SLD à mener en parallèle. Seule une stratégie de recherche équitable garantit la complétude de la résolution SLD, comme par exemple par un parcours de l'arbre SLD en largeur d'abord, mais cela induit nécessairement un mécanisme trop coûteux pour être efficace ; en conséquence, c'est une stratégie de recherche par un parcours de l'arbre SLD en profondeur d'abord qui est présente dans le mécanisme de gestion de l'indéterminisme, faisant appel à une pile de retour-arrière pour sauvegarder les points de choix qui conservent les branches de l'arbre SLD non encore explorées, abandonnant par là même la complétude du langage en tant que démonstrateur de théorèmes. Une autre source d'inefficacité potentielle est l'utilisation du test d'occurrence dans l'unification qui interdit d'unifier une variable avec un terme la contenant, test nécessaire à la correction de la règle de résolution SLD ; ce test nécessitant le parcours d'un terme, il est optionnel dans Prolog et non activé par défaut. Par là même, la correction du langage en tant que démonstrateur de théorèmes est abandonnée.

Bien que la programmation logique en clauses de Horn ait la puissance du calculable, le fondement procédural qu'est la résolution SLD a été « étendu » pour offrir un plus grand pragmatisme, facilité d'expression et de contrôle. Le modèle purement syntaxique de la programmation logique est particulièrement contraignant lorsqu'il s'agit de la manipulation des entiers qui sont le plus souvent représentés par des nombres de Peano (basés sur un symbole de constante « zéro » et un symbole de fonction d'arité un « successeur »). Une telle représentation linéaire en taille par rapport à la valeur de l'entier induit des sauts de complexité qui pénalisent les algorithmes. Par pragmatisme, dans Prolog a été intégré un évaluateur pour l'arithmétique utilisant les « entiers de la machine » (le prédicat `is`). Ces entiers n'étant pas définis par induction, les propriétés par rapport au théorème d'induction ne sont pas conservées ; de plus cet évaluateur étant fonctionnel, il brise le caractère relationnel de toute définition incluant l'utilisation de cet évaluateur et lui fait perdre la réversibilité. La volonté de dépasser ces limites en préservant le caractère logique a été une des motivations de l'introduction des contraintes dans le langage (cf section 4.3).

Dans le domaine de l'expressivité, l'extension majeure est la possibilité de déduire des informations négatives. Le programme étant défini, ceci est impossible sans ajouter une nouvelle règle. La plus communément rajoutée est la « négation par l'échec sous hypothèse de monde clos » : si un atome sans variable n'est pas conséquence logique d'un programme alors la négation de cet atome est déduite. Cette négation est donc différente de la négation de la logique classique, laquelle ne serait en général pas déductible. Sans quitter la logique, le principal moyen de contrôle est l'ordre des atomes dans les clauses et l'ordre de celles-ci dans le programme. Un mécanisme de contrôle extra-logique, la coupure, permet d'éliminer des branches de l'arbre SLD. Si ce mécanisme s'avère très utile pour éliminer des branches infinies ou ne menant qu'à des échecs ou des succès déjà calculés, il se révèle très dangereux car il peut aussi supprimer des branches menant à des succès non encore calculés et modifier ainsi la sémantique du programme sans coupure.

Un autre aspect de la programmation logique présent depuis le début est la capacité de méta-programmation, c'est-à-dire pour un programme de se modifier lui-même. Elle est parfaitement illustrée par le prédicat prédéfini `call(X)` qui permet de promouvoir le symbole de fonction principal du terme `X` en symbole de prédicat et de lancer l'exécution de celui-ci. La base de clauses est gérée par l'intermédiaire des prédicats `assert` et `retract` qui permettent respectivement d'ajouter et de retirer des clauses, tandis que l'accès au programme est rendu possible par le prédicat `clause`. Le fait que les variables et les méta-variables soient mélangées a poussé les promoteurs du langage Gödel [Hill et Lloyd, 1994] à proposer une vision close de la méta-programmation dans laquelle les variables-objet sont représentées par des constantes.

Prolog a bénéficié d'une large diffusion qui a rendue possible sa standardisation en 1995 [Deransart *et al.*, 1996]. Ce standard a permis de régler définitivement la question de la syntaxe (au départ, la syntaxe dite « de Marseille » était en concurrence avec celle dite « d'Edimbourg ») ainsi que de fixer la sémantique. La question de l'efficacité de l'évaluation a également constitué un aspect important de la recherche, et c'est avec la mise au point par Warren d'une machine abstraite permettant la compilation de Prolog [Warren, 1983] que celui-ci a pu recevoir un peu de visibilité dans un cadre

industriel. Toutefois, en ce qui concerne Prolog comme langage de programmation, l'aspect prototypage a été privilégié sur le génie logiciel, notamment au travers de la notion de spécification exécutable. Ceci n'a pas favorisé le développement d'applications en production et de ce fait n'a pas permis d'imposer le langage auprès d'un large ensemble d'utilisateurs. L'absence de typage, le contrôle non classique ne permettent pas aux programmeurs formés aux langages impératifs de contrôler finement le code généré, et le système de module n'est apparu qu'en 2000, au moment où le déclin du langage était amorcé. L'arrivée de Prolog a fondé de grands espoirs du côté de la parallélisation [de Kergommeaux et Codognet, 1994], les deux sources d'indéterminisme (choix d'un atome et choix d'une règle) étant vues comme propre à extraire de larges portions de calcul indépendant.

Le lecteur trouvera dans [Colmerauer et Roussel, 1996] un récit passionnant et passionné de l'invention de Prolog par ses auteurs.

4.2.4 Au-delà de la programmation logique en clauses de Horn

Prolog est un langage de programmation qui a la puissance du calculable. De par ses caractéristiques, il se trouve particulièrement bien adapté aux domaines symboliques, tels que celui du traitement de la langue naturelle, domaine où la programmation logique est née. Prolog étant indéterministe, il est aussi bien adapté au domaine, très proche du précédent, des automates et de la compilation. Comme il offre un mécanisme de déduction, il est également bien adapté aux systèmes manipulant symboliquement une base de connaissances, et aussi au domaine du prototypage et de la spécification exécutable, puisque le typage est dynamique et, qu'en première approche, le contrôle peut être ignoré. Les implantations abouties de Prolog intègrent de nombreux modules permettant d'utiliser Prolog comme un langage « de glu » entre des solveurs de contraintes, des applications internet, de l'interface graphique, etc mais aussi de dialoguer avec d'autres langages de programmation, le plus souvent impératifs.

Les extensions de la programmation logique en clauses de Horn incluses dans Prolog n'ont pas pour objectif d'en modifier les fondements. D'autres extensions plus radicales ont pour objectif soit de changer le fragment logique considéré, c'est le cas par exemple de λ-Prolog qui remplace les termes par des λ-termes et les clauses définies par des formules héréditaires de Harrop [Nadathur et Miller, 1998], soit d'intégrer d'autres paradigmes de langage, c'est le cas par exemple du langage Mercury [Somogyi *et al.*, 1996] (logique et fonctionnel) ou du langage Oz [Van Roy *et al.*, 2003] (logique, fonctionnel, contraintes, objet et concurrent), soit d'offrir des formalismes pour décrire ses propres systèmes de contraintes, c'est le cas du langage CHR (pour « constraint handling rules ») [Fruhwirth, 1998], soit d'intégrer sous la forme de résolution de contraintes des domaines sémantiques, c'est le cas des langages *CLP(X)* [Jaffar et Lassez, 1987 ; Fages, 1996], les deux derniers aspects faisant l'objet de la section qui suit. Enfin, au contact des logiques non monotones est né le vaste domaine de la programmation logique non monotone qui fait l'objet de la section 4.4.

Code Prolog pour Fibonacci

```
fib(0,0).
fib(1,1).
fib(N,F) :-
  N > 1,
  N1 is N-1,
  N2 is N-2,
  fib(N1,F1),
  fib(N2,F2),
  F is F1+F2.
```

Code CLP pour Fibonacci

```
fib(0,0).
fib(1,1).
fib(N,F1+F2) :-
  N > 1,
  fib(N-1,F1),
  fib(N-2,F2).
```

FIGURE 1 – Comparaison entre Prolog et CLP

4.3 Programmation en logique avec contraintes

La programmation en logique avec contraintes (PLC) ou « constraint logic programming » (CLP) est apparue afin de permettre un traitement élégant d'autres domaines que celui des termes finis de la programmation logique d'origine. En effet, un langage comme Prolog ne correspond à la définition de la programmation logique que pour les clauses définies évaluées dans l'ensemble des termes finis, si toutefois le test d'occurrence est activé. D'autres domaines, comme les domaines numériques, sont très imparfaitement pris en charge par l'arithmétique fonctionnelle du `is` (cf. figure 1). De même, au départ, un meilleur traitement de la négation au travers de la contrainte de différence constituait une motivation, même si la sémantique des ASP en constitue une bien meilleure réponse (cf. section 4.4).

Le premier saut conceptuel ayant mené à la PLC fut l'intuition ayant permis le remplacement de l'unification dans Prolog II par la résolution d'équations sur un domaine [Colmerauer, 1983]. En l'occurrence, ce domaine restait celui des termes (mais cette fois infinis, ce qui permettait de justifier théoriquement l'omission du test d'occurrence) et la contrainte de différence s'est ajoutée à la contrainte d'égalité. Cette vision a permis de voir Prolog comme une instance de CLP dans laquelle on ne peut résoudre que la contrainte d'égalité sur les arbres finis. Ceci a conduit d'une part à ce que l'on appelle le « schéma CLP » [Jaffar et Lassez, 1987] dans lequel les contraintes élémentaires sont traitées en fonction de leur type par une théorie du domaine, et d'autre part à la mise au point d'un domaine d'arbres hétérogène appelé π_4 et qui constituait la base de Prolog IV [Benhamou *et al.*, 1996]. D'autre part, grâce à l'intégration de techniques de CSP (constraint satisfaction problems) [Waltz, 1975], le développement du langage CHIP (Constraint Handling in Prolog) [Dincbas *et al.*, 1988] a conduit au très fort développement de la programmation par contraintes sur les domaines finis et à mettre en évidence ses liens avec la recherche opérationnelle.

Il est utile de commencer la description de CLP par un petit exemple qui le compare à Prolog. La figure 1 propose un programme Prolog et un programme CLP calculant les valeurs de la suite de Fibonacci et montre que la programmation sur les domaines

numériques devient un prolongement naturel de la programmation sur les termes. En particulier, la version Prolog comporte le prédicat `is` et n'est donc pas réversible. Seul un appel avec le premier paramètre clos ne donnera pas un échec. La version CLP ne souffre pas de ce défaut[2] et renvoie bien `N=7` au but `fib(N,13)`.

4.3.1 Le schéma CLP et CLP($\mathcal{R}$)

En PLC, une contrainte est avant tout un objet syntaxique ayant une interprétation fixée sur un domaine particulier, comme $X \geq 2$ ou $X = Y + Z$. Les prédicats utilisables font partie de deux catégories : les prédicats de contraintes qui sont interprétés sur un domaine et les prédicats de programme. Les prédicats de programme sont traités comme en Prolog. Une clause CLP est donc de la forme $H \leftarrow c, B_1, \ldots, B_n$ où c est une conjonction de contraintes. Les clauses du programme sont traitées avec la règle de résolution CSLD qui est une extension de la règle SLD de Prolog dans laquelle les contraintes sont ajoutées à un store de contraintes puis simplifiées selon une théorie axiomatique correcte (et complète pour la satisfaction) afin d'atteindre une « forme résolue » pour laquelle la satisfaisabilité est évidente. Dans le cas de Prolog, la théorie pour l'égalité des termes est l'égalité forte de Clarke (CET) plus CWA (l'hypothèse du monde clos) et la forme résolue est un ensemble d'équations de la forme `X=t` où `X` est une variable et `t` un terme. On peut noter que la plupart de ces équations sont posées lors du passage de paramètres, bien qu'elles puissent l'être explicitement grâce au prédicat d'égalité.

Au cours de la dérivation CSLD, les contraintes issues de chaque clause ainsi que les égalités sont ajoutées aux contraintes du but et simplifiées incrémentalement grâce à un algorithme adéquat. La sémantique logique issue de la programmation logique est conservée. En particulier, les réponses calculées sont conséquences logiques du programme dans tous les modèles de la structure d'interprétation des contraintes. Des instances particulières du schéma CLP sont Prolog III [Colmerauer, 1990] et CLP($\mathcal{R}$) [Jaffar *et al.*, 1992] dans lesquelles les contraintes sont interprétées comme des relations sur les réels. Dans ces systèmes, les équations et les inéquations linéaires sont traitées séparément par l'algorithme d'élimination de Gauss-Jordan et une variante de l'algorithme du simplexe. Un aspect intéressant est que les contraintes non linéaires sont acceptées dans le langage mais non traitées par le solveur, qui devient de fait incomplet. À la place, un mécanisme de délai retarde les contraintes non linéaires dans l'espoir qu'elles le deviennent après instanciation de certaines variables. Par la suite, Prolog IV [Benhamou *et al.*, 1996] intégre une résolution sur les réels par propagation d'intervalles qui repose sur π_4, une structure d'arbres dont les feuilles peuvent être des réels, au sens mathématiques. Les booléens 0 et 1 sont des entiers, donc des réels qui sont des éléments de π_4. BNR-Prolog [Older et Vellino, 1990] a été un précurseur dans la résolution par réduction d'intervalles. Il est intéressant de noter que dans l'approche d'origine de CLP, le solveur était conçu comme une « boîte noire » qui décidait de façon opaque de la satisfiabilité des contraintes. Toutefois d'importants efforts ont assez

2. Ce programme souffre toutefois du défaut de ne pas terminer si on l'appelle avec un but insatisfaisable comme `fib(N,4)` car il tentera de générer des valeurs de plus en plus grandes pour `F1` et `F2`. Il est simple de corriger ce défaut en remarquant que dans `fib(N, F)`, on a $F \geq N$.

rapidement permis l'accès et la personnalisation du solveur afin d'en faire une « boîte de verre ».

4.3.2 CLP(FD)

Les contraintes sur les domaines finis ont par leur nature de nombreuses applications et ouvrent à CLP le champ immense de l'optimisation combinatoire. Pour des raisons de complexité (la résolution de ces contraintes est NP-complète), la satisfaction des contraintes n'est pas testée complètement. À la place les contraintes sont simplement propagées comme dans les CSP (voir [Waltz, 1975 ; Rossi *et al.*, 2006] et le chapitre II.6 du présent ouvrage) et la recherche d'une solution se fait grâce à un prédicat de labeling qui lance l'énumération des valeurs des domaines. La satisfiabilité des contraintes est testée grâce à un algorithme de consistance locale qui utilise une représentation explicite des domaines des variables, le plus souvent en conservant les valeurs encore possibles pour la variable, ou simplement ses bornes. Le résultat est incomplet, si l'algorithme renvoie *false*, alors les contraintes sont insatisfaisables mais le plus souvent il renvoie le résultat *inconnu* signifiant que la déduction est incomplète. Il ne reste toutefois pas inactif même dans ce cas puisque les algorithmes de consistance ont pour effet de réduire les domaines en supprimant des valeurs inconsistantes localement.

De ce fait, la tentation est grande d'utiliser Prolog comme un générateur de CSP, comme il est d'ailleurs possible de le faire avec un autre langage. Quand les contraintes sont posées, l'utilisateur lance le labeling qui effectue une recherche avec backtrack afin d'énumérer les solutions. La combinaison entre cette forme de chaînage avant et le chaînage arrière de Prolog simplifie l'écriture de stratégies de recherches complexes. De part ses multiples paramètres, le prédicat de labeling permet de choisir la méthode de recherche de la solution en précisant une heuristique pour le choix de la variable à énumérer, puis une heuristique pour l'ordre dans lequel les valeurs du domaine seront choisies. Enfin il propose également de choisir une solution selon une fonction de préférence en choisissant une variable qui sera minimisée ou maximisée.

4.3.3 Écrire son solveur avec CLP

Pourtant, l'usage de CLP a de multiples avantages, notamment en ce qui concerne la personnalisation de l'algorithme de recherche et l'écriture des algorithmes de propagation pour les domaines de contraintes. En effet, l'écriture de ceux-ci dans les modèles basés sur des bibliothèques de contraintes nécessite une excellente connaissance du solveur et de la façon dont il est programmé. A l'opposé, les systèmes CLP proposent souvent un mécanisme d'accès à la représentation des domaines qui permet de profiter de la recherche Prolog en écrivant son propre prédicat de labeling. Par exemple, le système Sicstus Prolog [Carlsson *et al.*, 1997] propose les prédicats `fd_min` et `fd_max` permettant de récupérer les bornes courantes d'une variable à domaine fini. Il est simple ensuite d'utiliser ces valeurs pour effectuer un branchement.

Mais il est également possible de définir facilement ses propres contraintes grâce à un langage qui s'intègre naturellement à Prolog : les indexicaux [Hentenryck *et al.*, 1998]. Un indexical est une contrainte d'appartenance de la forme X *in* r où X est une variable et r une expression ensembliste pouvant utiliser les primitives d'accès aux

domaines des autres variables. Voici un exemple permettant de définir la propagation de la contrainte $X = Y + Z$ avec la consistance de bornes :

```
X in min(Y)+min(Z)..max(Y)+max(Z)
Y in min(X)-max(Z)..max(X)-min(Z)
Z in min(X)-max(Y)..max(X)-min(Y)
```

Les systèmes clp(FD) (devenu Gnu-Prolog) [Diaz et Codognet, 2001] et Sicstus Prolog [Carlsson *et al.*, 1997] implantent ce langage. Les indexicaux sont réveillés dès qu'une variable située en partie droite est modifiée et sont donc finement intégrés dans la boucle de propagation du solveur. Ils ont bien sûr une sémantique logique évidente (le domaine de X est inclus dans l'ensemble r) mais ils ont aussi une interprétation opérationnelle puisque, après exécution de l'indexical, le nouveau domaine de la variable X est égal à l'intersection de son ancien domaine avec l'ensemble r. Les indexicaux sont bien adaptés à l'écriture de propagateurs pour des contraintes simples, et ils peuvent être exécutés efficacement.

Les « constraint handling rules » (ou CHR) [Frühwirth, 1994] étendent encore les possibilités d'écriture d'un solveur en autorisant plusieurs atomes en tête de règle . Le langage des CHR est composé de règles qui sont appliquées en chaînage avant sur un store d'atomes de contraintes. Elles se déclenchent dès qu'une règle est applicable et ce jusqu'à extinction des possibilités d'application. Elles permettent de réécrire le store de contraintes en conservant une compatibilité avec Prolog pour représenter les domaines car elles en réutilisent les termes. Les CHR comportent trois types de règles où `H`, `H'`, `C` et `B` sont des ensembles d'atomes :

- les règles de propagation de la forme : `H => C | B.`
- les règles de simplification de la forme : `H <=> C | B.`
- les règles de simpagation de la forme : `H \ H' <=> C | B.`

La signification de ces règles est la suivante. Si les atomes de la tête sont présents dans le store, et si la garde `C` est vérifiée, alors une règle de propagation ajoute simplement le corps de la règle `B` au store courant, une règle de simplification remplace la tête `H` par le corps `B` et une règle de simpagation effectue une combinaison des deux en ne remplaçant que la partie `H'` de la tête. Il s'agit bien là d'une réécriture syntaxique. Tout comme en Prolog, l'unification sert de passage de paramètre entre la contrainte du store et la règle CHR. Il est à noter que l'intégration avec Prolog est bidirectionnelle car la garde `C` peut être un prédicat Prolog dont on lance l'évaluation. Afin d'éviter un bouclage trivial, une règle de propagation ne peut être appliquée plus d'une fois au même atome. Chaque règle peut être nommée en la préfixant de `nom @`.

La correspondance entre store de contraintes et règle à tête multiple est une richesse du formalisme mais en même temps une source potentielle d'inefficacité. Le système maintient donc une notion de contrainte active pour laquelle on cherche à appliquer une règle. Si une telle règle est trouvée, alors on recherche dans le store le reste de la tête (dont les atomes sont dits passifs). La grande expressivité des CHR peut s'avérer délicate à manier. Par exemple une règle de propagation `p(X), p(Y) => ...`pourra potentiellement s'appliquer plusieurs fois sur un store composé de plus de deux atomes de prédicat `p`. De manière générale, il est conseillé d'écrire de préférence des règles de simplification ne comportant qu'une seule tête. Nous terminons cette présentation par l'exemple d'un solveur classique, celui de la contrainte $\leq$:

```
reflexivity  @ leq(X,X) <=> true.
antisymmetry @ leq(X,Y), leq(Y,X) <=> X = Y.
idempotence  @ leq(X,Y) \ leq(X,Y) <=> true.
transitivity @ leq(X,Y), leq(Y,Z) ==> leq(X,Z).
```

Par exemple, le but (`leq(X,Y), leq(Y,Z), leq(Z,X)`) est réécrit après application de la transitivité en (`leq(X,Y), leq(Y,Z), leq(Z,X), leq(X,Z)`). Par la règle d'antisymétrie, on obtient (`leq(X,Y), leq(Y,Z), X=Z`), puis l'égalité est propagée pour donner (`leq(Z,Y), leq(Y,Z), X=Z`) et enfin, par une nouvelle application de la règle d'antisymétrie (`Y=Z, X=Z`).

4.3.4 Quelques systèmes de CLP

La famille CLP a permis d'enrichir Prolog grâce à la résolution de contraintes sur de multiples domaines, ainsi que de réfléchir sur la notion même de solveur et son intégration dans un langage de haut niveau. Mais alors que la résolution de contraintes est un sujet de recherches florissant et donne lieu à de nombreuses applications industrielles, les langages de CLP semblent boudés malgré leurs immenses qualités. Manque de formation des étudiants, difficulté d'écrire un code efficace et intégré dans un environnement applicatif, les causes sont multiples et difficiles à cerner. Les systèmes de qualité pourtant ne manquent pas. Plutôt que de tenter d'être exhaustif, nous avons choisi de citer quelques systèmes qui disposent chacun d'atouts dont l'intérêt dépend de l'application visée :

- Gnu-Prolog (`http://www.gprolog.org/`) : issu du système clp(fd), Gnu-Prolog a introduit grâce aux indexicaux l'approche « boîte de verre » pour les contraintes sur les domaines finis [Diaz et Codognet, 2001]. Compilant le code Prolog vers une implantation optimisée de la WAM, il s'agit d'un système robuste et efficace.
- Sictus-Prolog (`http://www.sics.se/sicstus/`) : il s'agit d'un système commercial, et il est certainement le plus complet des compilateurs Prolog sur le plan des fonctionnalités [Carlsson *et al.*, 1997]. En effet, Sicstus Prolog inclut de puissantes bibliothèques permettant de l'interfacer dans un environnement de production : interfaces avec le langage C, ODBC, Java, .NET, tcl/tk, HTML, XML, Zinc, etc. Il possède de nombreuses structures de données prédéfinies (graphes, arbres avl, ensembles, multi-ensembles, tableaux, etc.) et une extension objets. Au niveau de la partie contraintes, il possède un solveur sur les domaines finis comportant de nombreuses contraintes globales, dont certaines originales, un solveur sur $\mathbb{R}$, $\mathbb{Q}$, sur les booléens, ainsi que l'intégration des CHR.
- Eclipse (`http://eclipseclp.org/`) : issu du premier système incorporant les domaines finis, le système CHIP de l'ECRC [Dincbas *et al.*, 1988], Eclipse possède des librairies originales qui n'ont pas d'équivalent (contraintes ensemblistes, interface avec des solveurs linéaires, librairie Propia pour réaliser des inférences propres aux contraintes sur des prédicats Prolog, etc.).
- Prolog IV (`http://prolog-heritage.org/`) n'est plus maintenu actuellement mais sa dernière version est téléchargeable. Il a 120 prédicats qui mettent en place des contraintes entre les réels, entre les entiers, entre les listes et leur

longueur, etc. C'est un langage créé par la société PrologIA et le Laboratoire d'Informatique de Marseille (CNRS) [Benhamou *et al.*, 1996]. Il a servi pour l'enseignement et pour des applications à Air Liquide.

— Autres systèmes : on peut aussi citer les systèmes Yap-Prolog (`http://www.dcc.fc.up.pt/~vsc/Yap/` qui est parallélisable, Swi-Prolog (`http://www.swi-prolog.org/`), Jekejeke Prolog (`http://www.jekejeke.ch/`), écrit en Java et incorporable dans une application Android ou encore XSB Prolog (`http://xsb.sourceforge.net/`) et B-Prolog (`http://www.probp.com/`) qui permet d'étendre les indexicaux par des règles appelées *action rules.*

4.4 *Answer set programming*

Avec sa syntaxe de programmes logiques avec négation par défaut et sa sémantique non monotone, l'*answer set programming* (ASP [Niemelä, 1999 ; Lifschitz, 2002 ; Gelfond et Leone, 2002 ; Baral, 2003 ; Lifschitz, 2008b ; Eiter *et al.*, 2009]), ou encore *programmation par ensembles réponses*, est un formalisme déclaratif apparu dans les années 90 [Gelfond et Lifschitz, 1988, 1991]. Une session spéciale de la conférence ICLP 08 [de la Banda et Pontelli, 2008] en a célébré les 20 ans. Aujourd'hui, par la disponibilité de plusieurs solveurs performants, l'ASP apparaît comme une implantation effective du raisonnement non monotone tel qu'il avait été théorisé par Reiter en logique des défauts [Reiter, 1980]. Mais, bien au-delà du raisonnement non monotone, l'ASP est également un formalisme adapté à de nombreux domaines de la représentation des connaissances en intelligence artificielle (raisonnement de sens commun, web sémantique, raisonnement causal, ...) et à la résolution de problèmes combinatoires (planification, problèmes de théorie des graphes, configuration, bioinformatique ...). Ce dernier aspect, certainement le plus prometteur pour l'ASP, est basé sur l'idée de coder chaque problème à l'aide d'un programme logique dont les modèles correspondent aux solutions du problème. Puis, à l'instar de ce qui se fait pour le paradigme SAT (cf. chapitre II.5), d'utiliser un solveur ASP pour calculer ces modèles et ainsi exhiber les solutions du problème initial.

4.4.1 Fondements théoriques

Le paradigme de l'ASP a ses racines dans la *sémantique des modèles stables* [Gelfond et Lifschitz, 1988] pour *programmes logiques normaux.* Ceux-ci sont des ensembles de règles de la forme

$$c \leftarrow a_1, \ \ldots, \ a_n, \ not\ b_1, \ \ldots, \ not\ b_m. \tag{4.1}$$

où $n \geq 0, m \geq 0$ et $c, a_1, \ldots, \ a_n, \ b_1, \ldots \ b_m$ sont des atomes propositionnels et le symbole *not* est une négation par défaut au sens de la logique des défauts de Reiter [Reiter, 1980]. Pour une telle règle r, on nomme $tête(r) = c$ la tête, $corps^+(r) = \{a_1, \ldots, \ a_n\}$ le corps positif et $corps^-(r) = \{b_1, \ldots, \ b_m\}$ le corps négatif et son sens intuitif est : «si le corps positif est vérifié et si aucun élément du corps négatif n'est vérifié, alors la tête de la règle est vérifiée». Dans le cas d'une règle où $n = m = 0$ celle-ci sera notée $c \leftarrow .$ ou plus simplement $c.$ et l'on parlera d'un *fait.*

Formellement, un ensemble d'atomes X est *clos* par rapport à un programme *défini* P (donc sans négation par défaut) si $\forall r \in P, corps^+(r) \subseteq X \Rightarrow tête(r) \in X$. Le plus petit ensemble d'atomes clos par rapport à P est noté $Cn(P)$ et il est appelé le *modèle de Herbrand minimal* de P. De manière syntaxique, $Cn(P)$ est le plus petit point fixe de l'opérateur de conséquence immédiate $T_P : 2^{\mathcal{X}} \longrightarrow 2^{\mathcal{X}}$ où $\mathcal{X}$ est l'ensemble des atomes du programme P et $T_P(A) = \{b \mid b \leftarrow a_1, \ldots, a_n. \in P \text{ et } \forall i = 1, \ldots, n, a_i \in A\}$. T_P permet ainsi de construire la suite

$$\begin{aligned} T_P^0 &= \varnothing \\ T_P^{i+1} &= T_P(T_p^i), \forall i > 0 \end{aligned}$$

telle que $\bigcup_{i \geq 0} T_p^i = Cn(P)$.

Pour un programme logique normal P et un ensemble d'atomes X le *réduit* (dit de Gelfond et Lifschitz) de P par X est le programme ci-dessous.

$$P^X = \{tête(r) \leftarrow corps^+(r). \mid r \in P, corps^-(r) \cap X = \varnothing\}$$

Celui-ci étant un programme défini, il possède donc un unique modèle $Cn(P)$. Un *modèle stable* de P est alors un ensemble d'atomes S tel que $S = Cn(P^S)$.

Comme l'illustrent les exemples suivants, un programme logique normal peut avoir aucun, un ou plusieurs modèles stables.

Exemple 10.

— $P_1 = \{\ a \leftarrow b., \quad b \leftarrow a.\ \}$ possède 1 seul modèle stable $\varnothing$, alors qu'il a deux modèles classiques $\varnothing$ et $\{a, b\}$.

— $P_2 = \{\ a \leftarrow not\ b., \quad b \leftarrow not\ a.\ \}$ possède deux modèles stables $\{a\}$ et $\{b\}$.

— $P_3 = \{\ a \leftarrow ., \quad b \leftarrow a, not\ d., \quad d \leftarrow b.\ \}$ n'en possède aucun, il est *incohérent.*

— $P_4 = \{\ c \leftarrow a, not\ b., \quad a \leftarrow .\ \}$ a un modèle stable unique $\{a, c\}$ et $P_4 \cup \{b \leftarrow .\}$ a un modèle stable unique $\{a, b\}$.

Clairement, l'orientation de la règle est fondamentale et n'est pas à confondre avec l'implication matérielle de la logique classique. C'est l'illustration du fait que l'on est dans un cadre de programmation logique et non de logique classique. Avec P_1, on remarque que tout modèle stable est un modèle classique dont chaque atome est justifié (ou soutenu ou produit) par une règle. P_2 illustre une caractéristique fondamentale de l'ASP qui est de pouvoir représenter des situations alternatives et exclusives les unes des autres. Cette potentialité sera exploitée dans la représentation de problèmes combinatoires. Le programme P_4 illustre le caractère non monotone du formalisme puisque l'ajout du fait b fait perdre la conclusion c.

Du point de vue théorie de la complexité il faut noter que déterminer si un programme logique normal possède ou non un modèle stable est un problème NP-complet. L'appartenance à cette classe de complexité explique l'intérêt de l'ASP pour la résolution de nombreux problèmes combinatoires comme on le verra dans la sous-section 4.4.3.

Très souvent, la modélisation d'un problème mène à écrire des règles avec variables. Par exemple le fameux «tous les oiseaux volent sauf les autruches» se représentera par la règle $vole(X) \leftarrow oiseau(X)$, $not\ autruche(X)$. qu'il faut alors considérer comme un schéma général couvrant l'ensemble de toutes les instances concrètes possibles de cette règle. Ce qui ramène au cas propositionnel. Cela revient à considérer la fermeture universelle des règles et faire l'*hypothèse du monde clos* ou de *fermeture du domaine* (seuls les objets nommables dans le vocabulaire fourni sont acceptés). La première restriction est habituelle en logique, où on considère souvent qu'une formule Φ sans quantificateur doit être comprise comme $\forall X_i, i \in \{1, \cdots, k\}\Phi$ si $\{X_i, i \in \{1, \cdots, k\}\}$ est l'ensemble des variables libres figurant dans Φ. La seconde restriction consiste à ne considérer que les modèles de *Herbrand* : les seules valeurs pouvant être prises par une variable sont les constantes ou les termes pouvant être écrits grâce aux symboles de fonction (y compris les symboles de constante, c'est-à-dire les symboles de fonction d'arité 0). Le cas simple n'admet pas d'autre symbole de fonction que les symboles de constante. Les systèmes actuels admettent de plus en plus en plus souvent les symboles de fonction, avec diverses restrictions pour limiter le domaine.

La classe des *programmes logiques étendus* [Gelfond et Lifschitz, 1991] autorise l'utilisation de la négation forte, celle de la logique classique, de manière à mieux représenter les connaissances. Par exemple, les règles $traverser\_rails \leftarrow not\ train.$ et $traverser\_rails \leftarrow \neg train.$ même si elles sont proches syntaxiquement n'ont pas la même sémantique. La première indique de traverser des rails si l'on n'a pas d'information sur la présence d'un train. Tandis que la deuxième indique de traverser quand on a l'information indiquant l'absence de train. D'un point de vue formel, si l'on ne s'intéresse pas aux modèles contradictoires, un programme étendu peut-être réécrit sous la forme d'un programme normal de la manière suivante : chaque littéral $\neg x$ est remplacé par un atome nx et une règle $false \leftarrow x,\ nx,\ not\ false$ est ajoutée. Cette dernière règle joue l'effet d'une *contrainte* que l'on abrège en une règle sans tête $\leftarrow x,\ nx.$, car elle empêche x et nx d'appartenir à un même modèle stable. Ainsi, la sémantique des modèles stables s'applique directement et seuls des modèles consistants sont produits.

La classe des *programmes logiques disjonctifs* [Gelfond et Lifschitz, 1991] permet l'emploi de règles de la forme suivante.

$$c_1;\ \ldots;\ c_p \leftarrow a_1,\ \ldots,\ a_n,\ not\ b_1,\ \ldots,\ not\ b_m. \tag{4.2}$$

D'un point de vue sémantique, pour un tel programme P on garde la même notion de réduit que dans le cas non disjonctif en considérant que pour une telle règle r on a $tête(r) = \{c_1;\ \ldots;\ c_p\}$. On appelle alors *answer set*, ou *ensemble réponse*, de P tout ensemble d'atomes X tel que X soit un modèle **minimal** de P^X.

Exemple 11. $P_5 = \{\ b; c \leftarrow a.,\quad a \leftarrow .\ \}$ possède 2 answer sets $\{a, b\}$ et $\{a, c\}$. $\{a, b, c\}$ n'en est pas un car il n'est pas minimal.

Il est à noter que l'introduction de la condition de minimalité dans la définition des answer sets implique que le problème de l'existence d'un answer set est un problème Σ_2^p-complet. Ainsi, le principal avantage pratique de la programmation logique disjonctive ne réside pas dans l'accroissement de l'expressivité (à cause de l'accroissement concomitant du temps de calcul) mais dans le fait que très souvent on peut écrire

des programmes de façon beaucoup plus simple et naturelle que si on s'interdit les disjonctions. Et les systèmes qui admettent ces disjonctions s'efforcent bien sûr d'être au moins aussi performants que ceux les refusant, alors qu'une part de la difficulté d'écriture des programmes est évitée à l'utilisateur.

Au final on notera que le terme *anwser set* ou *ensemble réponse* est utilisé pour désigner tout type de modèle stable et le lecteur intéressé trouvera dans [Lifschitz, 2008a] douze façons différentes de définir cette notion de modèle stable.

4.4.2 ASP et logique (plus ou moins) traditionnelle

La programmation logique a été associée dès le début aux formalismes *non monotones* (cf. chapitre 2) qui permettent d'énoncer aisément des règles avec exception et suivant la présentation de [Ferraris *et al.*, 2007] les formalismes non monotones peuvent être classés en deux catégories :

1. Les formalismes exprimés en termes de « traduction » (« translational formalisms ») comme la complétion ou la circonscription : on ajoute aux formules explicitement données des formules (qui dépendent des formules données au départ). Pour la complétion on ajoute certaines implications réciproques, pour la circonscription on ajoute des formules qui « minimisent » certaines extensions de prédicats.
2. Les formalismes exprimés en termes de « point fixe » comme la logique des défauts [Reiter, 1980] ou la logique autoépistémique [Moore, 1985].

À l'origine, l'ASP était plutôt vu comme ayant une définition en termes de point fixe [Gelfond et Lifschitz, 1988] et la relation avec la logique autoépistémique et les défauts a été clairement faite dès le début. D'ailleurs la définition rappelée ici en section 4.4.1 est « à point fixe ». Or, il se trouve qu'aujourd'hui on sait aussi exprimer la notion d'answer set en termes de « traduction ». Cela fournit donc des définitions plus conformes à celles de la logique traditionnelle, même si bien sûr le connecteur « $\leftarrow$ » conserve un rôle particulier. Voici une brève description de deux méthodes de ce type.

Relations avec la logique du « ici et là »

Un exemple de définition alternative proposée dans la littérature repose sur une logique introduite dès le début du renouveau actuel de la logique, appelée la logique du *ici et là* (« here-and-there »). O »l'exposition de [Cabalar et Ferraris, 2007] et la référence « historique » pour la logique du « ici et là » est un article de Heyting de 1930 – voir [Mancosu, 1998] pour une traduction en anglais. Cette logique peut être définie comme une logique modale propositionnelle ayant un univers à deux mondes (« ici » et « là ») où les interprétations sont des couples (X, Y) d'ensembles d'atomes satisfaisant $X \subseteq Y$ (les atomes de X sont vrais « ici » et ceux de Y « là »).

Une *formule* F de la logique du « ici et là » est une combinaison d'atomes (propositionnels) à l'aide des connecteurs $\bot, \vee, \wedge$ et $\rightarrow$ (*faux, ou, et, implique*). Les connecteurs $\neg, \top$ et $\equiv$ sont alors définis (comme en logique classique) par $\neg F =_{def} F \rightarrow \bot$, $\top =_{def} \neg\bot$ et $F \equiv G =_{def} (F \rightarrow G) \wedge (G \rightarrow F)$. Quand une interprétation (X,Y) *satisfait* une formule F, on note classiquement (X,Y) mod F : on peut utiliser sans

ambiguïté le même symbole qu'en logique classique, car ici une interprétation est un couple d'interprétations classiques. La définition récursive de cette notion de satisfaction est analogue à celle de la logique classique, sauf pour l'implication (et donc aussi pour la négation) :

1. Pour tout atome a, (X, Y) mod a si $a \in X$.
2. (X, Y) mod $F \wedge G$ si (X, Y) mod F *et* (X, Y) mod G.
3. (X, Y) mod $F \vee G$ si (X, Y) mod F *ou* (X, Y) mod G.
4. (X, Y) mod $F \rightarrow G$ si (X, Y) mod F *implique* (au sens habituel) (X, Y) mod G et
   Y mod $F \rightarrow G$ ($\rightarrow$: implication classique dans cette dernière formule).

Intuitivement, *les atomes de X sont considérés comme étant vrais* et *ceux qui ne sont pas dans Y comme étant faux.* On a précisément (X, Y) mod $\neg F$ si et seulement si Y mod $\neg F$.

Une interprétation (X, X) est *totale* et on a alors équivalence avec la satisfaction en logique classique : (X, X) mod F si et seulement si X mod F.

Pour les rapports avec les programmes logiques, on définit une *expression emboîtée* (« nested expression ») comme une formule sans symbole d'implication $\rightarrow$ et une *règle étendue* (expression non habituelle, utilisée ici pour éviter les ambiguïtés) comme une formule $F \rightarrow G$ où F et G sont des expressions emboîtées. On a ainsi une tête G et un corps F comme d'habitude en programmation logique, mais ici on peut avoir des *et* et des *ou* dans chacun des deux. Au lieu d'écrire $G \leftarrow F$ comme traditionnellement en programmation logique, on écrit ici comme en logique classique $F \rightarrow G$, et on remplace « *not* » par le symbole de négation « $\neg$ ». On n'admet donc pas la « vraie négation » de certains programmes logiques, ce qui n'est pas une vraie restriction.

Une règle étendue peut être transformée de façon naturelle en un ensemble de règles classiques du type (4.2), appelées ici *règles non emboîtées.* Par exemple (toujours avec les notations « logiques »), $(b \vee (c \wedge \neg d)) \rightarrow (a \wedge (e \vee p))$ sera considérée comme l'ensemble des règles non emboîtées suivantes : $\{b \rightarrow a, \quad (c \wedge \neg d) \rightarrow a, \quad b \rightarrow (e \vee p), \quad (c \wedge \neg d) \rightarrow (e \vee p)\}$.

Ainsi, la notion d'ensemble réponse pour des règles étendues est bien définie à partir de la notion habituelle. On va voir qu'on peut encore étendre la notion de règle jusqu'à permettre d'emboîter aussi les "$\rightarrow$".

Voici donc l'écriture « logique » des règles non emboîtées, les L_i désignant des atomes propositionnels :

$$(L_{k+1} \wedge \ldots \wedge L_n) \rightarrow (L_1, \vee \ldots \vee L_k) \quad (0 \leq k \leq n) \tag{4.3}$$

Comme d'habitude, une conjonction vide ($n = k$) est assimilée à la formule $\top$ et une disjonction vide ($k = 0$) à $\bot$.

Il y a une correspondance très forte entre les ensembles réponses et la logique du « ici et là » (voir par exemple des références dans [Cabalar et Ferraris, 2007]). Mais on obtient une correspondance encore meilleure avec un raffinement récent, la *logique de l'équilibre* (« equilibrium logic »[Pearce, 1996]). La « logique de l'équilibre » ne garde, dans la logique du « ici et là », que les *modèles de l'équilibre*, qui sont pour une formule F, les modèles de F pour lesquels on a : $\forall X \subseteq Y, (X, Y)$ mod F si et

seulement si $X = Y$. Comme cette définition ne dépend que de Y, on se retrouve avec des interprétations classiques Y (on pourrait noter $Y \mod_{equil} F$ afin d'éviter toute ambiguïté avec la satisfaction classique).

Pearce a démontré qu'un ensemble Y d'atomes est un ensemble réponse d'un programme logique constitué de règles étendues si et seulement si c'est une modèle de l'équilibre de ce programme.

Comme la logique de l'équilibre permet d'écrire des formules propositionnelles sans restriction, on peut permettre des emboîtements aussi pour les implications $\rightarrow$. Toute formule propositionnelle peut être considérée comme un programme logique, et non plus seulement les règles classiques, même « étendues » comme ci-dessus. On peut, comme [Cabalar et Ferraris, 2007], se demander si cette extension a encore une utilité dans le cadre de la programmation logique. Mais il est très possible que l'habitude vienne de considérer naturellement des règles comme $rg1 : (p \leftarrow q) \vee r$ et $rg2 : p \leftarrow (q \leftarrow r)$ comme ayant un sens clair (on reprend ici les notations « programmation logique »). En tout cas on obtient ainsi une façon assez naturelle de généraliser la notion de « programme logique » à des règles de ce genre. La logique de l'équilibre montre que $rg1$ et $rg2$ sont équivalentes (au sens fort défini ci-dessous) respectivement aux ensembles de règles emboîtées suivants :

$rg1' : \{(p \vee r) \leftarrow q, \quad (\neg q \vee r) \leftarrow \neg p\}$;

$rg2' : \{p \leftarrow \neg r, \quad p \leftarrow q, \quad (p \vee \neg q \vee r) \leftarrow\}$.

Il s'agit donc d'une nouvelle définition des ensembles réponses, liée à une logique issue de recherches du début du $XX^{ème}$ siècle sur des logiques alternatives. Et cette définition étend très naturellement la notion de « règle » d'un programme logique.

La notion d'équivalence dans la logique de l'équilibre correspond à ce qu'on appelle *équivalence forte* entre programmes logiques : non seulement les programmes ont les mêmes ensembles réponses, mais si on les complète par un programme arbitraire, on obtient toujours deux programmes ayant les mêmes ensembles réponses. Il s'agit de la notion d'équivalence la plus pertinente pour les programmes logiques [Lifschitz *et al.*, 2001] (même si [Ferraris *et al.*, 2010] évoque une « équivalence » encore plus forte). Cette correspondance avec la logique de l'équilibre permet de démontrer que tout programme logique est fortement équivalent à un programme non emboîté, c'est-à-dire à un ensemble (une « conjonction ») de règles du type (4.3) (des « clauses »). Ainsi, on peut considérer que les ensembles de règles non emboîtées jouent, pour ces programmes logiques très généraux, le rôle *des formes normales conjonctives* pour la logique classique [Cabalar et Ferraris, 2007]. Cela montre aussi d'ailleurs que l'on peut sans perte d'expressivité se limiter aux programmes plus classiques (non emboîtés), avec la restriction qu'il n'existe pas d'algorithme polynomial qui fasse cette conversion dans tous les cas (en conservant le vocabulaire initial). Notons pour terminer que [Cabalar et Ferraris, 2007] définit aussi ce qui correspondrait à l'analogue en programmation logique des formes normales disjonctives.

Relations avec la circonscription

La *circonscription*, introduite par McCarthy il y a plus de trente ans [McCarthy, 1980, 1986] est une formalisation logique rigoureuse de la notion naturelle de « modèle minimal », destinée par exemple à remplacer automatiquement les « si » des défini-

tions en des « si et seulement si ». En ce sens, il s'agit d'étendre la « complétion de prédicats » [Clark, 1977] à des formules logiques quelconques.

La « circonscription » se traduit par l'ajout à la théorie donnée d'un ensemble de formules (ou d'une formule du second ordre) qui dépend des formules explicitement données dans la théorie. Il s'agit d'une extension assez naturelle à la logique classique qui permet de faire du raisonnement *non monotone* : un résultat vrai à partir d'un ensemble de données peut cesser d'être vrai si on étend l'ensemble des données. Ce genre de comportement est indispensable pour une traduction satisfaisante des règles avec exceptions, car il permet d'apprendre au fur et à mesure de nouvelles exceptions sans remettre en cause toute la théorie déjà connue.

Des connexions entre circonscription et « ensembles réponse » sont connues depuis longtemps et il existe une littérature abondante à ce sujet (voir par exemple [Yahya et Henschen, 1985 ; Gelfond *et al.*, 1986 ; Lifschitz, 1989 ; Yuan et You, 1993 ; Sakama et Inoue, 1993]). Nous nous limiterons ici à la connexion décrite dans [Ferraris *et al.*, 2010]. Comme la circonscription concerne le calcul des prédicats, on part ici de programmes admettant des atomes prédicatifs. Un programme logique est ici un ensemble de formules φ sans variable libre, implicitement fermées universellement ($\forall x_1, \ldots, x_n \varphi$ où les variables figurant dans φ sont les x_i) de la logique du premier ordre. Un programme plus traditionnel (ensemble fini de règles (4.2), sans « vraie négation » donc, mais avec des atomes du premier ordre), est ainsi transformé en une formule logique classique de la façon suivante :

1. Remplacer les « , » par des $\wedge$, les « ; » par des $\vee$ et les *not* par des $\neg$.
2. Remplacer *Tête* $\leftarrow$ *Corps* par l'implication classique *Corps* $\rightarrow$ *Tête* .
3. Prendre la conjonction de la clôture universelle de ces formules.

Ainsi, le programme $\{p(a) \leftarrow .,\quad q(b) \leftarrow .,\quad r(x) \leftarrow p(x), not\ q(x).\}$ donne la formule $p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x))$. De plus, $\neg F$ est considéré comme une abréviation de $F \rightarrow \bot$ et donc une contrainte $\leftarrow p(x), not\ q(x).$ correspond à la formule $\forall x \neg(p(x) \wedge \neg q(x))$.

La *circonscription* envisagée ici est une version simplifiée où tous les prédicats sont *circonscrits*. Rappelons brièvement les définitions dans ce cas.

Si $\mathbf{p} = (p_1, \ldots p_n)$ et $\mathbf{q} = (q_1, \ldots q_n)$, où les p_i et q_i sont des symboles de prédicats (chaque p_i ayant la même arité que q_i), on note

$\mathbf{p} \leq \mathbf{q}$ si pour tout $i \in \{1, \ldots, n\}$, on a $\forall \mathbf{x}(p_i(\mathbf{x}) \rightarrow q_i(\mathbf{x}))$,
où $\mathbf{x}$ est la liste des variables de p_i, et

$\mathbf{p} < \mathbf{q}$ si $\mathbf{p} \leq \mathbf{q}$ et $\mathbf{q} \not\leq \mathbf{p}$.

Pour une formule F, $Circ[F]$ représente la formule (du second ordre)
$F \wedge \neg \exists \mathbf{q}((\mathbf{q} < \mathbf{p}) \wedge F(\mathbf{q}))$.

Ici, $\mathbf{p}$ est la liste des symboles de prédicats apparaissant dans F, $\mathbf{q}$ est un n-uplet correspondant de variables de prédicats, et $F(\mathbf{q})$ désigne la formule F dans laquelle chaque occurrence d'un atome $p_i(t_1, \ldots, t_l)$ est remplacée par $q_i(t_1, \ldots, t_l)$. On ne garde ainsi que les modèles de F où l'extension des prédicats est minimale.

L'apport de [Ferraris *et al.*, 2010] consiste à montrer qu'une très légère modification de cette définition classique fournit la notion de modèle stable : on remplace $F(\mathbf{u})$ (substitution des u_i aux p_i originaux) par $F^*(\mathbf{u})$ défini récursivement ainsi :

1. $p_i(t_1,\ldots,t_m)^* \quad =_{def} \quad u_i(t_1,\ldots,t_m)$;
2. $(t_1 = t_2)^* \quad =_{def} \quad (t_1 = t_2)$; $\quad \bot^* \quad =_{def} \quad \bot$;
3. $(F \wedge G)^* \quad =_{def} \quad F^* \wedge G^*$; $\quad (F \vee G)^* \quad =_{def} \quad F^* \vee G^*$;
4. $(F \rightarrow G)^* \quad =_{def} \quad (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
5. $(\forall x F)^* \quad =_{def} \quad (\forall x F)$; $\quad (\exists x F)^* \quad =_{def} \quad (\exists x F)$

La seule différence visible ici avec la définition de $F(\mathbf{u})$ concerne (là encore) l'implication. Bien sûr, il y a (là encore) également une différence concernant la négation « $\neg\_$ », puisqu'elle se traduit par « $\_ \rightarrow \bot$ » : En particulier, si F_1 est le symbole propositionnel (prédicat d'arité 0) a, si F_2 est $\neg a$ et si F_3 est $\neg\neg a$ on a $F_1^*(u) = u[= F_1(u)]$, $F_2^*(u) = \neg u \wedge \neg a[\not= F_2(u) = \neg u]$ et $F_3^*(u) = \neg(\neg u \wedge \neg a)[\not= F_3(u) = \neg\neg u \equiv F_1(u)]$.

Cela fournit une seconde façon de définir en « logique pure » la notion de modèle stable. On renvoie le lecteur intéressé aux textes cités pour plus de détails. En particulier, notre exposition de la méthode du « ici et là » est restreinte au cas propositionnel, mais c'est uniquement pour simplifier l'exposition succincte donnée ici. En réalité, les deux méthodes permettent de définir des « programmes logiques » (avec prédicats d'arité quelconque) en un sens plus étendu que classiquement, et donnent une syntaxe et une sémantique purement logique. Un avantage de ces extensions, en restant dans le cadre de la programmation logique (sans souci théorique particulier) est qu'elles peuvent inclure certains des ajouts utilisés classiquement en programmation logique. De tels ajouts se révèlent en effet indispensables dès que l'on veut traiter des problèmes « réels » en ASP et sont introduits dans tous les systèmes ASP qui tournent effectivement. Le fait d'englober ces ajouts dans les fondements théoriques constitue donc une avancée intéressante. L'autre avantage est que cela permet de mieux appréhender la signification logique des « ensembles réponses ».

4.4.3 Représentation des connaissances et résolution de problèmes en ASP

Au fil des années et notamment grâce à la disponibilité de plusieurs solveurs dédiés (voir la sous-section 4.4.4), l'ASP a démontré sa très grande souplesse pour représenter et résoudre de nombreux problèmes entrant dans le champ de l'intelligence artificielle. Même si les frontières ne sont pas totalement étanches entre elles, nous distinguerons deux catégories de problèmes : ceux de type principalement combinatoires et ceux concernés par la représentation des connaissances pour le raisonnement de sens commun. Ne pouvant citer exhaustivement toutes les applications basées sur l'ASP nous invitons le lecteur à se référer aux actes des conférences [Erdem *et al.*, 2009b ; Hermenegildo et Schaub, 2010] et celles des années précédentes pour en découvrir d'autres.

L'ASP pour résoudre des problèmes combinatoires

L'article [Niemelä, 1999] présente en détail cette faculté de l'ASP à représenter et résoudre des problèmes combinatoires (de type CSP) issus de la théorie des graphes, de la planification (cf. chapitre II.9), des puzzles logiques, des jeux de type Sudoku (cf. chapitre II.2), ...En particulier, ceux issus de la *bioinformatique* (cf. chapitre III.6)

semblent très bien adaptés à une résolution via l'ASP comme cela a déjà été le cas pour quelques uns [Erdem *et al.*, 2009a ; Gebser *et al.*, 2011b, 2010].

D'un point de vue théorique tout problème NP doit pouvoir se coder de manière polynomiale à l'aide d'un programme logique normal et de même un problème Σ_2^p le sera à l'aide d'un programme logique disjonctif. Des problèmes appartenant à des classes de complexité supérieure peuvent être encoder de manière polynomiale à l'aide de l'ASP sous la forme d'un programme logique au 1er ordre (voir par ex [Stéphan *et al.*, 2009]). Mais, la résolution par un solveur nécessitant un passage au propositionnel (voir section 4.4.4) il en résultera une augmentation exponentielle de la taille du programme propositionnel généré.

D'un point de vue pratique, représenter en ASP un programme combinatoire consiste à écrire un programme logique normal comportant les trois catégories de règles suivantes :

— des règles de description/énumération des données,
— des règles de «*guess*» pour décrire l'espace de recherche et générer tous les ensembles de réponses possibles, toutes les solutions potentielles du problème,
— des règles de «*check*» pour décrire les contraintes et supprimer les ensembles qui ne peuvent pas être des solutions,

comme illustré dans l'exemple 12.

Exemple 12. 3-coloration d'un graphe

DONNÉES
// les n sommets du graphe et ses arêtes
$s(1) \leftarrow .... s(n) \leftarrow .... a(1,4) \leftarrow ....$
GUESS
// un sommet est rouge s'il n'est pas vert, ni bleu
$rouge(X) \leftarrow s(X),\ not\ vert(X),\ not\ bleu(X).$
// un sommet est vert s'il n'est pas rouge, ni bleu
$vert(X) \leftarrow s(X),\ not\ rouge(X),\ not\ bleu(X).$
// un sommet est bleu s'il n'est pas rouge, ni vert
$bleu(X) \leftarrow s(X),\ not\ rouge(X),\ not\ vert(X).$
CHECK
// deux voisins ne doivent pas être tous les deux de la même couleur
$\leftarrow a(X,Y),\ rouge(X),\ rouge(Y).$
$\leftarrow a(X,Y),\ vert(X),\ vert(Y).$
$\leftarrow a(X,Y),\ bleu(X),\ bleu(Y).$

On remarquera à nouveau ici l'utilisation de règles sans tête que l'on appelle *contraintes*. Une telle règle $\leftarrow\ corps.$ est sémantiquement équivalente à la règle $bug \leftarrow corps, not\ bug.$, avec *bug* un atome n'apparaissant nulle part ailleurs dans le programme. Ainsi, aucun ensemble d'atomes satisfaisant le corps ne pourra être accepté comme answer set du programme. Programmer en ASP, consiste donc de manière totalement déclarative à écrire des règles capables de générer tous les «points» de l'espace de recherche et à ajouter des contraintes qui vont supprimer les ensembles ne respectant pas les contraintes du problème initial.

Selon les solveurs employés, on pourra utiliser un certain nombre d'extensions ou facilités de codage (la plupart déjà introduites dans [Niemelä *et al.*, 1999]) parmi lesquelles :

- les expressions arithmétiques ($X + Y < Z, Z = Y \ mod \ 2, \ldots$) ;
- des littéraux conditionnels ($\{p(X) : q(X)\}$ représentant l'énumération $\{\ldots, p(a_i), \ldots\}$ satisfaisant également $q(a_i)$) ;
- des contraintes de cardinalité comme ($K_{min} \ \{\ldots, l_i, \ldots\} K_{max}$ représentant les ensembles contenant entre K_{min} et K_{max} littéraux parmi les l_i), une version simplifiée en étant la règle de choix comme dans $\{x, y, z\}$;
- des extensions avec des littéraux pondérés ($P_{min}\{\ldots, l_i = p_i \ldots\} P_{max}$, dans ce cas la somme des poids p_i des littéraux l_i appartenant à l'ensemble réponse doit être comprise entre P_{min} et P_{max}) ;
- des fonctions d'agrégat ($count, min, max, sum, \ldots$ pour calculer une valeur numérique à partir d'un ensemble, ex : $0 <= \#count X, Y : a(X, Z, k), b(1, Z, Y) <= 3$).

Signalons enfin les directives *#minimize* et *#maximize* appliquées à une fonction objectif permettent de ne retourner que le ou les ensembles réponses satisfaisant un critère d'optimalité donné par l'utilisateur.

L'ASP pour la représentation des connaissances en IA

En tant que formalisme de programmation logique non monotone, un des premiers champs d'application de l'ASP est tout naturellement le *raisonnement par défaut* au sens de Reiter. En effet, toute règle de programme logique normal $c \leftarrow a_1, \ldots, a_n, not \ b_1, \ldots, not \ b_m.$ peut se coder en logique des défauts [Reiter, 1980] par la règle $\frac{a_1 \wedge \cdots \wedge a_n : \neg b_1, \ldots, \neg b_m}{c}$. Même si la correspondance inverse, de la logique des défauts vers les programmes logiques normaux n'est pas possible en théorie (les classes de complexité n'étant pas les mêmes), en pratique de très nombreuses connaissances de sens commun modélisées en logique des défauts peuvent se coder en un programme logique étendu en vue de leur traitement automatisé par un solveur ASP. Par exemple, pour le «frame problem», la loi d'inertie pour une propriété p indexée sur le temps se représente facilement par la règle $p(T + 1) \leftarrow p(T), not \ \neg p(T + 1)$. Ainsi, l'ASP se révèle particulièrement bien adapté pour traiter le problème de la *génération de plan* [Lifschitz, 2002].

Voici un exemple d'utilisation pratique du raisonnement par défaut. Partant d'une description textuelle d'un accident de la route, le système décrit dans [Kayser et Nouioua, 2009] est capable d'en exhiber automatiquement la cause. L'idée directrice de ce travail est que la sémantique de la langue naturelle est basée sur des normes. De même, un accident étant un évènement anormal c'est qu'il résulte de la violation d'une norme. La modélisation théorique du raisonnement a donc été réalisée dans le cadre de la logique des défauts. Puis, son implantation a été effectuée en ASP et l'utilisation des systèmes `Lparse` et `Smodels` a permis d'obtenir un système opérationnel pour toute la chaîne automatisée de traitement. Dans cet exemple d'une forme de *raisonnement causal*, l'ASP a montré sa flexibilité dans la représentation des connaissances grâce à des règles par défaut et à l'utilisation d'un langage réifié permettant de simuler des modalités et des comportements de logique de second ordre.

Ont également été traités via l'ASP le problème de la *spécificité* et de la gestion de *règles à exception* [Garcia *et al.*, 2009], l'*argumentation* [Egly *et al.*, 2010], la *fusion et la révision* de bases de connaissances [Benferhat *et al.*, 2010; Hué *et al.*, 2010],...Dans les domaines de la *gestion de connaissances* et du *web sémantique* des formalismes combinant ontologies et raisonnement à base de règles ont été mis au point par l'équipe à l'origine du système DLV [Grasso *et al.*, 2009; Ielpa *et al.*, 2009] et ont abouti à la mise au point d'applications réelles (voir chap 8 de [Eiter *et al.*, 2009]).

Enfin, autour de la sémantique de base de l'ASP ont été développées différentes extensions permettant de prendre en compte certains aspects classiques du raisonnement en intelligence artificielle. On peut ainsi citer le raisonnement vague ou flou [Nieuwenborgh *et al.*, 2007], incertain avec une approche probabiliste [Baral *et al.*, 2009] ou possibiliste [Nicolas *et al.*, 2006; Confalonieri *et al.*, 2010] et la gestion des préférences [Schaub et Wang, 2001; Brewka *et al.*, 2004].

4.4.4 Les solveurs pour l'ASP

D'un point de vue pratique l'utilisation de l'ASP est aujourd'hui rendue aisée par la disponibilité de plusieurs solveurs[3]. Pour se convaincre de la performance de ces systèmes, le lecteur pourra consulter les résultats des dernières compétitions de solveurs ASP[4] [Calimeri *et al.*, 2011] et trouvera dans l'article [Giunchiglia *et al.*, 2008] une analyse des algorithmes utilisés dans trois des plus fameux systèmes.

Comme évoqué précédemment, la représentation des connaissances en ASP passe par l'écriture d'un programme contenant des variables. Or, la quasi-totalité des solveurs travaillent avec un programme propositionnel obtenu par instanciation du programme initial via un logiciel dédié appelé *grounder* comme illustré à la figure 2. Ce grounder transforme chaque règle avec variables en l'ensemble de toutes les règles obtenues en remplaçant chaque variable par l'ensemble des termes définis par l'utilisateur : les symboles de constante qui se trouvent dans le programme, plus éventuellement des entiers, plus éventuellement des termes construits si le programme contient des symboles de fonction. Dans ce dernier cas, l'utilisateur doit en général fixer une limite à l'imbrication des termes, de la même façon que s'il utilise des entiers, il doit parfois en fixer la limite.

Sans pouvoir être exhaustif, nous citons les systèmes les plus marquants de ces dernières années. `Smodels` [Simons *et al.*, 2002] (avec `Lparse` [Syrjänen, 1998] son grounder) a été le pionnier dans le domaine. `DLV` [Leone et Faber, 2008; Leone *et al.*, 2006] (avec son grounder interne) est un système aux multiples extensions et intégrations au sein de systèmes plus complexes. Par exemple, `DLV-complex` [Calimeri *et al.*, 2008] offre la possibilité d'utiliser des structures de données (listes et ensembles) et des prédicats *built-in* dédiés. `Smodels` et `DLV` dont les performances sont encore parmi les meilleures auront été déterminants dans la popularité de l'ASP car ils ont démontré sa capacité à être un outil efficace de représentation des connaissances et du raisonnement.

3. voir `http://www.kr.tuwien.ac.at/staff/tkren/deb.html` (dépôt pour Ubuntu), `http://assat.cs.ust.hk/` et `http://www.cs.utexas.edu/users/tag/cmodels.html`

4. `https://www.mat.unical.it/aspcomp2011` [Third ASP Competition] and `https://www.mat.unical.it/aspcomp2013` [Fourth (Open) ASP Competition, open to any system based on declarative problem solving].

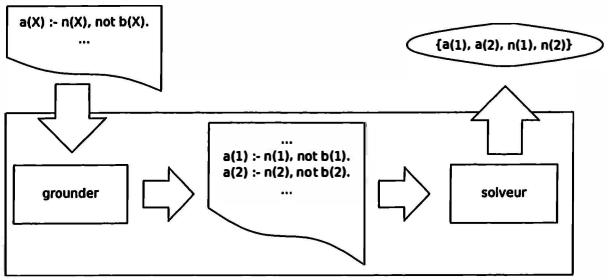


FIGURE 2 – Architecture du calcul des answer sets

Si l'on s'en tient à une vision très globale, ces solveurs sont basés sur un algorithme de recherche (de type DPLL pour SAT) alternant des phases de propagation déterministes et des points de choix non déterministes. Ils manipulent trois sous-ensembles des atomes apparaissant dans le programme : les vrais, les faux et les indéterminés. Au début de la recherche, tous les atomes sont indéterminés. Puis l'application des règles de propagation retire certains atomes des indéterminés pour les placer dans les vrais ou les faux. Lorsque plus aucune propagation n'est possible, un point de choix est créé par le placement d'un certain atome dans l'ensemble des vrais (l'autre branche du point de choix le plaçant dans l'ensemble des faux). Si, en cours de recherche un atome se retrouve classé à la fois faux et vrai, alors c'est un échec et un retour arrière est effectué. Sinon, lorsque l'ensemble des indéterminés est vide, c'est qu'un answer set a été trouvé. Précisons que chaque système a ses propres heuristiques de choix et techniques de propagation et que dans le cas des programmes disjonctifs un test final de minimalité est nécessaire pour assurer que l'on a bien un answer set.

Intégrant l'héritage de ces premiers systèmes ainsi que des techniques issues des solveurs SAT et CSP, `Clasp` [Gebser *et al.*, 2009] (avec `Gringo` [Gebser *et al.*, 2007] son grounder) est aujourd'hui l'un des systèmes les plus efficaces. C'est encore plus vrai depuis la version `Gringo` 3 [Gebser *et al.*, 2011a] qui a nettement augmenté les performances pratiques du couple `Gringo -Clasp` (facilité d'écriture et efficacité). L'approche de `Clasp` est fondée sur la détection et l'exploitation de *nogoods* de manière à avoir une recherche dirigée par les conflits et pouvoir mettre en place des techniques de *backjumping*. Par ailleurs, il existe également une famille particulière de solveurs comme `Assat` [Lin et Zhao, 2004] et `Cmodels` [Giunchiglia *et al.*, 2006] basés sur des solveurs (pseudo) booléens. En effet, via la complétion de Clark un programme logique peut être considéré comme un ensemble de clauses dont les modèles peuvent être calculés par un solveur SAT. Il reste ensuite à vérifier si un tel modèle booléen est bien un modèle stable du programme logique initial (voir programme P_1 de l'exemple 10). Si ce n'est pas le cas, une certaine boucle dans l'ensemble des règles est exhibée, convertie en nouvelles clauses et réinjectée dans le solveur SAT jusqu'à converger vers un modèle stable ou la preuve de l'incohérence du programme logique.

D'une manière générale, les solveurs ASP acceptent maintenant presque tous des symboles de fonction, avec diverses restrictions. Ils incorporent en général une arithmétique élémentaire : nombres entiers, souvent relatifs, avec fonction successeur et les

quatre opérations. Il faut en général préciser les limites que l'on entend ne pas dépasser (sinon en théorie on obtient un domaine infini, et la taille de l'instanciation explose). C'est ainsi qu'au fil des années, il est apparu que l'une des difficultés calculatoires en ASP provient, en partie et parfois exclusivement, de la phase d'instanciation du programme. Face à cela, de nouveaux grounders comme `Gringo` 3 et de nouveaux solveurs comme `GASP` [Palu *et al.*, 2009] et `ASPeRiX` [Lefèvre et Nicolas, 2009a,b] ont vu le jour. Leur point particulier est de travailler directement avec les règles avec variables et de les instancier au fur et à mesure du calcul des modèles. Ces systèmes alternent des phases de propagation basées sur l'application en chaînage avant des règles du programme et des points de choix basés sur les règles non monotones qu'il s'agit ou non d'appliquer.

Mais, si l'on se place du point de vue de la performance en temps de calcul, il est clair que le système `Clasp` associé à la dernière version de son grounder `Gringo` et à l'ensemble des outils disponibles sur `http://potassco.sourceforge.net/` est aujourd'hui le plus performant, en particulier pour s'attaquer aux problèmes de nature combinatoire. Enfin, en ce qui concerne la gestion des connaissances (notamment via des ontologies) le système `DLV` [Grasso *et al.*, 2009] a permis de développer des applications industrielles.

4.4.5 Discussion

En ASP, le langage choisi est bien plus naturel que Prolog. Par exemple : on ne s'occupe pas de l'ordre des termes, ni de l'ajout d'artifices complexes à maîtriser comme le « cut ». À un Datalog pur on ajoute diverses « contraintes » qui permettent de choisir les « meilleures » solutions, là encore l'écriture de ces contraintes est très naturelle. Il est aussi plus naturel que la logique classique, en particulier grâce à la minimisation effectuée qui permet d'écrire des « définitions » comme les mathématiciens, avec des « si », qui seront transformées automatiquement en des « si et seulement si », comme en langage naturel. Cela simplifie beaucoup l'écriture de définitions comportant un grand nombre de règles. Comme d'autre part ces « règles » ont une structure beaucoup plus simple que les formules logiques générales, elles sont plus faciles à appréhender par un utilisateur non logicien. Un exemple est celui de la fermeture transitive d'une relation binaire R qui s'écrit simplement comme suit :

```
FT(X,Y) <- R(X,Y).
FT(X,Y) <- FT(X,Z), R(Z,Y).
```

R étant donnée par exemple ainsi :

```
R(a,b) <- .  R(a,c) <- .  R(a,d) <- .
R(c,e) <- .  R(f,g) <- .  R(e,f) <- .
```

En logique classique, il faudrait ajouter un « seulement si » qui rigoureusement nécessite une formule du second ordre dans le cas général, et en tout cas qui complique singulièrement l'expression sous forme de règles. En programmation logique, on obtient avec ces seules règles la réponse attendue, où FT est la fermeture transitive de R.

En réalité, tous ceux qui ont essayé d'écrire des programmes ASP se rendent vite compte que l'écriture des programmes n'est pas si simple qu'annoncé. Voici quelques problèmes rencontrés en pratique.

– L'utilisation de la récursivité est très encadrée pour certains systèmes, et cela complique inutilement l'écriture des programmes.
– Les domaines évolutifs (non fixés au départ) posent encore quelques problèmes.
– Réutiliser un petit programme dans un plus grand est très difficile.
– Les structures de données sont simplistes : on y trouve des prédicats, des constantes, éventuellement des entiers et des symboles de fonctions. On n'y trouve guère d'ensembles, de suites, de chaînes de caractères (avec les opérations de bases appropriées dans chacun de ces cas).
– Même dans ce cadre, les systèmes ont souvent des restrictions sévères par rapport à la théorie, lesquelles compliquent singulièrement l'écriture des programmes.

La situation actuelle semble donc pouvoir être résumée ainsi :

1. Il existe un foisonnement de papiers théoriques très fouillés et qui rendent le formalisme attractif.
2. Il existe des systèmes concrets qui tournent raisonnablement bien et permettent de tester l'ASP sur des problèmes de plus en plus conséquents en taille et en intérêt.
3. Il existe des ébauches de solution au gros problème des formalismes existants, qui sont la pauvreté des structures de données et la difficulté pratique de réutiliser des morceaux de programme dans un programme plus grand. Certaines de ces ébauches s'attaquent à l'interaction avec des solveurs dédiés à des domaines ou des problèmes précis : calculs numériques, bases de données, ...
4. Il existe aussi un foisonnement d'ajouts spécifiques (« front ends ») aux systèmes ASP existants, censés traiter des domaines particuliers de façon plus naturelle (planification, causalité, diagnostic,...) L'utilisation pratique de ces systèmes montre qu'il est en fait parfois préférable de partir d'un système générique (`DLV` ou `Clasp`) car ces ajouts contiennent souvent des restrictions qui les rendent en réalité peu utilisables pour des problèmes généraux, hors du problème très précis qui a motivé leur création.

Les (assez nombreux) systèmes évoquées en point 4 ne résolvent ce problème que pour des cas très précis. Par contre, les quelques systèmes évoqués au point 3 sont très prometteurs. Il est dommage que seules de petites équipes traitent ce point, alors que cela semble être actuellement le gros problème qui freine le développement de la programmation ASP.

Un autre problème (concernant cette fois le point 2 seul) consiste en une amélioration encore plus grande de la puissance des systèmes existants. Une meilleure intégration entre instanciateur et solveur (instanciation incrémentale, unification à la volée, ...), et aussi l'utilisation simultanée de plusieurs stratégies de recherche (*bottom up* et *top down* par exemple) devraient permettre de gros progrès à ce sujet (la marge de progression à ce sujet demeure importante, même après le récent saut franchi par `Gringo3`). Du point de vue du développeur en ASP, la disponibilité d'outils de déboguage, de visualisation, de bibliothèques, ...aiderait grandement à la mise au point d'applications de grande taille.

4.5 Conclusion

La programmation logique, CLP et l'ASP sont présentés comme des paradigmes de *programmation déclarative.* Écrivez (ou décrivez) votre problème dans un langage naturel issu de la logique classique, comme un ensemble de règles écrites dans un langage simple et universel à la Datalog, utilisez des relations numériques et le système s'occupe de trouver les solutions sous la forme de contraintes résolues ou d'un ensemble d'ensembles réponses. L'utilisateur n'a pas (ou presque pas) à s'occuper de la façon dont le système va calculer effectivement les solutions car c'est le système qui effectue cette tâche. Cela permet de gagner beaucoup de temps lors de l'écriture d'un programme et, comme les solveurs sont de plus en plus efficaces, les solutions sont trouvées en un temps raisonnable pour des problèmes de plus en plus complexes.

Si l'on peut dire que la programmation en logique et en logique avec contraintes a connu son heure de gloire dans les années 80 et 90, la programmation par contraintes et les ASP qui en sont issus sont toujours l'objet d'intenses recherches. Curieusement, c'est vers une réduction de leurs prétentions calculatoires (NP-complétude au lieu de toute la puissance du calculable) que les langages logiques ont évolué. La présence de nombreux problèmes réels de nature combinatoire y est sûrement pour beaucoup. Il est clair que l'ASP a renouvelé le champ traditionnel de la programmation logique et que ce nouveau paradigme est mûr pour être utilisé dans de très nombreux domaines d'applications réelles. Mais, la confirmation de cette potentialité passera par l'amélioration des solveurs existants, tant au niveau performances qu'au niveau de la facilité d'utilisation avec le passage au premier ordre. Certainement plusieurs de ces progrès pourront être atteints en allant s'inspirer de techniques incluses dans les interpréteurs Prolog (évaluateur arithmétique, techniques d'unification, recherche *top down*, incorporation de mécanisme de traces, ...). De cette manière, l'héritage conceptuel qui existe entre les deux familles de programmation logique sera complété par un héritage des procédures de calcul et des implantations. Nul doute que cela sera au bénéfice de la programmation logique au sens large qui avec son approche à base de règles est particulièrement bien adaptée à de très nombreux domaines d'applications réelles, et très actuelles, de gestion de connaissances.

Au niveau de l'enseignement, la découverte de Prolog est toujours un choc (positif !) pour les étudiants en intelligence artificielle qui peuvent maitriser rapidement de très nombreux concepts sans avoir à entrer dans le code de bas niveau des mécanismes de recherche. La programmation à base de règle permet d'acquérir une vision de haut niveau qui peut être réutilisée dans d'autres contextes.

Références

BARAL, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press.

BARAL, C., GELFOND, M. et RUSHTON, J. N. (2009). Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9(1) :57–144.

BENFERHAT, S., BEN-NAIM, J., PAPINI, O. et WÜRBEL, É. (2010). An answer set programming encoding of prioritized removed sets revision : application to GIS. *Applied Intelligence*, 32(1) :60–87.

BENHAMOU, F., BOUVIER, P., COLMERAUER, A., GARRETA, H., GILLETA, B., MASSAT, J.-L., NARBONI, G.-A., N'DONG, S., PASERO, R., PIQUE, J.-F., TOURAÏVANE, CANEGHEM, M. V., VÉTILLARD, E. et ZHOU, J. (1996). *Le manuel de Prolog IV.* Prologia.

BREWKA, G., NIEMELÄ, I. et SYRJÄNEN, T. (2004). Logic programs with ordered disjunction. *Computational Intelligence*, 20(2) :333–357.

CABALAR, P. et FERRARIS, P. (2007). Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming*, 7(6) :745–759.

CALIMERI, F., COZZA, S., IANNI, G. et LEONE, N. (2008). Computable functions in ASP : Theory and implementation. *In Proc. Int. Conf. on Logic Programming (ICLP'08)*, pages 407–424.

CALIMERI, F., IANNI, G., RICCA, F., ALVIANO, M., BRIA, A., CATALANO, G., COZZA, S., FABER, W., FEBBRARO, O., LEONE, N., MANNA, M., MARTELLO, A., PANETTA, C., PERRI, S., REALE, K., SANTORO, M. C., SIRIANNI, M., TERRACINA, G. et VELTRI, P. (2011). The third answer set programming competition : Preliminary report of the system competition track. *In Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011*, volume 6645 de *Lecture Notes in Computer Science*, pages 388–403. Springer.

CARLSSON, M., OTTOSSON, G. et CARLSON, B. (1997). An open-ended finite domain constraint solver. *In Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'97)*, pages 191–206.

CLARK, K. L. (1977). Negation as failure. *In* GALLAIRE, H. et MINKER, J., éditeurs : *Logic and Data Bases*, pages 293–322. Plenum, New York.

COLMERAUER, A. (1983). Prolog in ten figures. *In Int. Joint Conf. on Artificial Intelligence (IJCAI'83)*, pages 487–499.

COLMERAUER, A. (1990). An introduction to prolog III. *Commun. ACM*, 33(7) :69–90.

COLMERAUER, A., KANOUI, A., ROUSSEL, P. et PASERO, R. (1973). Un système de communication homme-machine en français. Rapport technique, Groupe de Recherche en Intelligence Artificielle, Université d'Aix-Marseille.

COLMERAUER, A. et ROUSSEL, P. (1996). The birth of Prolog. *In* et RICHARD G. GIBSON, T. J. B., éditeur : *History of Programming Languages*, pages 37–52. ACM Press/Addison-Wesley.

CONFALONIERI, R., NIEVES, J. C., OSORIO, M. et VÁZQUEZ-SALCEDA, J. (2010). Possibilistic semantics for logic programs with ordered disjunction. *In Int. Symp. on Foundations of Information and Knowledge Systems (FoIKS'09)*, pages 133–152.

de KERGOMMEAUX, J. C. et CODOGNET, P. (1994). Parallel logic programming systems. *ACM Comput. Surv.*, 26(3) :295–336.

de la BANDA, M. G. et PONTELLI, E., éditeurs (2008). *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, volume 5366 de *LNCS*. Springer.

DERANSART, P., ED-DBALI, A. et CERVONI, L. (1996). *Prolog - the standard : reference manual.* Springer.

DIAZ, D. et CODOGNET, P. (2001). Design and implementation of the GNU Prolog system. *Journal of Functional and Logic Programming*, 2001(6).

DINCBAS, M., VAN HENTENRYCK, P., SIMONIS, H., AGGOUN, A. et HEROLD, A. (1988). The CHIP System : Constraint Handling in Prolog. *In* LUSK, E. et OVERBEEK, R., éditeurs : *9th International Conference on Automated Deduction*, Argonne. Springer.

EGLY, U., GAGGL, S. A. et WOLTRAN, S. (2010). Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2) :147–177.

EITER, T., IANNI, G. et KRENNWALLNER, T. (2009). Answer Set Programming : A Primer. *In* TESSARIS, S., FRANCONI, E., EITER, T., GUTIERREZ, C., HANDSCHUH, S., ROUSSET, M.-C. et SCHMIDT, R. A., éditeurs : *5th International Reasoning Web Summer School (RW 2009), Brixen/Bressanone, Italy, August 30–September 4, 2009*, volume 5689 de *LNCS*, pages 40–110. Springer.

ERDEM, E., ERDEM, O. et TÜRE, F. (2009a). HAPLO-ASP : Haplotype inference using answer set programming. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, pages 573–578.

ERDEM, E., LIN, F. et SCHAUB, T., éditeurs (2009b). *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 de *LNCS*. Springer.

FAGES, F. (1996). *Programmation logique par contraintes.* Ellipses.

FERRARIS, P., LEE, J. et LIFSCHITZ, V. (2007). A new perspective on stable models. *In Int. Joint Conf. on Artificial Intelligence (IJCAI'07)*, pages 372–379.

FERRARIS, P., LEE, J. et LIFSCHITZ, V. (2010). Stable models and circumscription. *Artificial Intelligence.*

FRUHWIRTH, T. (1998). Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3) :95–138.

FRÜHWIRTH, T. W. (1994). Constraint handling rules. *In Constraint Programming*, pages 90–107.

GARCIA, L., NGOMA, S. et NICOLAS, P. (2009). Dealing automatically with exceptions by introducing specificity in ASP. *In Proc. European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'09)*, pages 614–625.

GEBSER, M., KAMINSKI, R., KÖNIG, A. et SCHAUB, T. (2011a). Advances in gringo series 3. *In* DELGRANDE, J. P. et FABER, W., éditeurs : *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011*, volume 6645 de *Lecture Notes in Computer Science*, pages 345–351. Springer.

GEBSER, M., KAUFMANN, B. et SCHAUB, T. (2009). The conflict-driven answer set solver clasp : Progress report. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, pages 509–514.

GEBSER, M., KONIG, A., SCHAUB, T., T., S. et VEBER, P. (2010). The BioASP library : ASP solutions for systems biology. *In Proc. Int. Conf. on Tools with Artificial Intelligence (ICTAI'10)*. IEEE Computer Society.

GEBSER, M., SCHAUB, T. et THIELE, S. (2007). Gringo : A new grounder for answer

set programming. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, pages 266–271.

GEBSER, M., SCHAUB, T., THIELE, S. et VEBER, P. (2011b). Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, 11(2) :1–38.

GELFOND, M. et LEONE, N. (2002). Logic programming and knowledge representation - the a-prolog perspective. *Artificial Intelligence*, 138(1-2) :3–38.

GELFOND, M. et LIFSCHITZ, V. (1988). The stable model semantics for logic programming. *In* KOWALSKI, R. A. et BOWEN, K., éditeurs : *Proc. Conf. and Symp. on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts. The MIT Press.

GELFOND, M. et LIFSCHITZ, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4) :365–386.

GELFOND, M., PRZYMUSINSKA, H. et PRZYMUSINSKI, T. C. (1986). The extended closed world assumption and its relationship to parallel circumscription. *In Proc. Int. Symp. on Principles of Database Systems (PODS'86)*, pages 133–139.

GIANNESINI, F., KANOUI, H., PASERO, R. et CANEGHEM, M. V. (1985). *Prolog (Préface de A. Colmerauer)*. InterEditions.

GIUNCHIGLIA, E., LEONE, N. et MARATEA, M. (2008). On the relation among answer set solvers. *Annals of Mathematics and Artificial Intelligence*, 53(1-4) :169–204.

GIUNCHIGLIA, E., LIERLER, Y. et MARATEA, M. (2006). Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, 36(4) :345–377.

GRASSO, G., IIRITANO, S., LEONE, N. et RICCA, F. (2009). Some DLV applications for knowledge management. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, pages 591–597.

HENTENRYCK, P. V., SARASWAT, V. A. et DEVILLE, Y. (1998). Design, implementation, and evaluation of the constraint language cc(FD). *Journal of Logic Programming*, 37(1-3) :139–164.

HERMENEGILDO, M. V. et SCHAUB, T., éditeurs (2010). *Technical Communications of the 26th International Conference on Logic Programming, ICLP 2010, July 16-19, 2010, Edinburgh, Scotland, UK*, volume 7 de *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

HILL, P. M. et LLOYD, J. W. (1994). *The Gödel programming language*. MIT Press.

HUÉ, J., PAPINI, O. et WÜRBEL, É. (2010). Implementing prioritized merging with ASP. *In Proc. Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 138–147.

IELPA, S. M., IIRITANO, S., LEONE, N. et RICCA, F. (2009). An ASP-based system for e-tourism. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, pages 368–381.

JAFFAR, J. et LASSEZ, J.-L. (1987). Constraint logic programming. *In Proc. Symp. on Principles of Programming Languages (POPL'87)*, pages 111–119.

JAFFAR, J., MICHAYLOV, S., STUCKEY, P. J. et YAP, R. H. C. (1992). The CLP(R) language and system. *ACM Trans. Program. Lang. Syst.*, 14(3) :339–395.

KAYSER, D. et NOUIOUA, F. (2009). From the textual description of an accident to its causes. *Artificial Intelligence*, 173(12-13) :1154–1193.

KOWALSKI, R. (1974). Predicate logic as a programming language. *Information Processing.*

KOWALSKI, R. (1979). Algorithm = logic + control. *Communication of the ACM*, 22(7) :424–436.

LEFÈVRE, C. et NICOLAS, P. (2009a). A first order forward chaining approach for answer set computing. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, pages 196–208.

LEFÈVRE, C. et NICOLAS, P. (2009b). The first version of a new ASP solver : ASPeRiX. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, pages 522–527.

LEONE, N. et FABER, W. (2008). The DLV project : A tour from theory and research to applications and market. *In Proc. Int. Conf. on Logic Programming (ICLP'08)*, pages 53–68.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S. et SCARCELLO, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3) :499–562.

LIFSCHITZ, V. (1989). Between circumscription and autoepistemic logic. *In Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 235–244.

LIFSCHITZ, V. (2002). Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2) :39–54.

LIFSCHITZ, V. (2008a). Twelve definitions of a stable model. *In Proc. Int. Conf. on Logic Programming (ICLP'08)*, pages 37–51.

LIFSCHITZ, V. (2008b). What is answer set programming ? *In Proc. Int. Conf. on Artificial Intelligence (AAAI'08)*, pages 1594–1597.

LIFSCHITZ, V., PEARCE, D. et VALVERDE, A. (2001). Strongly equivalent logic programs. *ACM Trans. Comput. Log.*, 2(4) :526–541.

LIN, F. et ZHAO, Y. (2004). ASSAT : computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1-2) :115–137.

LLOYD, J. (1987). *Foundations of logic programming.* Springer-Verlag New York, Inc.

LOVELAND, D. (1968). A linear format for resolution. *In Proc. Symp. on Automatic Demonstration.*

LUCKHAM, D. (1968). Refinements theorems in resolution theory. *In Proc. Symp. on Automatic Demonstration.*

MANCOSU, P. (1998). *From Brouwer to Hilbert : the Debate on the Foundations of Mathematics in the 1920s.* Oxford University Press.

MCCARTHY, J. (1980). Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2) :27–39.

MCCARTHY, J. (1986). Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1) :89–116.

MOORE, R. C. (1985). Semantical considerations on nonmonotonic logic. *Artificial*

Intelligence, 25(1) :75–94.

NADATHUR, G. et MILLER, D. (1998). *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, chapitre Higher-order logic programming, pages 499–590. C.H.D. Gabbay and A. Robinson, Oxford University Press.

NICOLAS, P., GARCIA, L., STÉPHAN, I. et LEFÈVRE, C. (2006). Possibilistic uncertainty handling for answer set programming. *Annals of Mathematics and Artificial Intelligence*, 47(1-2) :139–181.

NIEMELÄ, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4) :241–273.

NIEMELÄ, I., SIMONS, P. et SOININEN, T. (1999). Stable model semantics of weight constraint rules. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, pages 317–331.

NIEUWENBORGH, D. V., COCK, M. D. et VERMEIR, D. (2007). An introduction to fuzzy answer set programming. *Annals of Mathematics and Artificial Intelligence*, 50(3-4) :363–388.

OLDER, W. J. et VELLINO, A. (1990). Extending prolog with constraint arithmetic on real intervals. *In Canadian Conference on Electrical and Computer Engineering.*

PALU, A. D., DOVIER, A., PONTELLI, E. et ROSSI, G. (2009). Answer set programming with constraints using lazy grounding. *In Proc. Int. Conf. on Logic Programming (ICLP'09)*, volume 5649 de *LNCS*, pages 115–129. Springer.

PEARCE, D. (1996). A new logical characterisation of stable models and answer sets. *In Non-Monotonic Extensions of Logic Programming (NMELP'96)*, pages 57–70.

REITER, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13(1-2) :81–132.

ROBINSON, J. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1) :23–41.

ROSSI, F., van BEEK, P. et WALSH, T., éditeurs (2006). *Handbook of Constraint Programming.* Elsevier.

SAKAMA, C. et INOUE, K. (1993). Relating disjunctive logic programs to default theories. *In Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'93)*, pages 266–282.

SCHAUB, T. et WANG, K. (2001). A comparative study of logic programs with preference. *In Int. Joint Conf. on Artificial Intelligence (IJCAI'01)*, pages 597–602.

SIMONS, P., NIEMELÄ, I. et SOININEN, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2) :181–234.

SOMOGYI, Z., HENDERSON, F. et CONWAY, T. (1996). The execution algorithm of Mercury, an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29.

STÉPHAN, I., MOTA, B. D. et NICOLAS, P. (2009). From (quantified) boolean formulae to answer set programming. *Journal of Logic and Computation*, 19(4) :565–590.

SYRJÄNEN, T. (1998). Implementation of local grounding for logic programs for stable model semantics. Rapport technique, Helsinki University of Technology.

VAN ROY, P., BRAND, P., DUCHIER, D., HARIDI, S., HENZ, M. et SCHULTE, C. (2003). Logic programming in the context of multiparadigm programming : the Oz experience. *Theory and Practice of Logic Programming*, 3(6) :717–763.

WALTZ, D. (1975). *Understanding line drawings in scenes with shadows*, pages 19–91. McGraw Hill.

WARREN, D. S. (1983). An abstract Prolog instruction set. Technical note 309, SRI.

WIRTH, N. (1978). *Algorithms + Data Structures = Programs.* Prentice Hall PTR.

YAHYA, A. H. et HENSCHEN, L. J. (1985). Deduction in non-Horn databases. *Journal of Automated Reasoning*, 1(2) :141–160.

YUAN, L.-Y. et YOU, J.-H. (1993). Autoepistemic circumscription and logic programming. *Journal of Automated Reasoning*, 10(2) :143–160.

Chapitre 5

Logique propositionnelle et algorithmes autour de SAT

La logique propositionnelle occupe une place à part en intelligence artificielle (IA). Extrêmement simple, elle colle parfaitement à la force de calcul brute des machines actuelles, tout en offrant un panel de solutions pragmatiques à tout un ensemble de problèmes fondamentaux d'IA. Son apparente simplicité permet en effet l'élaboration d'algorithmes compacts et efficaces – en pratique – pour attaquer toute une classe de problèmes importants, mais intraitables – en théorie –. Dans ce chapitre, nous proposons tout d'abord de cerner le type de problèmes d'IA typiquement accessibles à ce formalisme, puis nous présentons l'historique des progrès observés dans la résolution pratique du test de satisfaisabilité (SAT). Nous présentons ensuite les approches de compilation de bases de connaissances, fondamentales pour un grand nombre de problèmes d'IA, ainsi que l'extension de la logique propositionnelle aux quantificateurs (QBF).

5.1 Introduction

En observant les progrès obtenus ces dernières années autour de la logique propositionnelle, il est frappant de constater combien cette logique est ancienne. Formalisée par Aristote lui-même – on peut certainement la considérer historiquement comme la première logique –, elle connaît ces dernières années une renaissance fulgurante, notamment du fait des enjeux industriels qu'elle véhicule. On trouve en effet cette logique au cœur de démonstrateurs automatiques permettant la vérification de microprocesseurs, de programmes, ou encore de problèmes de cryptographie, de théorèmes mathématiques et bioinformatiques de première importance. Cela explique donc pourquoi, malgré un apparent paradoxe, Edmund Clarke, l'un des *Turing Award 2007* ait ainsi déclaré « la résolution pratique du problème SAT est une technologie clé pour l'informatique du 21ième siècle » [Biere *et al.*, 2009]. En compilation de bases de connaissances, aussi,

Auteur : LAURENT SIMON.

la logique propositionnelle offre un formidable compromis entre expressivité et performances.

Formalisée il y a plus de 2000 ans, cette logique remonte à la naissance du mot « raisonnement » lui-même. En effet, la racine grecque συλλογισμός, qui est à l'origine du latin *ratiocinatio*, qui donnera ensuite « raisonnement », est aussi à l'origine du mot « syllogisme », dont la signification dépasse le simple sens du mot latin pour « raisonnement ». En Grec ancien, un syllogisme portait trois sens : calcul, hypothèse et raisonnement. Ce genre de *raisonnement-calcul* se caractérise par l'articulation de deux propositions (les *prémisses*) conduisant à une *conclusion*. Il est d'ailleurs frappant de constater que déjà, pour Aristote, prémisses et conclusions ne pouvaient être que vraies ou fausses. Parmi les syllogismes proposés, le *modus ponens* est certainement l'un des plus simples et des plus connu. Il exprime simplement que « *si* a *est vrai* **et** *si* a *implique* b **alors** b *est vrai* » (voir par exemple [Chazal, 1996]). Nous verrons dans ce chapitre comment ce simple principe calculatoire de raisonnement, appliqué à des variables pourtant limitées aux valeurs vrai/faux permet de résoudre, en pratique, des problèmes industriels de tout premier plan.

5.2 Raisonner en logique propositionnelle

Une logique se définit d'abord de manière syntaxique, la sémantique étant définie par la suite sur ce langage. Intuitivement, les machines ne travailleront qu'au niveau syntaxique, manipulant des expressions vides de sens. En s'assurant ensuite que ces opérations respectent une certaine sémantique, il nous sera possible ensuite de tirer un sens des calculs effectués. Ainsi, le langage des formules propositionnelles peut se définir formellement à partir d'un ensemble fini de variables propositionnelles (ou symboles propositionnels) $\mathcal{V}$, de deux constantes *vrai* et *faux* ainsi que d'un ensemble de *connecteurs logiques* : $\neg$ (négation), $\vee$ (disjonction), $\wedge$ (conjonction), $\rightarrow$ (implication), $\leftrightarrow$ (équivalence), $\oplus$ (exclusion). On note $Var(f)$ l'ensemble des variables propositionnelles de $\mathcal{V}$ apparaissant dans f, et on nomme les formules de base ainsi : un **littéral** (ou formule atomique) est une formule de la forme x (littéral positif) ou $\neg x$ (littéral négatif) avec $x \in \mathcal{V}$. Les littéraux x et $\neg x$ sont dits **littéraux complémentaires**. Une **clause** est une disjonction finie de littéraux. Un **produit** est une conjonction finie de littéraux.

La sémantique classique de la logique propositionnelle repose sur la notion *d'interprétation*. On se donne un ensemble $\mathbb{B} = \{\mathtt{V}, \mathtt{F}\}$, constitué de deux *valeurs de vérité*, et l'on associe chaque variable propositionnelle à une de ces valeurs. Une **interprétation** I d'un ensemble de variables $V \subseteq \mathcal{V}$ est une application ayant pour domaine V et pour co-domaine $\mathbb{B} = \{\mathtt{V}, \mathtt{F}\}$. Lorsque $V \neq \mathcal{V}$, on dit que l'interprétation est *partielle* (au contraire d'une interprétation *totale*). Une fois l'interprétation des variables donnée, on peut récursivement définir l'interprétation de la formule en utilisant la table de vérité classique utilisée sur les connecteurs usuels de la logique (table 1).

x	y	$x \vee y$	$x \wedge y$	$x \rightarrow y$	$x \leftrightarrow y$	$x \oplus y$	$\neg x$
faux	*faux*	*faux*	*faux*	*vrai*	*vrai*	*faux*	*vrai*
faux	*vrai*	*vrai*	*faux*	*vrai*	*faux*	*vrai*	*vrai*
vrai	*faux*	*vrai*	*faux*	*faux*	*faux*	*vrai*	*faux*
vrai	*vrai*	*vrai*	*vrai*	*vrai*	*vrai*	*faux*	*faux*

TABLE 1 – Valuation sur les fonctions usuelles définies sur les connecteurs $\vee, \wedge, \rightarrow, \leftrightarrow, \oplus$ et $\neg$.

SAT : à la recherche de modèles

Une fois que la sémantique est fixée, on peut se demander si, étant donnée une formule f, il existe une interprétation I qui la **satisfait** (on aurait $[\![f]\!]_I = \mathtt{V}$). Si c'est le cas, on dit que I est un **modèle** de f, ce que l'on note $I \models f$. Inversement, si $[\![f]\!]_I = \mathtt{F}$, on dit que I **falsifie** f et que I est un **contre-modèle** de f, ce qui se note $I \not\models f$. Le problème qui nous intéresse ici est donc tout simplement de savoir si une formule donnée admet un modèle (problème de **satisfaisabilité**, SAT). Intuitivement, on voit bien que si aucun indice ne nous aide, on peut s'attendre à devoir essayer l'un après l'autre les $2^{Var(f)}$ modèles, ce qui est absolument prohibitif (cet ordre de grandeur est bien entendu fictif : calculer la complexité la plus faible d'un algorithme résolvant le problème SAT est cependant assez étudié, voir [Woeginger, 2003] pour un état de l'art, les résultats donnés ayant un majorant de l'ordre de $1.48^{Var(f)}$). À titre d'exemple, si l'on devait énumérer tous les modèles possibles sur une formule à 260 variables, alors les 2^{260} modèles à tester dépasseraient l'une des estimations communes du nombre de particules de l'univers. Quand on sait, de plus, que les solveurs modernes traitent des formules avec plusieurs millions de variables en quelques minutes, on comprend combien le problème de la résolution pratique de SAT est fascinant. Le nombre de modèles potentiels peut rapidement dépasser l'entendement.

Lorsqu'une formule n'admet aucun modèle, on dit qu'elle est **insatisfaisable** (on dit aussi que f est une **contradiction** ou est **incohérente** ou UNSAT), ce qui est alors noté $\models \neg f$. Au contraire, si toute interprétation sur $Var(f)$ est un modèle de f, alors on dit que f est une **tautologie**, ce qui est noté $\models f$. Par exemple : $(x \vee (y \rightarrow x))$ est une tautologie ; $(x \leftrightarrow (\neg x))$ est une formule insatisfaisable ; et $(x \vee (y \oplus z))$ est une formule satisfaisable (toute interprétation satisfaisant x est un modèle), mais non valide (l'interprétation affectant x, y et z à $\mathtt{F}$ est un contre-modèle).

Raisonner : prouver la conséquence sémantique

La notion de conséquence sémantique est importante car elle permet de caractériser les formules (g) découlant d'une hypothèse donnée (f). Elle permet de relier le simple test de satisfaisabilité (ou plutôt d'insatisfaisabilité) à des problématiques plus proches du raisonnement. Plus formellement, une formule g est une **conséquence sémantique** d'une formule f (noté $f \models g$), si tout modèle de f est un modèle de g. Non seulement cette notion de conséquence sémantique colle parfaitement au *sens* du connecteur logique $\rightarrow$ (grâce à la propriété $f \models g$ si et seulement si $\models f \rightarrow g$), mais

elle permet de relier le simple test de satisfaisabilité (ou plutôt d'insatisfaisabilité) à la notion de conséquence sémantique grâce au fait que $f \models g$ si et seulement si $f \wedge (\neg g)$ est contradictoire : pour vérifier si g est une conséquence logique de f, il « suffit » donc de vérifier que la formule $f \wedge (\neg g)$ est bien UNSAT.

Lorsque la conséquence est vérifiée dans les deux directions (*i.e.* si $f \models g$ et $g \models f$), alors les deux formules f et g ont exactement les mêmes modèles, et on dit qu'elles sont **équivalentes**. Cette notion permet de s'autoriser certaines transformations syntaxiques des formules, du moment où celles-ci préservent l'équivalence sémantique. On pourra par exemple utiliser l'*Idempotence* du connecteur $\vee$ pour réécrire $(f \vee f)$ en f. De nombreuses lois classiques de la logique permettent de réécrire une formule, comme l'élimination de la double négation $(f \equiv \neg(\neg f))$ ou encore les Lois de De Morgan. Citons tout de même deux dernières règles; la distributivité de $\wedge$ sur $\vee$: $(f \vee (g \wedge h)) \equiv (f \wedge g) \vee (f \wedge h)$: et l'élimination de $\oplus$: $(f \oplus g) \equiv (\neg f \wedge g) \vee (f \wedge \neg g)$.

Simplification et écriture normalisée de sous-formules

On vient donc de se doter d'une arme, la conséquence sémantique, pour prouver qu'une conclusion donnée découlait bien des hypothèses écrites. Il nous reste cependant encore à définir *comment* l'on va prouver cette conséquence. Généralement, avant de *raisonner* sur la formule, on va chercher à la normaliser, ce qui permet de profiter d'algorithmes efficaces (intuitivement, en normalisant, les possibilités de réécriture de la formule, donc de calculs, diminuent). Ainsi, le problème SAT, s'il est bien défini sur n'importe quels types de formules, s'exprime plus fréquemment sur des formules écrites sous **forme normale conjonctive** ($\mathcal{FNC}$) (conjonction de clauses). De manière duale, une formule f est dite sous **forme normale disjonctive** ($\mathcal{FND}$) si et seulement si elle correspond à une disjonction de produits. On notera que le problème SAT est trivial sur une formule sous $\mathcal{FND}$. Maintenant, pour pouvoir réécrire sous $\mathcal{FNC}$ (ou $\mathcal{FND}$) une formule donnée, il faut se doter de quelques outils simples. Ainsi, on doit vérifier que l'on peut bien substituer certaines de ses sous-formules par d'autres formules, simplifiées ou normalisées, mais équivalentes. Plus formellement, soient f, g et h trois formules telles que g est une sous-formule de f. On note $f_{g \mapsto h}$, la formule f dans laquelle la sous-formule h a été remplacée (on dit aussi substituée) par la formule g. Ainsi, si $h \equiv g$ alors $f \equiv f_{g \mapsto h}$. En profitant des lois usuelles des connecteurs logiques, (comme les *Lois de De Morgan*, la distributivité, l'associativité, ...), on peut donc maintenant écrire toute formule f en une formule équivalente, dans laquelle notamment tous les connecteurs $\rightarrow, \leftrightarrow$ et $\oplus$ ont été éliminés, ainsi que les doubles négations. Il ne reste alors plus qu'une formule bâtie sur la disjonction, la conjonction et la négation. Si on prend soin de « pousser » les négations jusqu'aux variables, on obtient une formule sous *forme normale négative* ($\mathcal{FNN}$). On peut en dernier lieu utiliser les propriétés de distributivité des opérateurs pour écrire la formule de départ sous $\mathcal{FNC}$ ou $\mathcal{FND}$, au choix.

Bien entendu, cela ne va pas sans contrepartie. Si l'on considère que la taille d'une formule est basée sur sa représentation linéaire, alors, lors de la réécriture, la taille de la formule normalisée peut croître de manière exponentielle (il est notamment illusoire de vouloir réécrire, en pratique, une formule donnée sous $\mathcal{FND}$ de manière à répondre ensuite simplement au problème SAT : sa taille serait alors totalement prohibitive).

Écriture efficace sous forme normale

N'ayant aucune garantie sur la réécriture de toute formule sous $\mathcal{FNC}$, Tseitin [1968] a introduit un mécanisme permettant de normaliser n'importe quelle formule sous $\mathcal{FNC}$ en préservant non pas l'équivalence, mais l'**équi-satisfiabilité**. Intuitivement, il s'agit d'introduire un ensemble de nouvelles variables dont la valeur de vérité représente la satisfaisabilité, ou non, de sous-formules. Par exemple, lorsqu'il faut encoder un *circuit*, une variable x est associée à chaque *porte* $f(x_1, \ldots, x_n)$ telle que x soit sémantiquement équivalente à la valeur f de la porte : $x \leftrightarrow f(x_1, \ldots, x_n)$. En répétant ce processus au niveau de chaque sous formule élémentaire, la formule globale peut s'écrire ensuite simplement comme la conjonction de tous les encodages, ce qui évite l'explosion due à la distribution des connecteurs logiques les uns sur les autres. Par exemple, si la sous-formule f s'écrit comme la disjonction $g \vee h$, on introduit deux nouvelles variables x_g et x_h représentant la satisfaisabilité des deux sous-formules g et h, respectivement. La satisfaisabilité de f peut alors s'écrire comme la $\mathcal{FNC}$ suivante $(\neg f \vee x_g \vee x_h) \wedge (f \vee \neg x_g) \wedge (f \vee \neg x_h)$. Les contraintes supplémentaires sur le circuit sont encodées sous forme de clauses unitaires. Grâce à cette astuce, on perd la notion d'équivalence sémantique, mais la réponse à l'existence de modèles de la formule de départ demeure. De plus, cette transformation est extrêmement efficace, puisqu'elle peut être réalisée en temps linéaire (voir 5.2.2 pour la généralisation de la règle d'introduction de variables nouvelles).

Du fait de l'importance industrielle des solveurs SAT, d'autres méthodes de transformations de circuits vers SAT ont été proposés, parfois non linéaires, mais permettant une efficacité encore accrue des solveurs SAT sur ces problèmes. On trouvera ainsi une première variante de l'encodage de Tseitin dans [Plaisted et Greenbaum, 1986] qui n'encode plus directement les valeur de vérité des sous-formules, mais raisonne plutôt sur leurs polarité, et les dépendances entre elles au travers du circuit. Toujours dans le but d'améliorer les performances des démonstrateurs SAT sur les problèmes d'origine industrielle, d'autres encodages ont été proposés [Jackson et Sheridan, 2004 ; Velev, 2004]. On pourra, de plus, citer la méthode se basant sur la représentation NICE [Chambers *et al.*, 2009], qui utilise tout d'abord une représentation par un DAG dont les feuilles sont les variables du circuit et les noeuds internes les opérateurs logiques, incluant une restriction de l'opérateur logique IFF, « si et seulement si », mais interdisant la négation en tant qu'argument ; et l'opérateur ITE « si x alors y sinon z ». Enfin, citons la méthode basée sur la représentation par DAG appelés AIGER [Biere, 2006] (utilisé dans la compétition de vérification formelle de la conférence FMCAD [1]). Ces représentations sont ensuite utilisées pour produire une $\mathcal{FNC}$ non seulement plus compacte, mais permettent aussi, expérimentalement, d'obtenir des gains de performance substantiels.

Ces résultats ont cependant été remis en cause il y a peu, grâce à une méthode [Järvisalo *et al.*, 2010] assez caractéristique des techniques réçentes issues de SAT. Ainsi, dans un premier temps, c'est le simple encodage de Tseitin qui est utilisé puis, plutôt que de raisonner spécifiquement sur la structure du problème à l'aide de méthodes *adhoc*, on effectue autant que possible des simplifications basées sur l'encodage $\mathcal{FNC}$. Cette technique élimine des clauses *inutiles* (appelées *clauses bloquées* [Kull-

1. Voir `www.cs.utexas.edu/users/hunt/FMCAD/`

mann, 1999]) ainsi que certaines variables peu fréquentes, en les « oubliant » (voir section 5.4.2). Il a été montré, dans [Järvisalo *et al.*, 2010], que cette technique capturait la puissance de la plupart des techniques de simplifications pouvant être mises en œuvre lors de la transformation en $\mathcal{FNC}$(*cône d'influence*, ...). On pourra de plus noter que, de manière assez paradoxale et sans obligatoirement utiliser de prétraitements, les performances des solveurs SAT restent souvent supérieures lorsqu'ils travaillent sur des $\mathcal{FNC}$ plutôt que directement sur des encodages préservant la structure de la formule initiale (comme AIGER ou NICE). Ainsi, si la représentation sous $\mathcal{FNC}$ est souvent jugée moins informative, car « aplatissant » les dépendances entre sous-formules, ou « cachant » les opérateurs logiques initialement exprimées (dans [Grégoire *et al.*, 2005], les auteurs proposent de retrouver justement ces dépendances fonctionnelles), cela est au final loin d'être évident. En effet, les résultats expérimentaux montrent plutôt le contraire (toujours sur les familles de problèmes industriels), et il faut bien noter que l'introduction de nouvelles variables permet justement d'exprimer par la suite des relations sémantiques puissantes entre sous-formules (par exemple, que la négation d'une partie de la formule est impliquée par une autre partie, apparemment disjointes dans l'expression initiale).

Impliqués : les théorèmes de la logique propositionnelle

La majorité des algorithmes actuels manipulent des clauses. On peut dès lors chercher à éliminer celles qui n'ont manifestement pas d'intérêt avec la sémantique classique de la logique propositionnelle, comme les clauses sous-sommées (la clause c_1 **sous-somme** ou **subsume** la clause c_2 (ou encore c_2 est sous-sommée par c_1) si et seulement si $c_1 \models c_2$. Plus simplement, cela revient à dire que tout littéral de c_1 est aussi un littéral de c_2.

De plus, on dit qu'une clause c *fondamentale* (sans littéraux complémentaires) est un **impliqué** d'une formule f si $f \models c$. c est un **impliqué premier** d'une formule f si $f \models c$ et si, pour tout impliqué c' de f, $c' \models c$ implique $c' = c$ (à l'équivalence logique près). Intuitivement, un impliqué premier de f est un *théorème* de la formule f. L'ensemble des tous les impliqués premiers de f est défini de manière unique, à l'équivalence logique près. Cet ensemble, noté $\mathcal{PI}(f)$ représente toutes les clauses non sous-sommées que l'on peut déduire à partir de f. Bien entendu, le problème de calculer $\mathcal{PI}(f)$ est beaucoup plus difficile que de prouver SAT (si $\mathcal{PI}(f)$ n'est pas réduit à la clause vide, alors f est SAT) car, en plus de la difficulté théorique pour prouver qu'une clause est un impliqué premier, il peut y en avoir un nombre exponentiellement grand.

5.2.1 Calculer pour raisonner : la résolution

La résolution [Robinson, 1965] est un mécanisme simple permettant de déduire une nouvelle clause d'après deux autres clauses. C'est l'une des règles de déduction de base en logique propositionnelle, d'abord en raison de sa simplicité, mais aussi de son efficacité pratique. Elle se définit ainsi. Soient deux clauses c_1, c_2 et x une variable propositionnelle, la règle de résolution est la règle d'inférence suivante :

$$(x \vee c_1), (\neg x \vee c_2) \vdash_{\mathcal{R}} c_1 \vee c_2$$

La clause $c_1 \vee c_2$ est appelée **clause résolvante** sur x des clauses $(x \vee c_1)$ et $(\neg x \vee c_2)$. Cette règle est très proche du raisonnement commun. Elle peut être vue comme une application directe de la règle de *coupure*, ou *syllogisme conjonctif* (si on a $f \rightarrow g$ et $g \rightarrow h$, alors on a $f \rightarrow h$), dans le cas particulier où f est une clause, g un littéral et h un produit. Bien entendu, cette simple règle de déduction est correcte : on a bien $(c_1 \vee x) \wedge (c_2 \vee \neg x) \models c_1 \vee c_2$. Il est de plus aisé de vérifier qu'elle est aussi complète pour la réfutation. Si f est insatisfaisable, alors on a la garantie qu'il existe bien une suite finie de résolutions permettant de déduire la clause vide.

Une preuve par résolution dans Σ d'une clause c' fondamentale est une suite de clauses $P = c_1, \ldots, c_k$ telle que $c_k = c'$ et telle que l'on ait, pour tout $i \leq k$: ou bien $c_i \in \Sigma$; ou bien il existe c_m et c_n ($m < i$ et $n < i$) dans P tels que c_i soit la résolvante de c_m et c_n. On note $\Sigma \vdash_{\mathcal{R}} c'$ le fait qu'il existe une telle preuve. Sauf mention contraire, nous notons simplement $\Sigma \vdash c'$ pour $\Sigma \vdash_{\mathcal{R}} c'$. La résolution permet donc de proposer des algorithmes pour la déduction automatique. La question est maintenant d'obtenir des algorithmes efficaces. On pourrait en effet saturer la base de clauses initiale [Quine, 1955] par résolution jusqu'à ce que plus aucune nouvelle information ne puisse être déduite. Si l'on peut imaginer obtenir ainsi la réponse souhaitée, cela ne serait pas sans entraîner un coût prohibitif. Pour tenter de le réduire, il est courant de limiter le nombre de résolutions possibles à chaque étape, de manière à réduire les possibilités de choix des algorithmes, en en écartant certaines qui ne seraient pas indispensables à la complétude (*efficacité*). Il faut cependant prendre gare à ne pas trop restreindre les possibilités, sans quoi le système pourrait perdre la *complétude* pour la réfutation. Dans le meilleur des cas (ou le moins pire), cette restriction peut entraîner un accroissement exponentiel de la *longueur de preuve minimale* pour la réfutation. Il faut donc trouver un compromis entre longueur de preuve, complétude et efficacité. De nombreuses restrictions ont été proposées [Chang et Lee, 1973] : on peut par exemple imposer que l'ordre des résolutions soit le même pour toutes les clauses produites dans la preuve (résolution dirigée [Davis et Putnam, 1960 ; Dechter et Rish, 1994]) ou simplement empêcher de produire toute clause demandant plusieurs résolutions sur une même variable (résolution régulière [Davis *et al.*, 1962 ; Galil, 1977]). Si ces deux restrictions ne sont pas trop fortes (elles conservent la complétude pour la réfutation) ce n'est pas le cas par exemple lorsqu'on impose de n'effectuer que des résolutions unitaires : une preuve par résolution unitaire dans Σ d'une clause c impose que toutes les résolutions soient unitaires (au moins une des deux clauses est unitaire).

En pratique, malgré la perte de complétude, la résolution unitaire est très utilisée pour ces bonnes propriétés algorithmiques (elle permet de concevoir des algorithmes résolvant les problèmes en temps linéaire (ou sous-linéaire) dans la taille de la formule manipulée [Dowling et Gallier, 1984 ; Zhang, 1997]). On la retrouve aussi au cœur des solveurs dit « modernes », qui assurent une propagation unitaire extrêmement efficace en cours de calcul. Cependant, pour baser un moteur de raisonnement uniquement sur cette méthode, on peut aussi restreindre les formules traitées à celles qui garantissent tout de même la complétude. C'est le cas, par exemple, si toutes les clauses sont des clauses de Horn (toutes les clauses ont au plus un littéral positif), ou Horn-Renomables. Plus généralement, on peut parler de base de clauses *complète pour la résolution unitaire* si la base permet de garantir la complétude pour la réfutation grâce à la résolution

unitaire.

5.2.2 Limites de la résolution

La formalisation des systèmes de preuves autour de la résolution permet de classifier la *puissance* des algorithmes qui les implantent. Ainsi, il existe des familles de problèmes dont la preuve minimale requiert un nombre exponentiellement grand d'étapes de résolutions pour la résolution régulière, mais pas pour la résolution en général.

Il a par ailleurs été montré que certains problèmes d'apparence simple ne permettent pas d'être résolus en temps polynomial par toute procédure basée sur la résolution. C'est ainsi le cas du fameux problème des pigeons [Haken, 1985], qui est donc difficile pour toute preuve par résolution. Il est frappant de constater que les meilleurs démonstrateurs SAT actuels, capables de vérifier des circuits électroniques complexes, sont incapables de démontrer en temps raisonnable que l'on ne peut pas placer 15 pigeons dans 14 trous (il leur faudrait plusieurs jours pour cela).

Pas encore de problèmes *difficiles* pour la résolution étendue

La résolution étendue [Tseitin, 1983] allie simplicité apparente et puissance. En effet, ce système de preuve est simplement la résolution avec la possibilité, à n'importe quelle étape du calcul, d'ajouter de nouvelles variables équivalentes à toute formule bâtie sur les variables déjà connues. En pratique, et sans perte de puissance, cet ajout peut être restreint à l'ajout de la variable *étendue* x et de la simple formule $x \leftrightarrow l_1 \vee l_2$ (qui peut être encodée par 3 clauses, dont 2 binaires), où x est une variable qui n'apparaissait pas jusque-là dans la preuve, alors que l_1 et l_2 sont deux littéraux qui y apparaissaient déjà (soit initialement, soit en ayant été eux-mêmes introduits). Depuis l'excellent papier de Cook au sujet de la résolution étendue [1976], ce système de preuve fascine. En effet, la puissance derrière l'ajout de lemmes en cours de preuve est tel que, jusqu'à présent, aucun problème n'a encore pu être identifié comme *difficile* pour la résolution étendu. En pratique, cependant, il est extrêmement difficile de deviner quels lemmes peuvent être introduits dans la preuve, même si des approches récentes, basés sur la notion de compression de la preuve construite, ont démontré la viabilité de cette technique [Audemard *et al.*, 2010].

5.3 À quels types de problèmes SAT s'attaque-t-il ?

Même s'il ne représente qu'une partie des raisonnements possibles en logique propositionnelle, le problème SAT occupe, comme on l'a dit précédemment, une place à part. Voyons le type de problèmes, à la fois théoriquement, et pratiquement, qu'il permet de résoudre.

5.3.1 D'un point de vue théorique

On ne peut parler du problème SAT sans aborder quelques notions de complexité. Il s'agit en effet du premier problème démontré comme NP-complet[Cook, 1971]. Cette notion est primordiale à bien des égards, notamment parce qu'elle permet de classer

les problèmes selon une complexité indépendante des algorithmes qui les résolvent en pratique. Il n'est en effet pas possible, dans le cas général, de connaître le temps d'un algorithme *a priori* sans l'exécuter. Pour estimer ce temps, des majorants théoriques peuvent être fixés mais, si on peut en trouver par l'analyse d'un algorithme donné, le doute subsiste : peut-on trouver un algorithme plus rapide pour ce problème ? On peut ainsi classer les problèmes – et non plus seulement les algorithmes – suivant leur complexité. Pour une introduction à ce sujet, le lecteur intéressé pourra consulter les incontournables [Garey et Johnson, 1979] et [Lassaigne et de Rougemont, 1996].

Pour simplifier, nous introduisons donc la notion de classe de complexité à l'aide de la notion de fonction calculable. Intuitivement, une fonction f est dite *calculable* simplement s'il existe un algorithme qui permet de calculer son résultat (nous imposons juste à ce niveau que le programme finissent un jour son calcul, quel que soit le résultat de la fonction). On peut alors mesurer le temps de calcul de f par le nombre d'étapes élémentaires de l'algorithme.

La classe P est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial. En terme de fonction calculable, ces problèmes sont caractérisés par la réponse à la question « Que vaut $f(x)$? », avec f calculable en temps polynomial par rapport à la taille de la donnée x. L'ensemble des problèmes appartenant à cette classe sont parfois appelés *faciles*. En théorie, le passage à l'échelle des exemples traités est toujours possible, moyennant des machines plus puissantes, ce qui n'est plus le cas pour les problèmes appartenant aux classes définies ci-dessous (sous l'hypothèse $P \neq NP$).

La classe NP est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial de manière non déterministe. Cette classe correspond à l'ensemble des problèmes de décision s'exprimant logiquement comme $\exists x.f(x)$, ou « existe-t-il un x tel que l'on ait $f(x)$? », avec f une fonction calculable en temps polynomial. Intuitivement, on voit que le problème consiste à *deviner* la bonne solution x. En pratique, les meilleurs algorithmes connus de recherche de ces solutions sont de coût exponentiel dans le pire des cas. Pourtant, rien ne prouve que l'on ne trouvera jamais d'algorithmes polynomiaux pour résoudre des problèmes NP-complets, même si la conjecture P≠NP est très forte. Quoi qu'il en soit, ces problèmes ont une forte caractéristique pratique : on peut vérifier une solution du problème en temps polynomial (il suffit de calculer $f(x)$).

Étant donné un problème de décision, nous pouvons définir son problème complémentaire, comme $\forall x.f(x)$, ou « a-t-on $f(x)$ pour tout x ? », où, encore une fois, f est une fonction calculable en temps polynomial par rapport à la taille de x. La classe coNP est l'ensemble de ces problèmes, dont le problème complémentaire appartient à NP. De nouveau, une très forte conjecture, non encore prouvée ou infirmée, découle de cette définition : NP≠coNP. Même si la différence entre poser une question NP ou coNP semble de prime abord similaire (répondre *non* à l'un revient à répondre *oui* à l'autre), c'est dans la longueur minimale de la preuve que la différence se fait sentir. La preuve de l'inexistence d'une solution peut être exponentiellement longue par rapport à x (sous l'hypothèse habituelle NP≠coNP). Nous n'allons donc pas directement répondre à la question « Peut-on trouver un algorithme efficace pour SAT ? », puisque malgré les efforts considérables de la communauté, cette question reste liée à la conjecture P≠NP. On va cependant pouvoir classer les problèmes entre eux, grâce à la notion de complé-

tude. Intuitivement, les problèmes NP-complets sont les plus difficiles des problèmes de NP : si l'on savait en résoudre, ne serait-ce qu'un seul, en temps polynomial alors on saurait les résoudre tous en temps polynomial (la complétude est basée sur la notion de réduction polynomiale, non introduite ici par soucis de place).

5.3.2 D'un point de vue pratique

Jusqu'à la fin des années 1990, le problème SAT pouvait être vu comme un moyen de motiver l'impossibilité pratique de résoudre un problème. Si vous aviez entre les mains un nouveau problème, vous le réduisiez à SAT pour démontrer qu'il n'y a que peu d'espoirs de pouvoir le résoudre en pratique. Les seules brèches étaient souvent basées sur l'identification de sous-classes polynomiales, comme les clauses de Horn [2], permettant de garantir une résolution efficace du problème. Attaquer de front le problème SAT est cependant longtemps resté extrêmement difficile en pratique.

Pour étudier la performance pratique des solveurs SAT, il a été proposé, fort classiquement, de construire des problèmes artificiels, générés pour être accessibles aux technologies de l'époque. Ainsi sont nés les problèmes aléatoires, au début des années 1990 [Mitchell *et al.*, 1992]. Leur utilisation dans le but de progresser pour la résolution de problèmes réels n'a cependant duré qu'un temps, jusqu'à ce que soit mis à jour certains phénomènes insoupçonnés de ces instances [Monasson *et al.*, 1998] (voir section 5.4.1). À la fin des années 1990, en effet, la séparation entre solveurs dédiés aux formules aléatoires et ceux dédiés aux formules industrielles (ou issus du monde réel) était consommé [Le Berre *et al.*, 2009 ; LeBerre et Simon, 2006].

Puis, en 2001, conjointement avec l'apparition de ce que l'on appelle les « solveurs modernes », est apparu la notion de *Bounded Model Checking* [Biere *et al.*, 1999 ; Prasad *et al.*, 2005] qui permet, en déroulant un certain nombre d'étapes, de transformer en SAT des problèmes de logiques temporelles, ou encore d'atteignabilité d'automates. Lorsque ces automates représentent le fonctionnement d'un microprocesseur, un état particulier, *faute*, est généralement représenté, et l'existence d'un chemin entre l'état initial et la faute représente un exemple de mauvais fonctionnement. Cette longueur de chemin doit être bornée (*Bounded*) pour garantir la transformation en logique propositionnelle. Il est aussi possible de tester l'équivalence de deux circuits (le circuit de référence et le circuit optimisé, effectivement implanté dans un CPU) grâce à SAT, en construisant une formule dont la satisfiabilité implique l'équivalence des deux circuits. On va trouver aussi des moteurs SAT dans l'état de l'art la résolution de problèmes important de bioinformatiques [Choi *et al.*, 2008], de Cryptographie [Le Berre *et al.*, 2009] ou encore au cœur des méthodes les plus performantes en planification (voir chapitre II.9).

Aujourd'hui, de plus en plus de problèmes *équivalents* à SAT, c'est-à-dire pouvant être réduit efficacement à lui, sont exprimés sous forme de logique propositionnelle, dans le simple but de trouver une solution de manière efficace. Cette réduction à SAT permet souvent d'obtenir des performances pratiques supérieures aux méthodes *ad hoc* proposées dans les différents domaines dont font parti chacun des problèmes. Ainsi, le

2. Une clause de Horn est une clause contenant au plus un littéral positif. Une formule de Horn est une formule sous $\mathcal{FNC}$ ne contenant que des clauses de Horn

principe du BMC a pu être étendu avec succès aux logiques temporelle dans [Armoni *et al.*, 2005]. Parfois, le moteur SAT est caché – mais essentiel – dans des démonstrateurs au pouvoir d'expression étendu, comme dans les démonstrateurs SMT (Satisfiabilité Modulo Theory [Shostak, 1979 ; Giunchiglia et Sebastiani, 1996]). Ce formalisme enrichit en effet l'expressivité de la logique propositionnelle en ajoutant des atomes de théories classiques (comme la *logique des différences*, la *théorie de l'arithmétique sur les rationnels* ou les *vecteurs de bits*, très utilisée en vérification formelle). Les clauses peuvent dès lors être de la forme $\neg x \vee (v_3 = v_1 - 14) \vee lire(s, b - c) = d$, et c'est au rôle du module de raisonnement sur la théorie de vérifier que la conjonction des atomes, fournis comme modèles par le démonstrateur SAT, est bien cohérente avec la théorie.

5.4 Le pragmatisme à l'assaut du test de satisfaisabilité

Depuis un vingtaine d'année, et la première compétition DIMACS en 1993 [Dimacs, 1993], une grande partie de la communauté SAT se mobilise pour tenter de résoudre, en pratique, ce problème en théorie difficile. Les progrès ont été fulgurants, grâce notamment à l'introduction d'un nouveau paradigme de calcul, basé sur l'apprentissage massif lié aux conflits passés. De manière moins applicative, les progrès dans la connaissance des instances aléatoires ont tout autant été fulgurants. Ces progrès théoriques et pratiques, motivés par des compétitions annuelles [Le Berre *et al.*, 2009], sont certainement l'une des évolutions majeures de l'intelligence artificielle (IA) de ces dernières années.

5.4.1 Algorithmes incomplets

Lorsque l'on parle d'algorithmes incomplets pour SAT, on évoque en général les algorithmes basés sur la recherche locale stochastique (RLS) [Saïs, 2008 ; Hoos et Stützle, 2004]. Leur incomplétude vient de leur incapacité à prouver UNSAT (on verra que certains algorithmes récents permettent une recherche locale stochastique autorisant les deux réponses SAT et UNSAT). Utilisés depuis les années 1970 [Kernighan et Lin, 1970] principalement dans les problèmes d'optimisations, leur utilisation au problème SAT a donné lieu à des résultats particulièrement fascinants, dont certains en relation avec la physique théorique, mais aussi, très récemment, en permettant de prédire avec une grande précision leur temps de résolution médian sur la classe des problèmes aléatoires.

Comme on le voit dans le schéma de l'algorithme 5.1, leur principe est d'une apparente simplicité : partant d'une première interprétation *totale*, généralement choisie au hasard, ils effectuent des mouvements locaux (rapides) permettant de se rapprocher d'une solution. Pour mesurer l'éloignement actuel de la solution globale, la méthode la plus simple (et efficace dans bien des cas) consiste à simplement compter le nombre de clauses falsifiées (comme dans les solveurs historiques GSAT [Selman *et al.*, 1992] et WALKSAT [Selman *et al.*, 1995]). Bien entendu, face à cette simplicité apparente, une grande partie de l'efficacité de ces méthodes reste cachée dans les détails : leur premier problème vient de leur sensibilité au paramétrage, ainsi qu'à leur façon de répondre aux questions « comment définir un bon voisin permettant à la fois de se

Algorithme 5.1 : Schéma algorithmique de la Recherche Locale Stochastique

```
Générer une Interprétation Initiale I ;
tant que Critère d'arrêt non respecté faire
    Choisir un voisin v de l'interprétation I ;
    si v est meilleur que I alors I := v ;
    ;
Retourner I ;
```

rapprocher de la solution mais aussi de s'échapper de minima locaux ? » ; « Comment choisir les voisins (toutes les interprétations à distance de Hamming de 1 ou uniquement une restriction de ceux-ci ?) ». De plus, dans cette description, l'algorithme part d'une seule interprétation initiale, ce qui n'est généralement pas le cas. Si aucune solution n'est trouvée après un nombre fixé de mouvements (*flips*), l'algorithme repart à partir d'une nouvelle interprétation tirée au hasard (il ne s'agit pas forcément de véritables redémarrages (*restarts*) dans la mesure où certaines approches peuvent préserver des compteurs heuristiques d'un redémarrage à l'autre (comme le poids de chaque clause, mis à jour selon la difficulté de trouver une interprétation qui la satisfasse lors de la recherche [Selman et Kautz, 1993])).

On notera aussi combien cette approche semble bien adaptée aux problèmes d'optimisations, en offrant un moyen simple de s'approcher de la solution, quelque soit la difficulté du problème. Très classiquement, RLS est ainsi capable de renvoyer un modèle représentant l'affectation la plus proche de la solution globale trouvée. Cela nous rapproche du problème connexe MAXSAT, qui cherche à maximiser le nombre de clauses satisfaites dans un problème UNSAT. Ce problème, qui a aussi ses compétitions annuelles, a des intérêts pratiques importants en IA et en vérification formelle (voir [Safarpour *et al.*, 2007] pour un exemple d'applications *industrielles*). Dans ce domaine aussi les progrès ont été fulgurants, et semblent encore s'accélérer récemment [3]. Sur la plupart de ces exemples *structurés*, les solveurs à base de méthodes complètes (*SAT Incrémental*) semblent cependant donner de meilleurs résultats.

Quelques détails sur la RLS

Si les algorithmes de RLS travaillent sur des interprétation totales, on notera que l'incorporation des mécanismes de propagation unitaire y est possible [Hirsch et Kojevnikov, 2002]. Des propositions ont aussi été faites pour utiliser les principes des algorithmes génétiques [Lardeux *et al.*, 2006]. Nous focalisons cependant cette section sur GSAT et WALKSAT, archétypes de RLS pour SAT. Dans ces solveurs, la fonction de voisinage est généralement une restriction des interprétations ayant une distance de Hamming de 1 (un seul littéral est inversé à chaque mouvement). GSAT a cependant ce défaut d'être particulièrement agressif (il ne choisit que parmi les voisins améliorant le score actuel), ce qui l'entraîne assez facilement dans des minima locaux. En ajoutant la possibilité de mouvements non optimaux, WALKSAT est, de manière surprenante au

3. voir le site des évaluation MAXSAT, `maxsat.ia.udl.cat`

vu de sa simplicité, l'une des meilleurs méthodes pour les instances aléatoires satisfaisables. Son principe est de choisir une clause aléatoirement parmi les clauses falsifiées et d'échanger la valeur d'un de ces littéraux pour la rendre vraie. Deux cas sont pris en compte. S'il existe des littéraux pouvant être inversés sans falsifier une clause anciennement vraie, alors c'est l'un de ces littéraux qui est choisi. Sinon, le littéral choisi sera, avec une probabilité b (b pour *bruit*) n'importe quel littéral de la clause, ou avec probabilité $(1-b)$ le littéral minimisant le nombre de nouvelles clauses falsifiées. Dans les premières versions de WALKSAT, $b = 0.5$ était proposé. Comme on le voit dans la suite, une erreur de quelques centièmes dans la valeur de b peut donner des comportements totalement différents.

Dans l'éventail des méthodes améliorant les performances des RLS, on se doit de citer la recherche Tabou [Mazure *et al.*, 97] qui empêche localement le solveur de réessayer une interprétation déjà vue (généralement la mémoire peut être limitée aux 6 dernières interprétations). Cette idée a aussi donné lieu aux méthodes Novelty [Tompkins et Hoos, 2004], privilégiant statistiquement les variables les plus anciennes pour éviter de modifier des variables qui viennent d'être touchées. Ces différents mécanismes doivent permettre de parcourir des plateaux ou s'échapper de minima locaux.

Malgré tout, même si la recherche locale est primordiale dans nombre de problèmes d'optimisations, elle a du mal à lutter contre les algorithmes complets lorsqu'il s'agit de quitter le giron des instances aléatoires uniformes (où elle est d'une efficacité redoutable), ou de certaines classes d'instances très régulières, aux paysages doux.

Recherches locales stochastiques sur les problèmes *aléatoires*

Lorsque les problèmes aléatoires ont été proposés, les premières expérimentations laissaient supposer que cette classe de problèmes était solvable, en moyenne, en temps polynomial (quadratique) par DPLL [Goldberg *et al.*, 1982], introduit dans la suite. Ce résultat fut rapidement contesté, notamment lorsque [Franco, 1986] montra que des tirages au sort aléatoires des interprétations donnaient de meilleurs résultats *en moyenne* que DPLL ! Les instances intéressantes étaient en quelque sorte *noyées* au milieu des instances faciles. Il ne suffisait pas de générer des instances aléatoires uniformes en fixant m le nombre de clauses et n le nombre de variables, en les autorisant à prendre toutes les valeurs possibles, mais il fallait savoir à quel endroit cette famille d'instance n'était pas *triviale*. Ce pic de difficulté fut identifié expérimentalement dans [Kautz et Selman, 1996], et relié à un phénomène particulièrement fascinant : la transition de phase de la satisfaisabilité des formules générées en considérant le rapport $r = m/n$. Ainsi, il a été démontré (expérimentalement seulement ; il s'agit en réalité toujours d'une conjecture forte) l'existence d'un seuil s tel que pour $r \leq s$ les formules sont satisfaisables avec une grande probabilité et pour $r > s$ elles sont insatisfaisables (sous les mêmes conditions). Nous ne pouvons que conseiller la lecture du chapitre sur la transition de phase de [Saïs, 2008], qui explique en profondeur les problématiques d'un des phénomènes les plus fascinants lié à la logique propositionnelle.

Le seuil a été montré expérimentalement à $r_s = 4.267$ (alors que les meilleurs bornes prouvées sont respectivement 3.53 et 4.508 [Achlioptas et Ricci-Tersenghi, 2010]). En fait, en dessous de $r_1 = 3.003$ les instances sont triviales (la règle de Generalized Unit Clause (GUC) permet de résoudre presque à coup sûr toutes les instances). Entre r_1 et

$r_2 = 3.9$ une approche gloutonne de type GSAT fonctionne très bien. Ensuite, entre r_2 et r_s la dispersion des instances change soudainement. Intuitivement, on va trouver un nombre exponentiellement grand d'îlots de solutions, mais dont la distance les séparant deux à deux sera très grande. Dans cette zone, on parle de variables « gelées », notion issue de celle des *backbones*, exploitée dans KCNFS [Dubois et Dequen, 2001] et SP (survey propagation, [Monasson *et al.*, 1998; Braunstein *et al.*, 2002; Kroc *et al.*, 2007]). Il s'agit intuitivement de variables prenant la même valeur dans tous ces îlots de solutions. La description de SP dépasse le cadre de cet ouvrage, mais son importance nécessite tout de même d'en toucher deux mots. Cet algorithme est issu de méthodes employées par les physiciens pour étudier les propriétés magnétiques des verres de spin. En pratique, il s'agit de calculer le *biais* de chaque variable, c'est-à-dire la valeur de la variable statistiquement préférée par l'ensemble de la formule, à l'aide d'une méthode par passage de messages, proche de la Belief Propagation [Pearl, 1982]. Lorsque l'on applique cette méthode aux formules aléatoires uniformes très proches du seuil (jusque $r = 4.25$), SP arrive à résoudre des instances de 10^6 variables, en identifiant justement les variables gelées de la formule initiale. De plus, il semble que SP soit d'autant plus efficace que la formule est grande (ce qui peut paraître en totale contradiction avec l'idée même de NP-complétude). L'étude de SP, et son applicabilité à d'autres types de formules, est un domaine très actif.

D'autres résultats sont tout aussi fascinants [Kroc *et al.*, 2010]. Ainsi, malgré l'ancienneté de WALKSAT, des développements récent le remettent sur le devant de la scène. S'il est connu depuis longtemps que le paramétrage du bruit b de cet algorithme est délicat, une relation simple entre b et r vient d'être mise en évidence pour obtenir des algorithmes efficaces. Le plus étonnant est que WALKSAT montre un comportement linéaire en temps jusqu'une certaine limite très proche du seuil (peu après 4.235 WALKSAT ne semble plus être linéaire sur les instances aléatoires, même avec un bruit optimal). À titre d'exemple, WALKSAT résoud assez facilement une formule avec 100 000 variable à $r = 4.20$, avec un bruit $b = 0.57$. Si on le paramètre avec $b = 0.58$ ou $b = 0.50$ alors les performances se dégradent énormément. Le bruit *optimum* à associer à WALKSAT suit 3 lois linéaires, chacune définie justement en trois zones découpées suivant le régime des instances aléatoires (suivant que r soit inférieur à r_1, r_2 et r_s respectivement). On notera aussi que certains phénomènes statistiques n'apparaissent que pour $kSAT$ avec $k > 3$.

Notons tout de même, pour finir cette section sur les instances aléatoires, leur importance pratique. Il a été en effet montré un lien étroit entre la preuve de la difficulté moyenne de 3SAT aléatoire et la difficulté d'approximer certains problèmes NP-difficiles, qui échappent encore à toute preuve de difficulté d'approximation [Feige, 2002]. Si on arrive donc à résoudre efficacement 3SAT aléatoire *en moyenne*, il doit être possible d'approximer efficacement certains problèmes qui nous échappent encore.

Challenges pour les méthodes incomplètes

Sans nul doute, la RLS a encore de beaux jours devant elle, même sur le problème de décision SAT, où l'état de l'art reste dominé par les méthodes complètes (voir ci-dessous, section 5.4.2). Ainsi, il reste toujours difficile de l'appliquer en pratique sur des exemples industriels, malgré le challenge 6 du célèbre article de B. Selman, H. Kautz and D.

McAllester [1997], pourtant posé il y a plus de 15 ans : « Improve stochastic local search on structured problems by efficiently handling variable dependencies » (ces challenges ont été mis à jour 6 ans plus tard dans [Kautz et Selman, 2003]). L'autre challenge important de cet article, le challenge 5 demandant à des algorithmes de recherche locale de prouver UNSAT, n'a pas encore de réponse claire à ce jour, même si quelques brèches ont toutefois été ouvertes. On pourra citer de manière assez anecdotique [Audemard et Simon, 2007] et, de manière plus performante, l'approche CDLS [Audemard *et al.*, 2009] qui propose une technique permettant de profiter de l'affectation totale de la RLS pour en déduire un conflit *expliquant* l'échec de cette affectation. La clause est ensuite apprise pour *échapper* de ce minimum local. Il existe aussi d'autres approches, plus anciennes, dont le principe est d'hybrider approches complètes et incomplètes [Mazure *et al.*, 1998 ; Habet *et al.*, 2002], ou encore de retrouver la structure de circuit de l'instance encodée en $\mathcal{FNC}$ pour y effectuer la recherche (voir [Pham *et al.*, 2007], basé sur le travail de [Ostrowski *et al.*, 2002]). Aucune de ces approches n'a pour l'instant remis en cause la supériorité des approches complètes sur les instances issues du monde réel.

5.4.2 Algorithmes complets, ou systématiques

À la différence de la très grande majorité des RLS, les approches complètes, ou systématiques, permettent de prouver SAT ou UNSAT. Il est d'ailleurs intéressant de noter que, dans les approches les plus récentes spécialisées dans les problèmes industriels, il n'y a pas de solveurs spécialisés dans SAT ou dans UNSAT uniquement.

DP60 : un raisonnement trop puissant pour SAT

Dans sa première version [1960], l'algorithme de Davis et Putnam fonctionnait par « oublis » successifs des variables de la formule. En effet, si on prend la décomposition de Shannon, on sait que toute formule f peut se réécrire en $f' \equiv (x \wedge f|_{\{x\}}) \vee (\neg x \wedge f|_{\{\neg x\}})$ pour tout $x \in Var(f)$. La formule $f|_{\{x\}}$ (resp. $f|_{\{\neg x\}}$) étant f dans laquelle toutes les occurrences de x ont été remplacées par *vrai* (resp. $faux$). La formule obtenue f' n'a plus aucune occurrence de x, mais préserve la satisfaisabilité de f. La difficulté pratique de DP vient de la réécriture de f' sous $\mathcal{FNC}$, celle-ci faisant intervenir la distribution des deux ensembles de clauses obtenus l'un sur l'autre. Pour prouver la satisfaisabilité (ou plutôt la non satisfaisabilité) il suffit d'éliminer toutes les variables de la formule, les unes après les autres. L'élimination (ou *oubli*) de x dans f peut se calculer simplement en considérant la formule $f|_{\{x\}} \vee f|_{\{\neg x\}}$ à la place de f. Si l'on n'obtient pas la clause vide, alors la formule est satisfaisable. DP, revisité dans [Dechter et Rish, 1994 ; Rish et Dechter, 2000 ; Simon, 2001a] a fait l'objet de beaucoup de travaux, de part ses liens avec des classes polynomiales estimées proches d'instances du monde réel (basée sur la *largeur induite* [Val, 2000]), mais aussi, plus récemment, pour traiter des problèmes au dessus de SAT, notamment des problèmes de *Compilations de Bases de Connaissances.* En effet, si l'on met de côté les clauses produites tout au long du calcul, il est possible, lorsque toutes les variables ont été éliminées, d'énumérer tous les modèles de la formule dans un temps proportionnel à leur nombre. On voit que DP répond de fait à un problème plus difficile que le *simple* problème de satisfaisabilité. À part pour quelques

Algorithme 5.2 : Procédure DPLL62

Initialisation : $\mathcal{F}$ une formule propositionnelle ;
Procédure DPLL($\mathcal{F}$) ;
si *il existe dans* $\mathcal{F}$ *un* ***littéral pur*** l **alors** Retourner DPLL($\mathcal{F}\backslash l$) ;
;
si *il existe dans* $\mathcal{F}$ *une* ***clause unitaire*** l **alors** Retourner DPLL($\mathcal{F}\backslash l$) ;
;
si $\mathcal{F}$ *contient au moins une* ***clause vide*** **alors** Retourner($Faux$) ;
;
si $\mathcal{F}$ ***est vide*** **alors** Retourner($Vrai$) ;
;
$v \leftarrow$ une variable de $\mathcal{F}$ **(Heuristique de choix pour la division)** ;
si DPLL($\mathcal{F}|_{\{v\}}$) **alors**
| Retourner($Vrai$)
sinon
| Retourner(DPLL($\mathcal{F}|_{\{\neg v\}}$))

problèmes spécifiques de faible largeur induite, DP-60 n'a jamais vraiment pu rivaliser avec la version de 62, basée sur un parcours systématique des modèles potentiels de la formule, avec retours arrières en cas d'échecs. On en retrouvera toutefois une version limitée – mais indispensable aux approches *modernes* –, dans les prétraitements des formules, section 5.4.3.

DPLL62 : l'anticipation doit limiter les retours arrières

Plutôt que de réécrire la formule sous $\mathcal{FNC}$ après chaque élimination de variable, et de tenter de lutter contre l'explosion combinatoire en mémoire comme dans DP60, il a été proposé dans DPLL62 [1962] de remplacer la disjonction dans f' (voir ci dessus) en une alternative de choix, une *division* de l'espace de recherche, en vérifiant d'abord la satisfaisabilité de $f|_{\{x\}}$ puis, si le résultat est négatif, en vérifiant la satisfaisabilité de $f|_{\{\neg x\}}$. Cela a naturellement donné lieu à une définition sous forme de retours arrières chronologiques, comme illustré par l'algorithme 5.2. Une liste non exhaustive de quelques solveurs DPLL62 marquants est par exemple SATZ [Li et Anbulagan, 1997], KCNFS [Dubois et Dequen, 2001], SATO [Zhang, 1997] et GRASP [Silva et Sakallah, 1996].

Dans cet algorithme, plusieurs règles sont à l'œuvre. On notera que la règle du *littéral pur* (une variable n'apparaissant que positivement ou négativement dans la formule) n'est pas toujours incorporée dans les solveurs, car souvent jugée non rentable, expérimentalement. Par contre, la détection de *clauses unitaires* est un des éléments fondamentaux de tout solveur SAT (y compris pour les solveurs SAT « modernes », dont c'est l'un des éléments clés des solveurs : sur une machine récente, et sur un problème industriel typique, on peut mesurer plus d'un million de ces détections par seconde)). L'autre élément primordial est la fonction heuristique de choix, et une grande partie de l'efficacité de ces solveurs ne repose que sur celle-ci. Ainsi, une part très importante

du calcul peut être facilement donnée à la mise à jour et au calcul de cette fonction.

De nombreuses heuristiques ont été proposées pour tenter de limiter la taille de l'arbre de recherche. L'heuristique la plus connue, appelée MOMS (pour Maximum number of Occurences in Minimum Size clauses) fut introduite dans [Davis *et al.*, 1962; Goldberg, 1979]. Cette heuristique privilégie la variable ayant le plus grand nombre d'occurrences dans les clauses les plus courtes. Dans bien des cas, cependant, elle risque de ne simplifier qu'une seule des deux branches de la recherche. Pour équilibrer les deux sous-arbres, l'heuristique de Jeroslow's & Wang [1990] estime la variable x à l'aide de la formule $\alpha \times m(x) \times m(\neg x) + m(x) + m(\neg x) + 1$ où $m(x)$ est une mesure de représentativité de x et α une constante (l'heuristique BOHM [Buro et Büning, 1993], souvent utilisée, est un rafinement de MOMS avec cette même idée d'équilibrage des sous-arbres).

Déjà, une scission entre les heuristiques dédiées aux instances aléatoires et aux instances issues du monde réel est apparu. Ainsi, [Li et Anbulagan, 1997] propose de privilégier la variable permettant un grand nombre de propagations unitaires, en cascades (recherchant ainsi le nombre maximum d'*avalanches* [Audemard *et al.*, 1999]), après son affectation, dans les deux sous-arbres. Il s'agit là d'une heuristique à forte anticipation (*lookahead*), qui montre bien combien l'effort de recherche était équilibré entre le calcul de l'heuristique lui-même et le parcours effectif de l'arbre de recherche. Poussée plus loin, [Heule et Van Maaren, 2007] a proposé une heuristique de double anticipation qui a donné de bons résultats sur les instances aléatoires. Sur celles-ci, cependant, la meilleur heuristique est BSH [Dubois et Dequen, 2001]. Elle cible la recherche des variables gelées des instances aléatoires, en mesurant la « corrélation » des variables entre elles. Sa description complète dépasse néanmoins le cadre de ce chapitre.

CDCL : des solveurs « modernes »

En privilégiant le calcul heuristique, les solveurs DPLL, décrits ci-dessus, doivent constamment (après chaque affectation et chaque libération de littéral) mettre à jour de nombreux compteurs. En cherchant à alléger ces mécanismes [Moskewicz *et al.*, 2001] a proposé une structure de données étendant l'idée de [Zhang, 1997] et permettant une détection paresseuse des clauses unitaires, relativement à une affectation. Cette structure, appelée *Watched Literals* dans [Moskewicz *et al.*, 2001] a révolutionné l'application de SAT aux instances industrielles, en autorisant le traitement de problèmes avec plusieurs centaines de milliers de variables. Cette structure de donnée doit cependant céder une contrepartie de première importance : il n'existe plus aucun moyen de connaître, durant la recherche, le nombre de clauses satisfaites, ou le nombre de clauses binaires (ou même unitaires) relativement à l'affectation courante. Pour pouvoir proposer une heuristique dans ces conditions, les solveurs ont troqué de la visibilité contre de la mémoire. Fonctionnant à l'aveugle, l'intégralité du solveur est passé d'un solveur par anticipation (DPLL62) à un solveur basé sur l'apprentissage, grâce à une analyse précise de son passé, et des conflits rencontrés. Aujourd'hui ces mécanismes sont particulièrement bien cernés [Eén et Sörenson, 2003], et peuvent se résumer à 500 lignes de code [Huang, 2007] où toutes les techniques utilisées sont profondément interdépendantes. Heuristiques, redémarrages ultra-rapides, sauts arrières non chrono-

Algorithme 5.3 : Formulation itérative de DPLL : une formulation CDCL (Conflict-Driven Clause Learning), pleinement tournée vers l'apprentissage

$\mathcal{I} = \varnothing$, $profondeur = 0$;
tant que *Vrai* **faire**
  Effectuer la PU sur $(\Sigma, \mathcal{I})$;
  si *un conflit est apparu* **alors**
    si $profondeur = 0$ **alors** Retourner UNSAT ;
    ;
    C = la clause Conflit déduite ;
    l = l'unique littéral de C affecté à la profondeur du conflit ;
    $profondeur = max\{profondeur(x) : x \in C/\{l\}\}$;
    $\mathcal{I} = \mathcal{I}$ moins tous les littéraux assignés à une profondeur supérieure à $profondeur$;
    $(\Sigma, \mathcal{I}) = (\Sigma \cup \{C\}, \mathcal{I}.l)$;
  sinon
    si $\mathcal{I}$ *est total* **alors** Retourner SAT ;
    ;
    Choisir un littéral l apparaissant dans $\Sigma|\mathcal{I}$;
    $\mathcal{I} = \mathcal{I}.l$;
    $profondeur = profondeur + 1$

logiques, propagations unitaires, gestion des clauses utiles à la suite de la recherche et bien entendu structures de données sont entièrement dédiés à l'apprentissage, et offrent souvent l'image d'un algorithme de plus en plus éloigné de la recherche arborescente binaire classique.

Comme on le voit en lisant la description de l'algorithme 5.3, le solveur essaye d'atteindre rapidement un conflit (une clause vide), puis apprend une clause, appelée clause assertive (ou clause FUIP), permettant de calculer par ailleurs le niveau de retour arrière nécessaire. L'heuristique privilégiera les variables vues dans les analyses de conflits récentes (en pratique, toutes les variables vues verront leur score incrémenté d'une valeur, qui elle-même croît exponentiellement après chaque conflit, en suivant une suite géométrique). L'analyse de conflit et la notion de FUIP (pour *First Unique Implication Point* est primordiale (elle a été montrée optimale dans la taille des sauts arrières [Audemard *et al.*, 2008] et dans la qualité des clauses apprises [Audemard et Simon, 2009]).

Sur l'exemple de la figure 1, la base de clause initiale contient les clauses suivantes, devenue unitaires par rapport à l'affectation courante (en parenthèses apparaît le niveau de décision auquel elles sont détectées comme étant unitaires) : $x_1 \vee x_2$ et $\neg x_2 \vee \neg x_3$ (niveau 1), $\neg x_4 \vee \neg x_2 \vee \neg x_5$ et $x_5 \vee x_3 \vee x_6$ (niveau 2), $x_7 \vee \neg x_6 \vee \neg x_8$ et $x_8 \vee \neg x_4 \vee x_9$ (niveau 3), ainsi que, $x_{10} \vee \neg x_9 \vee x_{11}$, $\neg x_1 \vee x_8 \vee \neg x_{12}$, $x_{12} \vee \neg x_{13}$, $x_{12} \vee x_{14}$, $\neg x_6 \vee x_{12} \vee x_{15}$, $x_{13} \vee \neg x_{14} \vee \neg x_{16}$, et $\neg x_{14} \vee x_{15} \vee x_{16}$ pour le niveau 4. Sur l'exemple donné, on suppose que les différentes variables de division (ou décision) sont, dans l'ordre, le littéral $\neg x_1$, x_4, $\neg x_7$ et enfin $\neg x_{10}$. Le conflit apparaît lors de la propagation unitaire

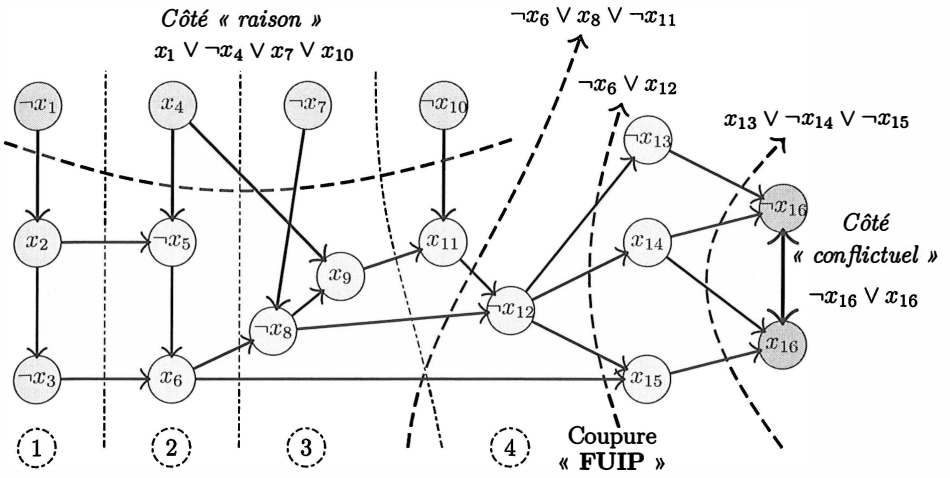


FIGURE 1 – Exemple de graphe d'implication associé aux différents schémas d'apprentissage possibles.

du quatrième niveau de décision. On représente sur la figure les différentes coupes permettant d'expliquer le conflit, séparant le côté « raison » (les décisions) du côté « conflictuel » (la contradiction). Plusieurs coupures sont possibles, mais une seule contient un littéral FUIP, c'est la *clause assertive* : elle ne contient qu'un seul littéral affecté au dernier niveau de décision (il existe toujours un FUIP, dans la mesure où l'on peut remonter depuis le conflit jusque la dernière variable de décision). On remarquera que dans la clause assertive $\neg x_6 \vee x_{12}$, le troisième niveau de décision n'apparaît plus. Il suffit donc de faire un retour arrière directement au niveau 2, où l'on pourra propager le littéral x_{12}, grâce à la clause assertive nouvellement apprise, devenue unitaire. Le graphe du conflit est un graphe dirigé sans cycles. L'analyse du conflit revient à remonter, en largeur d'abord, depuis le conflit jusqu'à ce que la coupure du graphe calculée ne contienne plus qu'un littéral du dernier niveau de décision. La mise à jour des valeurs heuristiques est effectuée lors de cette phase d'analyse, en augmentant d'une valeur b toutes les variables vues pendant l'analyse (toutes les variables *à droite* de la coupure). En pratique, la valeur de b est multipliée par une constante, pour se focaliser sur les variables vues dans les derniers conflits (b est en général multiplié par 1.05 à chaque conflit).

De manière théorique, il est intéressant de noter que le parcours en largeur du graphe des conflits revient en fait à effectuer des résolutions entre les clauses responsables de la propagation unitaire des variables du graphe, depuis la clause conflictuelle jusqu'à l'obtention de la clause assertive. Cela a permis à Darwiche [2009] de montrer que les CDCL ont la puissance des systèmes de preuve basés sur la résolution générale. Si, de plus, on ajoute que les solveurs DPLL sont eux basés sur la résolution régulière, on a là des outils théoriques montrant que la puissance des CDCL dépasse celle des DPLL. En effet, il existe dès lors des problèmes difficiles pour les DPLL et faciles pour les CDCL

(l'inverse n'est pas vrai).

On ne peut terminer cette courte introduction aux solveurs modernes sans évoquer quelques-uns de leur composants essentiels, comme les redémarrages rapides [Huang, 2007 ; Luby *et al.*, 1993] (qui ne sont pas au sens propre des redémarrages, mais plutôt un réordonnancement des dépendances entre variables, puisque les valeurs heuristiques sont conservées : le solveur demeure dans le même espace de recheche), la sauvegarde de phase [Pipatsrisawat et Darwiche, 2007], qui permet de privilégier la dernière polarité des variables lors du branchement, ainsi que la gestion agressive de la base de clauses apprises [Audemard et Simon, 2009]. Enfin, il faut bien noter, pour conclure sur les solveurs modernes, que beaucoup de choses restent à comprendre dans leurs mécanismes, et que l'on peut encore s'attendre à de nouveaux progrès importants à ce sujet dans les prochaines années.

5.4.3 Prétraitement des formules

De manière à combler, en partie, la lacune des solveurs modernes, incapables par exemple de détecter une variable pure, ou des équivalences de littéraux, on leur adjoint quasi systématiquement des techniques de prétraitement. Lors de la compétition SAT 2005, le seul système de prétraitement SATELITE [Eén et Biere, 2005] avait ainsi pu résoudre un grand nombre des problèmes industriels proposés. Ces prétraitements sont généralement des applications successives de règles d'oubli de variables (garantissant une diminution de la taille de la formule), de résolution hyper-binaire [Bacchus et Winter, 2003], ainsi que de raisonnement sur les clauses bloquées [Järvisalo *et al.*, 2010]. Même s'ils présentent une *garantie* de rapidité dans le temps maximal passé dans la simplification, leur intégration dans les solveurs CDCL pose problème, dans la mesure où le moindre calcul quadratique dans le nombre de variables serait voué irrémédiablement à l'échec, du fait de la taille parfois gigantesque de certains problèmes. L'élimination des symétries a aussi fait l'objet de nombreux travaux (voir [Saïs, 2008], chapitre 6), mais ce n'est que très récemment que les performances dans la détection des symétries a permis d'espérer embarquer ces procédures dans les solveurs modernes, nécessitant de pouvoir manipuler des instances avec des millions de clauses [Katebi *et al.*, 2010].

5.4.4 Méthodes exotiques

De fait de son importance, le problème SAT a été attaqué par différents formalismes, plus ou moins bien adaptés aux problèmes industriels contemporains. Ainsi, on se doit de citer la méthode de Stålmarck [1994], qui revient à appliquer des techniques d'anticipations associées à la propagation unitaire et au raisonnements d'équivalences. Avant les années 2000, les ROBDD (pour *Reduced Ordered Binary Decision Diagram*) [Bryant, 1986] étaient aussi une technique efficace sur certains types de problèmes très structurés, mais leur coût de construction est vite devenu prohibitif sur les instances de grande taille, et on les retrouve plutôt dans des problématiques de *compilations de bases de connaissances*, section suivante, ou lorsque la représentation, même implicite, de l'ensemble des modèles, est nécessaire. On notera aussi que la technique basée sur les points de référence, si elle est connue depuis quelques années, arrive enfin à rivaliser avec

les meilleurs solveurs CDCL actuels [Kottler, 2010]. Ce dernier résultat est d'autant plus intéressant que cela montre que, si les CDCL dominent encore très largement le domaine, cette suprématie commence à se fissurer, avec l'arrivée de méthodes presque aussi performantes, et basées sur des paradigmes différents.

5.4.5 Limitations et challenges pour SAT

SAT est un problème qui demande un usage intensif de la mémoire, sujet aux limites des « *memory bound functions* ». Ces fonctions ne permettent pas de suivre les progrès de la loi de Moore sur la rapidité des processeurs, et limitent grandement les possibilités de parallélisation efficace, notamment lors de l'utilisation de plusieurs cœurs. Intuitivement, ces fonctions ont un temps de calcul dominé par le temps passé à lire et/ou écrire en mémoire. Ces algorithmes parcourent de grandes portions de mémoire de manière imprévisible, empêchant tout mécanisme de cache d'être réellement efficace, et limitant la décomposition du problème en plusieurs endroits distincts de la mémoire. Dans ce cadre, et malgré d'important progrès (voir [Saïs, 2008 ; Hamadi *et al.*, 2009]), la difficulté reste importante et la parallélisation efficace des CDCL représente certainement l'un des plus importants défis pour la communauté SAT toute entière.

5.5 Compilation de bases de connaissances

Lorsqu'il s'agit d'exprimer de la connaissance, et de raisonner sur elle, le langage propositionnel peut paraître de prime abord trop restrictif. Il est ainsi relativement complexe d'exprimer des informations numériques, des relations d'ordre, ou des calculs combinatoires, même simples. Cependant, dans bien des cas, elle pourra être un partenaire de choix lorsque cette connaissance le permet, ou que le prix d'une propositionalisation n'est pas trop cher payé. De plus, les récents progrès en SAT permettent d'imaginer la manipulation de bases de connaissances de plusieurs centaines milliers de variables tout en espérant maintenir de bonnes performances. Parfois, la base de connaissance en logique propositionnelle sera construite sur une *abstraction* de la vraie base, permettant un raisonnement efficace, mais de haut niveau.

En se plaçant à un niveau général, on peut définir la compilation de bases de connaissance (CBC) comme le fait d'expliciter certaines informations contenues dans une base de connaissance (BC) afin d'en accélérer l'accès dans une phase ultérieure, voire de *garantir* un temps de réponse *raisonnable.* Dans le cadre de base de connaissances propositionnelles, l'information recherchée peut souvent se caractériser par un sous-ensemble des théorèmes que l'on peut déduire de la base initiale, par exemple par ceux que l'on peut obtenir par résolution. Cependant, retrouver ne serait-ce que l'un de ces théorèmes est déjà un problème difficile en soit et, de plus, il peut en exister un nombre exponentiel. Lorsque l'on veut *expliciter* automatiquement toute l'information *utile* (pour une tâche fixée), se pose la question de définir formellement le langage *cible* dans lequel sont construits les théorèmes visés. Cette notion a donné lieu à diverses formalisations, comme le calcul de *champs de production* [Siegel, 1987], de *langage restreint* [del Val, 1999], de *P-impliqués* [Castell, 1997] ou d'*impliqués P-premiers* [Le Berre, 2000], pour n'en citer que quelques-uns. Le lecteur peut consulter par exemple

[Haton *et al.*, 1991 ; de Kleer, 1992 ; Cadoli et Donini, 1997 ; Marquis, 1999] pour un aperçu plus complet des différentes méthodes.

5.5.1 Principes de la CBC

Le terme « compilation » regroupe assez généralement tout ce qui transforme une base de connaissances BC en une théorie permettant d'en accélérer l'utilisation (requêtes / transformations / mise à jour). L'un de ses principaux préceptes est que *la base BC n'évolue pas (ou rarement) et que de nombreuses requêtes doivent être effectuées dessus.* Ainsi, le temps de compilation, qui peut être assez long [4] en pratique, est amorti par le gain de temps sur la totalité des requêtes effectuées. Nous nous plaçons donc dans un cas où le temps de réponse critique est dans la phase d'interrogation. Le temps de compilation, appelé parfois temps « hors-ligne », n'est que secondaire. La problématique de la mise à jour de la base compilée, par contre, peut également être pris en compte.

Plus formellement, le but consiste à questionner une base de connaissance BC à l'aide de questions r appartenant à un langage R. Ces requêtes sont posées à l'aide de la question $Q?(BC, r)$, dont le calcul peut être *difficile.* Le but est alors de transformer la base BC en une base BCC de manière à ce que l'on obtienne des réponses rapides, en utilisant BCC à la place de BC. Les questions sont maintenant posées par $Q?(BCC, r)$ (on doit bien entendu retrouver la même réponse $(Q?(BC, r) = Q?(BCC, r))$ et le langage de compilation cible, BCC doit garantir une réponse polynomiale pour toute question r de R. L'un des points importants de la définition précédente réside dans le fait que la structure de données (BCC) doit être de taille polynomiale par rapport à la base initiale BC, sans quoi la notion d'efficacité sur cette structure de données n'aurait pas de sens pour la compilation : on aurait certes des algorithmes polynômiaux, mais uniquement sur une structure de donnée pouvant être de taille exponentielle par rapport à la taille de départ. La compilation n'est cependant pas miraculeuse : si une tâche ou un problème donné est d'une certaine complexité, qu'on la traite directement ou par compilation, la complexité dans le pire des cas demeure.

5.5.2 Les « débuts » : les cas particuliers des impliqués et impliquants premiers

Les deux langages des impliqués et impliquants premiers ont longtemps été privilégiés pour les problématiques de compilation de BC, sans toutefois la nommer encore explicitement ainsi. Pour rappel, soient f_1 et f_2 deux formules, si tout modèle de f_1 est aussi modèle de f_2, ce qui se note $f_1 \models f_2$, alors f_1 est appelé *impliquant* de f_2 et f_2 est appelé *impliqué* de f_1. Les clauses minimales (vis-à-vis de l'inclusion ensembliste) impliquées par une formule sont appelées les *impliqués premiers.* De même, les produits minimaux impliquant une formule sont appelés *impliquants premiers.* On peut noter que l'ensemble des impliqués premiers est exprimé en $\mathcal{FNC}$ alors que l'ensemble des impliquants premiers s'exprime en $\mathcal{FND}$. Intuitivement, les impliqués premiers

4. Ce temps est souvent bien plus long que le temps d'une seule requête.

caractérisent tous les théorèmes minimaux que l'on peut déduire de la BC, et les impliquants premiers caractérisent tous les modèles de cette même BC. Les deux notions sont duales.

L'un des premiers algorithmes de production d'impliqués premiers est l'algorithme de Tison [Tison, 1967], très proche de l'algorithme DP (voir section 5.4.2), lui-même étant une optimisation de l'algorithme de Quine [Quine, 1955], qui permet de réduire le nombre de résolutions à effectuer. Dans [Quine, 1955], l'auteur propose d'effectuer toutes les résolutions possibles jusqu'à ce que plus aucune résolution ne puisse permettre de déduire de nouvelle clause. L'algorithme de Tison utilise une stratégie ordonnée. La transitivité de la résolution et de la sous-sommation permettent d'imposer un ordre sur les variables sur lesquelles les résolutions sont possibles sans perte de complétude. Une fois toutes les variables traitées, toutes les clauses possibles ont été déduites. Une méthode incrémentale du calcul des impliqués premiers a été proposée dans [Kean et Tsiknis, 1990 ; de Kleer, 1992], avec notamment, dans le second article, la volonté de l'appliquer aux ATMS (voir section 5.5.5).

Des optimisations ont été proposées, notamment lorsque le langage cible n'est qu'une partie de tous les théorèmes possibles. Dans sa thèse, D. Le Berre [2000] travaille sur un cas particulier dans lequel P est un ensemble consistant de littéraux. En effet, en pratique, le cas où les littéraux sont tous positifs peut caractériser des *nogoods* dans les *ATMS*, ou des variables représentant la défaillance d'un élément du système modélisé. Cette restriction permet d'améliorer grandement les performance et de bénéficier des progrès réalisés, à la fin des années 90, par les solveurs SAT de type DPLL.

La méthode proposée par O. Coudert et J.-C. Madre [Bryant, 1986 ; Madre et Coudert, 1991 ; Coudert et Madre, 1992] fait partie des approches ayant permis un changement d'échelle pratiquement sans précédent dans la taille des formules à traiter. Ils proposent en effet une méthode permettant de calculer plus de 10^{20} impliqués premiers, et résolvant des dizaines de problèmes ouverts. L'algorithme proposé fonctionne en deux phases. La première partie caractérise l'ensemble des impliquants premiers d'une formule f en calculant sa représentation par ROBDD, puis la seconde partie utilise les propriétés de la décomposition de Shannon, décomposition sous-jacente à la représentation par ROBDD pour en extraire l'ensemble des impliqués premiers, encodés soit par un ROBDD (avec une représentation par métasomme) soit par ZBDD. Dans sa thèse, L. Simon [2001b] propose de représenter la $\mathcal{FNC}$ sous forme de ZBDD durant tout le calcul, et d'effectuer les opérations de distributions (primordiales pour les algorithmes à base d'oubli) directement sur cette structure de donnée. Ce changement de paradigme (le coût des opérations sur les ZBDD est décorrélé du nombre de résolutions effectivement faites) a permis de résoudre une grande partie des problèmes classiques, jusque là pris comme références.

5.5.3 Approximation de bases de connaissances

Cette idée est la base de l'approche proposée par [Selman et Kautz, 1996] où l'approximation est faite par des formules de Horn, pour le test de satisfaisabilité. Ces formules permettent une résolution efficace (linéaire) par les mécanismes puissants de résolution unitaire. Malheureusement, toute théorie ne peut pas être compilée en une

théorie équivalente de Horn [5]. Les auteurs proposent donc d'encadrer la théorie initiale par deux théories de Horn, en caractérisant les formules de Horn inférieures et supérieures les plus proches (plus grand minorant (GLB) et plus petit majorant (LUB)). En ce sens, cette approche est une approche par approximation de la théorie initiale.

5.5.4 Abstraction de la problématique de compilation

Au début des années 2000, [Darwiche et Marquis, 2001] proposent une nouvelle vision de la CBC, en en abstrayant les problématiques. Ainsi, plutôt que de sélectionner le langage cible en fonction des habitudes du domaine, ou de sa lisibilité directe, ils proposent de le définir en fonction des opérations qu'il permet de supporter efficacement. Étant donnée une BC formalisée par une formule Σ dans $\mathcal{L}$, le langage $\mathcal{L}$ satisfait un certain nombre de propriétés s'il peut répondre à certaines questions sur Σ en temps polynomial par rapport à la taille de Σ. Ils définissent ainsi par exemple **CO** pour la consistance, **VA** pour la validité et **CE** pour le fait de vérifier $\Sigma \models C$ pour n'importe quelle clause C de $\mathcal{FNC}$. D'autres opérations sont bien entendu définies, comme par exemple le fait de savoir si un produit implique Σ (**IM**) ou si l'on peut compter le nombre de modèles de Σ (**CT**), ou les énumérer (**ME**) (voir [Darwiche et Marquis, 2001] pour une présentation complète).

Il faut aussi pouvoir définir des opérations de transformation des BC, pour permettre par exemple leur mise à jour, ou la conjonction de deux BC (par exemple en autorisant l'arrivée de nouvelles connaissances). Les auteurs définissent par exemple $\vee$ **C** (resp. $\wedge$ **C**), si, pour tout ensemble fini de formules $\Sigma_1, \ldots, \Sigma_n$ de $\mathcal{L}$, il existe un algorithme polynomial donnant la représentation de $\Sigma_1 \vee \cdots \vee \Sigma_n$ (resp. $\Sigma_1 \wedge \cdots \wedge \Sigma_n$) dans $\mathcal{L}$ (les cas n borné à 2 sont traités comme des opérateurs spéciaux, resp. notés $\vee$ BC et $\wedge$ BC). On trouvera d'autres opérations comme la possibilité de calculer la négation d'une BC (noté $\neg$ **C**), la disjonction (noté $\vee$ **C**) ou encore l'« oubli » de variables (noté **FO** et **SFO** lorsqu'une seule variable doit être oubliée).

Dans la suite de leurs travaux, les auteurs introduisent aussi la notion de *compacité*, qui permet de comparer la taille minimale demandée à une *BCC* pour représenter une *BC* initiale suivant les langages de compilation visés (et donc les opérations et requêtes qu'elle autorise). Intuitivement, on cherchera à produire les BC les plus compactes autorisant les opérations et transformations impérativement nécessaires à l'application visée. Soient $\mathcal{L}_1$ et $\mathcal{L}_2$ deux sous-ensembles des $\mathcal{FNN}$, $\mathcal{L}_1$ est au moins **aussi compact que** $\mathcal{L}_2$ (noté $\mathcal{L}_1 \leq \mathcal{L}_2$), ssi pour toute formule Σ_2 de $\mathcal{L}_2$, il existe une formule équivalente Σ_1 de $\mathcal{L}_1$ telle que la taille de Σ_1 soit bornée polynomialement par la taille de Σ_2. Cette notion va donc permettre de hiérarchiser les langages cibles usuels, mais aussi de voir que de nouveaux langages sont, d'un point de vue théorique, mieux placés pour les problématiques usuelles de compilation. Ainsi, le langage $\mathcal{FNND}$ offre des compromis particulièrement intéressants.

Une formule sous *forme normale négative décomposable* ($\mathcal{FNND}$) est une $\mathcal{FNN}$ satisfaisant la propriété de décomposition : pour toute conjonction $\bigwedge_i f_i$ apparaissant dans la formule, aucune variable n'est partagée par deux f_i. Ce langage tente de ne

5. Notons que toute théorie peut être tout de même compilée en une théorie complète pour la résolution unitaire [del Val, 1994].

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
$\mathcal{FNN}$	o	o	o	o	o	o	o	o
$\mathcal{FNND}$	✓	o	✓	o	o	o	o	✓
$\mathcal{FNC}$	o	✓	o	✓	o	o	o	o
$\mathcal{FND}$	✓	o	✓	o	o	o	o	✓
$\mathcal{PI}$	✓	✓	✓	✓	✓	✓	o	✓
$\mathcal{PA}$	✓	✓	✓	✓	✓	✓	o	✓

TABLE 2 – Comparaison des requêtes en temps polynomial sur divers langages de compilation. Le symbole ✓ signifie que le langage donné satisfait la requête donnée, o qu'elle ne le satisfait pas.

$\mathcal{L}$	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
$\mathcal{FNN}$	✓	o	✓	✓	✓	✓	✓	✓
$\mathcal{FNND}$	✓	✓	✓	o	o	✓	✓	o
$\mathcal{FNC}$	✓	o	✓	✓	✓	o	✓	o
$\mathcal{FND}$	✓	✓	✓	o	✓	✓	✓	o
$\mathcal{PI}$	✓	✓	✓	o	o	o	✓	o
$\mathcal{PA}$	✓	o	o	o	✓	o	o	o

TABLE 3 – Le symbole ✓ signifie que le langage donné satisfait les opérations données, o qu'il ne les satisfait pas (nous ne faisons pas la différence entre la non satisfaction sous couvert P$\neq$NP et la non satisfaction dure)

s'appuyer que sur ce qui est suffisant pour garantir l'efficacité de la plupart des algorithmes classiques. Cette propriété de décomposition n'est pas nouvelle, même en IA, et a de forts liens avec la largeur induite (*induced width*). Cependant, en ne se basant que sur elle, le langage $\mathcal{FNND}$ permet, en quelque sorte, de ne demander que le minimum (parmi les techniques usuelles) pour garantir de bonnes propriétés. Dans [Subbarayan *et al.*, 2007], la notion d'arbres de ROBDD est introduite, et semble présenter une bonne alternative aux $\mathcal{FNND}$, au moins d'un point de vue théorique. La table 2 présente quelques requêtes efficaces sur ces langages de référence. Les langages $\mathcal{PA}$ (resp. $\mathcal{PI}$) sont les langages des impliquants premiers (resp. impliqués premiers). Il est aussi résumé, table 3 les performances théoriques des opérations possibles sur ces différents langages.

5.5.5 Exemple d'applications : ATMS et diagnostics à base de modèles

Tout au long des années 90, peu après l'introduction des ATMS [De Kleer, 1989], beaucoup de travaux basés sur les logiques non monotones se sont basés sur ce formalisme. Cependant, depuis les années 2000, c'est plutôt la tendance inverse qui est observée : d'une part, la maturité de la compilation de bases de connaissances a montré

que les ATMS n'étaient qu'un problème parmi d'autres au sein du cadre plus général de la CBC et, d'autre part, les performances obtenues autour de la résolution pratique de SAT et des CSP a rendu désuet toute tentative d'utilisation des ATMS pour les résoudre, au lieu d'une approche SAT ou CSP directe.

TMS et ATMS

Les systèmes de maintient de la cohérence (TMS pour *Truth Maintenance System*) ont été proposés dans [Doyle, 1979] puis étendus aux systèmes de maintien d'hypothèses cohérentes (ATMS pour *Assumption-based TMS*) dans [Reiter et de Kleer, 1987]. Un ATMS est conçu pour recevoir des formules, en logique propositionnelle, de sa partie « système de raisonnement ». La conjonction f de toutes ces formules représente la *base de connaissance* (BC) de l'ATMS, sur laquelle il va baser son raisonnement. De plus, $Var(f)$ est partitionné en deux parties : les variables d'hypothèses $Var_H(f)$ et les variables de données $Var_d(f)$. Dans [Reiter et de Kleer, 1987], est appelé *environnement* tout ensemble (il s'agit en fait de conjonctions) de littéraux. Le premier rôle d'un TMS est de garantir la cohérence de la BC, c'est-à-dire la satisfaisabilité de f lorsqu'on lui ajoute de nouvelles connaissances. Ensuite, grâce à tout le travail effectué en amont on peut demander à l'ATMS de vérifier si un environnement donné E est bien cohérent avec la BC (ce qui se traduit par un test de satisfaisabilité de $f \wedge E$). Si ce n'est pas le cas, on peut demander à l'ATMS de calculer le sous-environnement maximal de E qui soit consistent avec f.

On dit que la BC f et l'environnement E supportent les données d si $f \wedge E \models d$, c'est-à-dire si $f \wedge E \wedge \neg d$ n'est pas satisfaisable. Dans ce cas, l'ATMS peut être questionné pour renvoyer tous les ensembles minimaux des environnements E supportant ces mêmes données d. Parmi ceux-ci, on se focalise généralement sur ceux qui ne sont construits que sur des hypothèses. Tous les environnements minimaux construits uniquement sur $Var_H(f)$ et supportant les données d sont appelés *labels* de d.

Intuitivement, un ATMS va permettre d'exprimer toutes les hypothèses minimales expliquant certaines données. De plus, en autorisant des ajouts incrémentaux de formules dans la BC, un effet de bord intéressant peut se produire : les hypothèses expliquant des données peuvent être révisées, puisque l'ensemble des données impliquées évolue de manière non monotone à l'ajout de chaque nouvelle donnée. Comme montré dans [Madre et Coudert, 1991], l'architecture complète d'un ATMS peut être encodé par un système capable de mettre à jour efficacement l'ensemble des impliqués premiers (éventuellement restreints aux variables hypothèses) de la BC. L'approche Knowledge Compilation Map permet aussi de voir les ATMS et autres systèmes de maintien de la cohérence comme des cas particuliers de compilation de base de connaissance, avec une forte contrainte de mise à jour de la base. Ces deux derniers constats expliquent, au moins en partie, le fait que de moins en moins de travaux, ces dernières années, ne se focalisent que sur les ATMS. Les problématiques classiques d'IA exposées ci-dessus semblent peu à peu migrer vers les problématiques plus générales de la compilation de bases de connaissances.

Diagnostics à base de modèles

Pour finir le panorama des techniques les plus importantes d'IA autour du raisonnement sur les bases de connaissances en logique propositionnelle, nous proposons ici une courte introduction au diagnostics à base de modèles, issue de [de Kleer et Williams, 1987; de Kleer *et al.*, 1992], et restreinte au cadre propositionnel. Généralement, un système observé est défini comme un triplet (SD,COMPS,OBS) où SD est une formule de la logique du 1er ordre décrivant le comportement du système, OBS est une formule décrivant les observations (revenant dans la plupart des cas à une assignation de valeurs aux variables observables) et COMPS est l'ensemble des composants surveillés. Chacun de ces composants apparaît en tant qu'argument du prédicat prédéfini $Ab()$ dans SD (ex : $Ab(C_i)$ signifie que C_i est anormal, ou défaillant). En logique propositionnelle, il est possible de réunir ces ensembles en une seule théorie, T, avec, par exemple, la convention de nommage suivante : une variable okC_i encode un mode de comportement correct pour le composant C_i, c-à-d $\neg Ab(C_i)$. On appelle F l'ensemble des littéraux de modes négatifs $\{...,\neg okC_i,..\}$ représentant des composants défaillants. Pour un observable (booléen) encodé par une variable v, l'observation élémentaire $v = a$ est traduite par v si a est égal à $vrai$ et $\neg v$ si a est égal à $faux$.

Soit T une théorie décrivant un système observé et F un ensemble consistant de littéraux de mode négatifs, on a : $\Delta \subseteq F$ est un diagnostic ssi $T \cup \Delta \cup \overline{\{F \setminus \Delta\}} \not\models \bot$, et on note $Diag(T)$ l'ensemble des diagnostics de T et $Diag'(T)$ son ensemble de diagnostics minimaux. Plus précisément, et pour se conformer aux définitions de ce chapitre, les diagnostics minimaux sont les impliquants premiers de la conjonction des conflits minimaux où les conflits minimaux (appelés ensemble de conflits minimaux [Reiter, 1987] ou ensemble de conflits minimaux positifs dans [de Kleer *et al.*, 1992]) sont les impliquants premiers de la formule $SD \wedge OBS$ restreints aux littéraux de modes de F. Intuitivement, un conflit minimal fait référence à un ensemble de composants dont au moins l'un d'eux est défaillant. Les diagnostics minimaux sont par conséquent les plus petites conjonctions de composants défaillants expliquant toutes les fautes, étant données des observations. Cette définition statut qu'étant donné n'importe quel diagnostic minimal Δ d'une erreur observée, on peut supposer que tous les composants n'apparaissant pas dans Δ sont corrects. Nous remarquons aussi que, parce qu'on peut restreindre l'ensemble des erreurs possibles ($(\neg okOC \Rightarrow \neg valCC \vee ePurch)$ dans le précédent exemple, un diagnostic peut ne pas être étendu en supposant tous les composants C' incorrects (deux littéraux de mode étendant un diagnostic peuvent être incohérents).

La plupart des travaux sur le diagnostic à base de modèles sont donc très proches des problématiques liées aux ATMS, et donc sont entièrement subsumées par la compilation de bases de connaissances. En général, pour calculer les diagnostics, on calcule en premier lieu l'ensemble des conflits se restreignant aux littéraux de mode. C'est seulement après l'accomplissement de cette première étape que sont calculés les diagnostics minimaux. Bien entendu, on trouvera une introduction beaucoup plus détaillée aux diagnostics chapitre I.18.

5.6 Formules booléennes quantifiées

Les formules booléennes quantifiées (QBF) sont une extension naturelle de la logique propositionnelle classique, dans laquelle on s'autorise à quantifier existentiellement et/ou universellement certaines variables. Très clairement, soit f une formule propositionnelle (telle que définie au tout début de la section 5.2), toutes ses variables doivent être considérées comme étant quantifiées existentiellement, ce que l'on pourrait écrire $\exists X.f(X)$, où X est le vecteur de toutes les variables de f, noté $var(f)$. Les QBF sont tout simplement de telles formules dans lesquelles on s'autorise à quantifier chaque variable suivant son propre quantificateur. Le lecteur intéressé pourra se référer au chapitre 9 de [Saïs, 2008] ainsi qu'aux chapitres 23 et 24 de [Biere *et al.*, 2009].

Pour simplifier, nous introduirons dans cette section uniquement les QBF sous forme prénexe, polie et fermée, dont la matrice est sous CNF. Une QBF est sous forme prénexe si tous les quantificateurs ont été poussés à l'extérieur, *i.e.* elle peut s'écrire $f = Qx_1(...Qx_n(\varphi))$ où chaque occurrence de Q vaut soit $\forall$, soit $\exists$, et φ ne contient pas d'occurrence quantifiée de variable. On appelle φ la **matrice** de f. La suite $Qx_1...Qx_n$ est le **préfixe** de f. Qx_1 est la quantification la plus externe de f et Qx_n la plus interne. f est dite **polie** ssi chaque occurrence des variables est dans la portée d'un quantificateur unique, et **fermée** ssi toutes les variables sont bien quantifiées. Sans manque de généralité, on peut réécrire toute formule QBF de manière à ce qu'elle respecte ses trois propriétés, dès lors qu'on s'intéresse à sa validité [Büning *et al.*, 1995 ; Stockmeyer et Meyer, 1973] (comme pour SAT, cette transformation ne préserve pas l'équivalence).

L'interprétation sémantique des quantificateurs est aussi intuitive : $\exists x.f \equiv f|_{\mathbf{F}\mapsto x} \vee f|_{\mathbf{V}\mapsto x}$ et $\forall x.f \equiv f|_{\mathbf{F}\mapsto x} \wedge f|_{\mathbf{V}\mapsto x}$. La **validité** d'une QBF f est définie récursivement en se basant sur ces règles : si son préfixe est vide, la formule f est valide si elle est satisfaisable en logique "standard". Si f s'écrit $\exists x.f$ (resp. $\forall x.f$), f est valide si et seulement si $f|_{\mathbf{F}\mapsto x}$ ou (resp. et) $f|_{\mathbf{V}\mapsto x}$ sont satisfaisables. On notera aussi qu'il est possible de manipuler l'ordre des quantificateurs, dans une certaine mesure, grâce aux deux règles $\forall x.(\forall y.f) \equiv \forall y.(\forall x.f)$ et $\exists x.(\forall y.f) \equiv \forall y.(\exists x.f)$. L'ordre des quantificateurs n'est cependant pas anodin et on ne peut pas, en général, le changer sans risquer de changer la sémantique de la formule elle-même. De plus, le nombre d'alternances de quantificateurs dans le préfixe est un indicateur très fort de la difficulté de résolution pratique de la formule considérée, ce que l'on retrouve dans la hiérarchie polynomiale. On peut donc définir une formule QBF f en mettant en avant cette alternance, et en considérant les quantificateurs sur des vecteurs de variables : $f = Q_1X_1Q_2X2...Q_nX_n\varphi$, où $Q_i \neq Q_{i+1}$ pour tout $i \in [1, ..., n-1]$, et tels que $\forall i, j \in [1, ..., n], i \neq j$, on a $X_i \cap X_j = \varnothing$. Chaque Q_iX_i est appelé groupe de quantificateur. Intuitivement, ces groupes induisent des contraintes supplémentaires sur les dépendances des variables. Une variable quantifiée existentiellement pourra suivre une *politique* d'affectation dépendante de l'affectation des variables des groupes plus externes.

Si on ne borne pas le nombre d'alternances possibles, le problème QBF est, à l'instar de la NP-complétude de SAT, l'archétype des problèmes PSPACE-complet [Stockmeyer et Meyer, 1973]. On notera de plus que, si les certificats dans le cas « standard » SAT sont de longueur linéaire lorsque la formule est satisfaisable, il est impossible de borner polynomialement la longueur des certificats dans le cas QBF, que la formule soit valide

ou non. On parle d'ailleurs plutôt de **politique** de validité [Coste-Marquis *et al.*, 2006].

Le formalisme QBF permet d'exprimer de nombreux problèmes de manière compacte, ce qui les rend particulièrement intéressants pour nombre de problèmes classiques d'IA. À titre d'exemple, notons les travaux caractéristiques autour de la planification par réduction des problèmes à QBF [Cashmore *et al.*, 2012]. La réduction est possible et donne de bons résultats. Les progrès algorithmiques autour de SAT rendent cependant ces approches moins intéressantes que l'on aurait pu penser : les résultats pratiques obtenus demeurent loin derrière les approches à base de solveurs SAT.

5.6.1 Techniques pour la résolution des QBF

Les premiers algorithmes proposés remontent à [Büning *et al.*, 1995 ; Cadoli *et al.*, 1999]. Dans [Büning *et al.*, 1995], il est proposé d'étendre les techniques basés sur la résolution (typiquement DP) aux formules QBF. Cette approche se base sur la notion de Q-Résolution : l'élimination des littéraux existentiels se fait en appliquant la résolution, et en commençant par les littéraux les plus internes. Les littéraux universels sont éliminés de chaque clause dès lors qu'aucune variable existentielle n'a de rang plus interne dans la clause. Même si cet algorithme se base sur DP, qui est particulièrement peu efficace dans le cadre SAT, certaines QBF, admettant un grand nombre de variables quantifiées universellement, peuvent être résolues efficacement par cette méthode. Citons par ailleurs l'approche [Pan et Vardi, 2004] basée sur une représentation implicite des clauses, étendant l'approche [Simon et del Val, 2001], qui a donné de bons résultats.

Suivant de près les progrès des solveurs SAT, il a été naturellement proposé d'étendre les algorithmes de type DPLL à QBF. Il suffit *simplement* de restreindre le choix des variables durant la recherche avec retour arrière pour refléter les dépendances des groupes de variables [Cadoli *et al.*, 1998 ; Rintanen, 1999 ; Letz, 2002]. On notera que dans ces algorithmes, une clause propagée n'est pas forcément de taille 1 (elle peut contenir des littéraux universels sans littéral existentiel de rang plus interne). Les heuristiques de choix doivent aussi être restreintes aux variables du groupe de quantificateur le plus externe. De même, et contrairement à SAT, la détection et l'affectation des littéraux purs est très importante pour les solveurs QBF [Giunchiglia *et al.*, 2004].

Les principes des CDCL ont aussi été étendus aux QBF [Giunchiglia *et al.*, 2003, 2001 ; Zhang et Malik, 2002]. L'analyse de conflit est cependant légèrement différente lorsque le littéral assertif est quantifié universellement : le processus doit alors être poursuivit. Il est aussi possible d'apprendre des *goods* lors de la recherche [Zhang et Malik, 2002 ; Giunchiglia *et al.*, 2006].

5.7 Conclusion

La logique propositionnelle, malgré son apparente simplicité, représente paradoxalement l'un des enjeux majeurs de l'IA de ces dernières années. Au cœur de cette logique, le problème SAT fascine par ses liens avec la physique théorique, ainsi que par les enjeux applicatifs qu'il véhicule. Les performances affichées bousculent de nombreux

domaines, pour peu qu'ils soient relativement proches. Ainsi, on peut se demander pourquoi compiler une base de connaissance si chaque requête peut être, en pratique, traitée extrêmement rapidement par un démonstrateur SAT. La compilation permet de garantir des bornes supérieures, mais la taille de certains problèmes, une fois encodés, ne permettrait même pas à un algorithme quadratique en temps de répondre dans un délai raisonnable, là où, en pratique, des démonstrateurs SAT arrivent à répondre en quelques secondes sur la théorie non compilée.

De nombreux challenges demeurent toutefois pour la logique propositionnelle, comme une compréhension plus profonde des performances des démonstrateurs CDCL. Il n'existe en effet à ce jour pas d'explication réellement satisfaisante, au sens « scientifique », des performances pratiques de ces solveurs. Un tel paradigme, s'il existe, pourrait permettre un nouveau changement d'échelle dans les performances obtenues, par exemple en exploitant encore mieux le potentiel de parallélisation disponible dans tout ordinateur contemporain, ou en permettant d'étendre les gains de performance à la résolution des problèmes QBF, une famille de problèmes dont la résolution pratique reste encore extrêmement difficile.

Références

ACHLIOPTAS, D. et RICCI-TERSENGHI, F. (2010). Random formulas have frozen variables. *SIAM Journal of Computing.*

ARMONI, R., EGOROV, S., FRAER, R., KORCHEMNY, D. et VARDI, M. (2005). Efficient ltl compilation for sat-based model checking. *In ICCAD'2005.*

AUDEMARD, G., BENHAMOU, B. et SIEGEL, P. (1999). La méthode d'avalanche AVAL : une méthode énumérative pour SAT. *In Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC)*, pages 17–25.

AUDEMARD, G., BORDEAUX, L., HAMADI, Y., JABBOUR, S. et SAIS, L. (2008). A generalized framework for conflict analysis. *In SAT'2008.*

AUDEMARD, G., KATSIRELOS, G. et SIMON, L. (2010). A restriction of extended resolution for clause learning sat solvers. *In 24nd Conference on Artificial Intelligence(AAAI).*

AUDEMARD, G., LAGNIEZ, J.-M., MAZURE, B. et SAÏS, L. (2009). Learning in local search. *In 21st International Conference on Tools with Artificial Intelligence (ICTAI'09)*, Newark, New Jersey, USA. IEEE Computer Society.

AUDEMARD, G. et SIMON, L. (2007). Gunsat : a greedy local search algorithm for unsatisfiability. *In IJCAI'07 : Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2256–2261, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

AUDEMARD, G. et SIMON, L. (2009). Predicting learnt clauses quality in modern SAT solvers. *In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 399–404.

BACCHUS, F. et WINTER, J. (2003). Effective preprocessing with hyper-resolution and equality reduction. *In SAT'2003*, pages 341–355.

BIERE, A. (2006). AIGER format toolbox. `http://fmv.jku.at/aiger/`.

BIERE, A., CIMATTI, A., CLARKE, E. et ZHU, Y. (1999). Symbolic model checking without bdds.

BIERE, A., HEULE, M., VAN MAAREN, H. et WALSH, T., éditeurs (2009). *Handbook of Satisfiability.* IOS Press.

BRAUNSTEIN, A., MÉZARD, M. et ZECCHINA, R. (2002). Survey propagation : an algorithm for satisfiability. Rapport technique.

BRYANT, R. (1986). Graph - based algorithms for boolean function manipulation. *IEEE Trans. on Comp.*, 35(8) :677–691.

BÜNING, H. K., KARPINSKI, M. et FLÖGEL, A. (1995). Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1) :12–18.

BURO, M. et BÜNING, H. K. (1993). Report on a SAT competition. *Bulletin of the European Association for Theoretical Computer Science*, 49 :143–151.

CADOLI, M. et DONINI, M. (1997). A survey on knowledge compilation. *AI Communications*, 10 :137–150.

CADOLI, M., GIOVANARDI, A. et SCHAERF, M. (1998). An algorithm to evaluate quantified boolean formulae. *In Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 262–267.

CADOLI, M., SCHAERF, M., GIOVANARDI, M. et GIOVANARDI, M. (1999). An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *In Journal of Automated Reasoning*, pages 262–267. AAAI Press.

CASHMORE, M., FOX, M. et GIUNCHIGLIA, E. (2012). Planning as quantified boolean formula. *In ECAI 2012 - 20th European Conference on Artificial Intelligence*, pages 217–222.

CASTELL, T. (1997). *Consistance et déduction en logique propositionnelle.* Thèse de doctorat, Université Paul Sabatieur de Toulouse.

CHAMBERS, B., MANOLIOS, P. et VROON, D. (2009). Faster sat solving with better cnf generation. *In DATE.*

CHANG, C. L. et LEE, R. C. (1973). *Symbolic Logic and Mechanical Theorem Proving.* Computer Science Classics, Academic Press.

CHAZAL, G. (1996). *Éléments de logique formelle.* Hermès.

CHOI, A., ZAITLEN, N., HAN, B., PIPATSRISAWAT, K., DARWICHE, A. et ESKIN, E. (2008). Efficient genome wide tagging by reduction to sat. *In* CRANDALL, K. et LAGERGREN, J., éditeurs : *Algorithms in Bioinformatics*, volume 5251 de *Lecture Notes in Computer Science*, pages 135–147. Springer Berlin / Heidelberg.

COOK, S. (1971). The complexity of theorem proving procedures. *In ACM Symposium of Theory of Computing*, pages 151–158.

COOK, S. A. (1976). A short proof of the pigeon hole principle using extended resolution. *Sigact News*, Oct.-Dec. :28–32.

COSTE-MARQUIS, S., FARGIER, H., LANG, J., BERRE, D. L. et MARQUIS, P. (2006). Representing policies for quantified boolean formulae. *In KR*, pages 286–297.

COUDERT, O. et MADRE, J.-C. (1992). Implicit and incremental computation of primes and essential implicant primes of boolean functions. *In 29th ACM/IEEE Design Automation Conference (DAC'92)*, pages 36–39.

DARWICHE, A. et MARQUIS, P. (2001). A perspective on knowledge compilation. *In Proceedings of the 17th International Joint Conference on Articificial Intelligence (IJCAI'01)*, pages 175–182.

DAVIS, M., LOGEMANN, G. et LOVELAND, D. (1962). A machine program for theorem proving. *JACM*, 5 :394–397.

DAVIS, M. et PUTNAM, H. (1960). A computing procedure for quantification theory. *JACM*, 7 :201–215.

DE KLEER, J. (1989). A comparison of atms and csp techniques. *In IJCAI'89 : Proceedings of the 11th international joint conference on Artificial intelligence*, pages 290–296, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

de KLEER, J. (1992). An improved incremental algorithm for generating prime implicates. *In Proceedings of the National Conference on Artificial Intelligence (AAAI'92)*, pages 780–7085.

de KLEER, J., MACKWORTH, A. K. et REITER, R. (1992). Characterizing diagnoses and sytems. *Artificial Intelligence*, 56 :197–222.

de KLEER, J. et WILLIAMS, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, 32(1) :97–130.

DECHTER, R. et RISH, I. (1994). Directional resolution : The davis-putnam procedure. *In Proc. of the 4th Int'l Conf. on Principles of KR&R*, pages 134–145.

del VAL, A. (1994). Tractable databases : How to make propositional unit resolution complete through compilation. *In 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 551–561, Bonn.

del VAL, A. (1999). A new method for consequence finding and compilation in restricted languages. *In 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 259–264.

DIMACS (1993). Second challenge on satisfiability testing organized by the center for discrete mathematics and computer science of rutgers university.

DOWLING, W. et GALLIER, J. (1984). Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3) :267–284.

DOYLE, J. (1979). A truth maintenance system. *AI*, 12(3) :251–272.

DUBOIS, O. et DEQUEN, G. (2001). A backbone-search heuristic for efficient solving of hard 3-sat formulae. *In Proceedings of IJCAI'2001*, pages 248–253.

EÉN, N. et BIERE, A. (2005). Effective preprocessing in SAT through variable and clause elimination. *In proceedings of SAT*, pages 61–75.

EÉN, N. et SÖRENSON, N. (2003). An extensible sat-solver. *In SAT'2003*, pages 333–336.

FEIGE, U. (2002). Relations between average case complexity and approximation complexity. *In Proc. of the 34th ACM Symposium on Theory of Computing*, pages 534–543.

FRANCO, J. (1986). On the probabilistic performance of algorithms for the satisfiability problem. *Information Process. Lett.*, 23 :103–106.

GALIL, Z. (1977). On the complexity of regular resolution and the davis-putnam procedure. *Theoretical Computer Science*, 4 :23–46.

GAREY, M. et JOHNSON, D. (1979). *Computers and Intractability : A guide to the Theory of NP-Completeness.* W.H. Freeman and Company.

GIUNCHIGLIA, E., NARIZZANO, M. et TACCHELLA, A. (2001). Qube : A system for deciding quantified boolean formulas satisfiability. *In Proceedings of the First International Joint Conference on Automated Reasoning*, pages 364–369.

GIUNCHIGLIA, E., NARIZZANO, M. et TACCHELLA, A. (2003). Backjumping for quantified boolean logic satisfiability. *Artif. Intell.*, 145(1-2) :99–120.

GIUNCHIGLIA, E., NARIZZANO, M. et TACCHELLA, A. (2004). Monotone literals and learning in qbf reasoning. *In Principles and Practice of Constraint Programming – CP 2004*, volume 3258 de *Lecture Notes in Computer Science*, pages 260–273. Springer Berlin Heidelberg.

GIUNCHIGLIA, E., NARIZZANO, M. et TACCHELLA, A. (2006). Clause/term resolution and learning in the evaluation of quantified boolean formulas. *J. Artif. Int. Res.*, 26(1) :371–416.

GIUNCHIGLIA, F. et SEBASTIANI, R. (1996). Building decision procedures for modal logics from propositional decision procedure - the case study of modal k. *In Conference on Automated Deduction*, pages 583–597.

GOLDBERG, A. (1979). On the Complexity of the Satisfiability Problem. Rapport technique 16, New York University.

GOLDBERG, A., PURDOM, J. P. et BROWN, C. (1982). Average time analysis of simplified davis-putnam procedures. *Information Process. Lett.*, pages 72–75.

GRÉGOIRE, É., MAZURE, B., OSTROWSKI, R. et SAÏS, L. (2005). Automatic extraction of functional dependencies. *In Theory and Applications of Satisfiability Testing : 7th International Conference (SAT 2004), Revised Selected Papers, (SAT'04 Revised Selected Papers)*, pages 122–132. LNCS 3542.

HABET, D., LI, C. M., DEVENDEVILLE, L. et VASQUEZ, M. (2002). A hybrid approach for sat. *In CP '02 : Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 172–184, London, UK. Springer-Verlag.

HAKEN, A. (1985). The intractability of resolution. *Theorical Computer Science*, 39 :297–308.

HAMADI, Y., JABBOUR, S. et SAIS, L. (2009). Manysat : a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT).*

HATON, J.-P., BOUZID, N., CHARPILLET, F., HATON, M.-C., LÂASRI, B., LÂASRI, H., MARQUIS, P., MONDOT, T. et NAPOLI, A. (1991). *Le Raisonnement en Intelligence Artificielle.* InterEditions, collection Informatique Intelligence Artificielle (IIA).

HEULE, M. et VAN MAAREN, H. (2007). Effective incorporation of double look-ahead procedures. *In SAT'07 : Proceedings of the 10th international conference on Theory and applications of satisfiability testing*, pages 258–271, Berlin, Heidelberg. Springer-Verlag.

HIRSCH, E. A. et KOJEVNIKOV, A. (2002). Unitwalk : A new sat solver that uses local search guided by unit clause elimination.

HOOS, H. et STÜTZLE, T. (2004). *Stochastic Local Search : Foundations and Applications.* Morgan Kaufmann / Elsevier.

HUANG, J. (2007). The effect of restarts on the efficiency of clause learning. *In IJCAI'2007*, pages 2318–2323.

JACKSON, P. et SHERIDAN, D. (2004). Clause form conversions for boolean circuits. *In SAT*, pages 183–198.

JÄRVISALO, M., BIERE, A. et HEULE, M. (2010). Blocked clause elimination. *In TACAS*, pages 129–144.

JEROSLOW, R. et WANG, J. (1990). Solving Propositional Satisfiability Problems. *Annals of Mathematics and Artificial Intelligence*, pages 167–187.

KATEBI, H., SAKALLAH, K. A. et MARKOV, I. L. (2010). Symmetry and satisfiability : An update. *In SAT*, pages 113–127.

KAUTZ, H. et SELMAN, B. (1996). Pushing the envelope : Planning, propositional logic, and stochastic search. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'96)*, pages 1194–1201.

KAUTZ, H. et SELMAN, B. (2003). Ten challenges redux : Recent progress in propositional reasoning and search. *In In Proceedings of CP '03*, pages 1–18.

KEAN, A. et TSIKNIS, G. (1990). An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9 :185 :206.

KERNIGHAN, B. W. et LIN, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2) :291–307.

KOTTLER, S. (2010). Sat solving with reference points. *In SAT 2010*, pages 143–157.

KROC, L., SABHARWAL, A. et SELMAN, B. (2007). Survey propagation revisited. *In UAI*, pages 217–226.

KROC, L., SABHARWAL, A. et SELMAN, B. (2010). An empirical study of optimal noise and runtime distributions in local search. *In SAT'10.*

KULLMANN, O. (1999). On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97 :149–176.

LARDEUX, F., SAUBION, F. et HAO, J.-K. (2006). Gasat : a genetic local search algorithm for the satisfiability problem. *Evol. Comput.*, 14(2) :223–253.

LASSAIGNE, R. et de ROUGEMONT, M. (1996). *Logique et complexité.* Hermès. collection informatique.

LE BERRE, D. (2000). *Autour de SAT : le calcul d'impiquants P-restreints, algorithmes et applications.* Thèse de doctorat, Université Toulouse III - Paul Sabatier.

LE BERRE, D., ROUSSEL, O. et ROUSSEL, O. (2002-2009). The sat competitions (2002–2009). www.satcompetition.org.

LE BERRE, D., ROUSSEL, O. et SIMON, L. (2009). SAT competition. http ://www.satcompetition.org/.

LEBERRE, D. et SIMON, L., éditeurs (2006). *Special Volume on the SAT 2005 competitions and evaluations*, volume 2. Journal on Satisfiability, Boolean Modeling and

Computation.

LETZ, R. (2002). Lemma and model caching in decision procedures for quantified boolean formulas. *In Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 160–175.

LI, C. M. et ANBULAGAN (1997). Heuristics based on unit propagation for satisfiability problems. *In Proceedings of IJCAI'97*, pages 366–371.

LUBY, M., SINCLAIR, A. et ZUCKERMAN, D. (1993). Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4) :173–180.

MADRE, J.-C. et COUDERT, O. (1991). A logically complete reasoning maintenance system based on a logical constraint solver. *In 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 294–299, Sydney, Australia.

MARQUIS, P. (1999). *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5, chapitre Consequence Finding Algorithms, pages 41–145. Kluwer Academic Publishers.

MAZURE, B., SAÏS, L. et GRÉGOIRE, E. (1998). Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22(3-4) :319–331.

MAZURE, B., SAIS, L. et GREGOIRE, E. (97). Tabu search for sat. *In proceedings of AAAI*, pages 281–285.

MITCHELL, D., SELMAN, B. et LEVESQUE, H. (1992). Hard and easy distributions of sat problems. *National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465.

MONASSON, R., ZECCHINA, R., KIRKPATRICK, S., SELMAN, B. et TROYANSKY, L. (1998). Determining computational complexity from characteristic 'phase transition'. *Nature*, 400 :133–137.

MOSKEWICZ, M., CONOR, C., ZHAO, Y., ZHANG, L. et MALIK, S. (2001). Chaff : Engineering an Efficient SAT Solver. *In Proceedings of the 38th Design Automation Conference (DAC'01)*.

OSTROWSKI, R., GRÉGOIRE, E., MAZURE, B. et SAÏS, L. (2002). Recovering and exploiting structural knowledge from cnf formulas. *In CP'02*.

PAN, G. et VARDI, M. Y. (2004). Symbolic decision procedures for qbf. *In Proceedings of 10th Int. Conf. on Principles and Practice of Constraint Programming (CP 2004)*, pages 453–467. Springer.

PEARL, J. (1982). Reverend bayes on inference engines : A distributed hierarchical approach. *In AAAI'82*, pages 133–136.

PHAM, D. N., THORNTON, J. et SATTAR, A. (2007). Building structure into local search for sat. *In IJCAI'07*.

PIPATSRISAWAT, K. et DARWICHE, A. (2007). A lightweight component caching scheme for satisfiability solvers. *In proceedings of SAT*, pages 294–299.

PIPATSRISAWAT, K. et DARWICHE, A. (2009). On the power of clause-learning SAT solvers with restarts. *In Principles and Practice of Constraint Programming - CP 2009*.

PLAISTED, D. et GREENBAUM, S. (1986). A structure-preserving clause form transla-

tion. *Journal of Symbolic Computation*, 2(3) :466–483.

PRASAD, M., BIERE, A. et GUPTA, A. (2005). A survey of recent advances in SAT-based formal verification. *journal on Software Tools for Technology Transfer*, 7(2) :156–173.

QUINE, W. (1955). A way to simplify truth functions. *American Mathematical Monthly*, 52 :627–631.

REITER, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1) :57–96.

REITER, R. et de KLEER, J. (1987). Foundations of assumption-based truth maintenance systems : Preliminary report. *In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, pages 183–188.

RINTANEN, J. (1999). Improvements to the evaluation of quantified boolean formulae. *In Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, pages 1192–1197.

RISH, I. et DECHTER, R. (2000). Resolution versus search : two strategies for sat. *J. of Approximate Reasoning.*

ROBINSON, J. (1965). A machine oriented based on the resolution principle. *Journal of the ACM*, 12(1) :23–41.

SAFARPOUR, S., MANGASSARIAN, H., VENERIS, A., LIFFITON, M. H. et SAKALLAH, K. A. (2007). Improved design debugging using maximum satisfiability. *In FMCAD '07 : Proceedings of the Formal Methods in Computer Aided Design*, pages 13–19. IEEE Computer Society.

SAÏS, L., éditeur (2008). *Problème SAT : progrès et défis.* Hermès / Lavoisier.

SELMAN, B. et KAUTZ, H. (1993). Domain-independant extensions to gsat : Solving large structured satisfiability problems. *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 290–295.

SELMAN, B. et KAUTZ, H. (1996). Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2) :193–224.

SELMAN, B., KAUTZ, H. et COHEN, B. (1995). Local search strategies for satisfiability testing. *In DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532.

SELMAN, B., KAUTZ, H. et MCALLESTER, D. (1997). Ten challenges in propositional reasoning and search. pages 50–54. Morgan Kaufmann.

SELMAN, B., LEVESQUE, H. et MITCHELL, D. (1992). A new method for solving hard satisfiability problems. *In AAAI*, pages 440–446.

SHOSTAK, R. (1979). A practical decision procedure for arithmetic with function symbols. *Journal of ACM*, 26(2) :351–360.

SIEGEL, P. (1987). *Représentation et Utilisation de la connaissance en calcul propositionnel.* Thèse d'état, Université de Provence, GIA–Luminy.

SILVA, J. P. M. et SAKALLAH, K. A. (1996). Grasp a new search algorithm for satisfiability. *In ICCAD '96 : Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227, Washington, DC, USA. IEEE Computer Society.

SIMON, L. (2001a). *Multirésolution pour le test de consistance et la déduction en logique propositionnelle.* Thèse de doctorat, Université Orsay Paris XI.

SIMON, L. (2001b). *Multirésolution pour le test de consistance et la déduction en logique propositionnelle.* Thèse de doctorat, Université d'Orsay Paris 11.

SIMON, L. et del VAL, A. (2001). Efficient consequence finding. *In 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 359–365, Seattle, Washington, USA.

STÅLMARCK, G. (1994). A system for determining propositional logic theorem by applying values and rules to triplets that are generated from a formula. US Patent 5,276,897 ; Canadian Patent 2,018,828 ; European Patent 0 403 545 ; Swedish Patent 467 076.

STOCKMEYER, L. J. et MEYER, A. R. (1973). Word problems requiring exponential time. *In Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 1–9, New York, NY, USA. ACM.

SUBBARAYAN, S., BORDEAUX, L. et HAMADI, Y. (2007). Knowledge compilation properties of tree-of-bdds. *In AAAI'07.*

TISON, P. (1967). Generalized consensus theory and application to the minimization of boolean circuits. *In IEEE Transactions on Computers*, volume EC-16, pages 446–456.

TOMPKINS, D. A. D. et HOOS, H. H. (2004). Ubcsat : An implementation and experimentation environment for sls algorithms for sat and max-sat. *In In SAT*, pages 37–46.

TSEITIN, G. (1968). *Structures in Constructives Mathematics and Mathematical Logic*, chapitre On the complexity of derivations in the propositional calculus, pages 115–125. A.O. Slesenko.

TSEITIN, G. (1983). On the complexity of proofs in propositional logics. *In* SIEKMANN, J. et WRIGHTSON, G., éditeurs : *Automation of Reasoning : Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag.

VAL, A. (2000). Tractable classes for directional resolution. *In Proc. 17th (U.S.) Nat. Conf. on Articial Intelligence.*

VELEV, M. N. (2004). Efficient translation of boolean formulas to cnf in formal verification of microprocessors. *In ASP-DAC '04 : Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, pages 310–315.

WOEGINGER, G. J. (2003). Exact algorithms for np-hard problems : A survey. *In Combinatorial Optimization*, pages 185–207.

ZHANG, H. (1997). SATO : an efficient propositional prover. *In Proceedings of the International Conference on Automated Deduction (CADE'97), volume 1249 of LNAI*, pages 272–275.

ZHANG, L. et MALIK, S. (2002). Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. *In Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 200–215.

Chapitre 6

Raisonnement par contraintes

Dans ce chapitre, je vais présenter succinctement le raisonnement par contraintes. Ce domaine de recherche de l'intelligence artificielle (IA) est plus connu aujourd'hui sous le nom de programmation par contraintes, notamment depuis qu'il est utilisé à grande échelle pour résoudre des problèmes combinatoires dans des applications industrielles et surtout depuis qu'il s'est enrichi des apports de la programmation logique pour les aspects langage et de la recherche opérationnelle pour la propagation de contraintes complexes. Mais vu le thème de ce livre, je vais volontairement rester sur une présentation très IA du domaine. On pourra trouver que ce chapitre souffre d'un tropisme français. Si c'est vrai que je n'ai rien fait pour l'éviter, on doit quand même garder en tête que la communauté française de programmation par contraintes a toujours été à la pointe et est aujourd'hui sans conteste celle qui a la plus grosse production scientifique.

6.1 Introduction

La notion de contrainte comme restriction des combinaisons de valeurs que peuvent prendre un ensemble de variables est suffisamment naturelle pour être apparue assez tôt dans l'histoire de l'intelligence artificielle. On peut citer notamment les travaux de Fikes [1970] sur le système REF-ART, ou ceux de Waltz [1972] sur l'interprétation de contours.On a cependant l'habitude d'associer la naissance du raisonnement par contraintes au papier fondateur de Montanari [1974], qui définit formellement pour la première fois ce qu'est un réseau de contraintes. Ce papier sera suivi d'articles tout aussi importants par Mackworth [1977] et surtout Freuder [1978; 1982; 1985], qui donneront aux recherches leur orientation vers les cohérences locales et la propagation de contraintes, notions propres au raisonnement par contraintes.

La communauté française a eu pour pionniers Laurière [1978] et son système ALICE ainsi que Mohr [1986] avec l'algorithme de propagation AC4. Mais la place importante que la communauté française a aujourd'hui dans ce domaine doit sans doute beaucoup aux deux projets BAHIA et CSPFlex, du programme de recherches coordonnées IA

Auteur : Christian Bessiere.

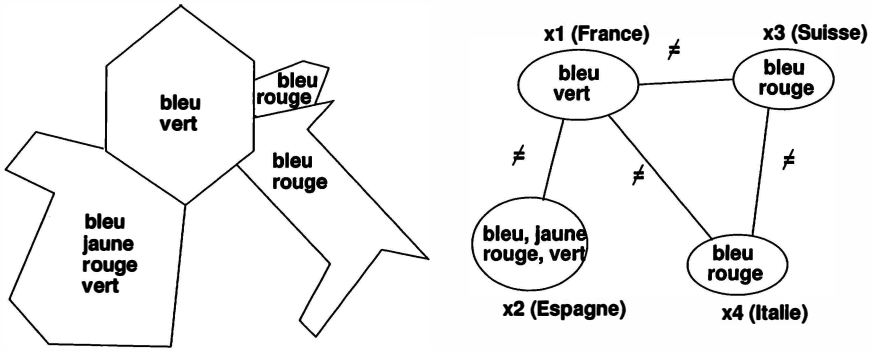


FIGURE 1 – Problème de coloriage de carte où chaque pays n'a qu'un ensemble restreint de couleurs autorisées (gauche). Réseau de contraintes modélisant le problème (droite).

(PRC IA) au début des années 90. Ils ont permis de structurer la communauté et ont donné un élan indéniable aux chercheurs qui ont eu la chance de participer à ce bouillon d'idées sur un domaine à ses débuts. corollary Un réseau de contraintes est composé de variables qui prennent chacune leur valeur dans leur domaine respectif, et de contraintes qui restreignent les combinaisons de valeurs possibles entre certaines variables. Un problème de satisfaction de contraintes (CSP pour *Constraint Satisfaction Problem*) consiste à savoir si un réseau de contraintes donné accepte ou non une solution, c'est-à-dire une affectation de valeurs aux variables qui satisfasse toutes les contraintes. Par exemple, on peut représenter le problème du coloriage de carte présenté en figure 1 (gauche) par un réseau de contraintes où chaque pays est représenté par une variable, son domaine de valeurs étant l'ensemble des couleurs disponibles pour ce pays (voir figure 1 (droite)). Chaque paire de variables représentant deux pays limitrophes est reliée par une contrainte spécifiant que les valeurs prises par ces deux variables doivent être différentes. Il est aisé de se convaincre qu'une solution à ce réseau de contraintes correspondra à un coloriage satisfaisant de notre carte.

Le premier intérêt des CSP, déjà mis en avant par Freuder en 1993 lors d'un tutoriel à IJCAI, est de fournir un formalisme stable sur lequel des algorithmes de résolution travaillent. Si vous arrivez à représenter votre problème en réseau de contraintes tel qu'une solution du réseau soit la représentation d'une solution de votre problème, vous pourrez utiliser n'importe quel algorithme de satisfaction de contraintes pour trouver votre solution.

Les utilisations du raisonnement par contraintes sont nombreuses. Parmi les premiers succès enregistrés, on peut citer l'optimisation du parcours des excavatrices à charbon dans des mines ou le planning des infirmières dans des hôpitaux. Depuis, on a vu le raisonnement par contraintes utilisé pour affecter des fréquences à des liaisons radio ou téléphone, affecter des trains à des quais, optimiser des chaînes de montage de voiture ou chercher des structures dans des séquences d'ARN.

Il existe d'autres formalismes qui définissent un standard de modélisation à partir duquel divers algorithmes peuvent être utilisés. SAT ou la programmation linéaire en

nombres entiers en sont des exemples. Un avantage des CSP par rapport à ces formalismes est l'expressivité disponible. Certains « motifs » se retrouvent dans beaucoup d'applications réelles, comme par exemple la contrainte `alldifferent` qui impose à un ensemble de variables de prendre des valeurs toutes différentes. Ces motifs peuvent être encapsulés dans une seule contrainte qui permettra à l'utilisateur de modéliser facilement tout un morceau de son problème et aux algorithmes de raisonner globalement sur ces motifs, qui autrement auraient dû être décomposés en sous-parties. Raisonner à partir du motif de départ permet de coller à la définition du problème et dans certains cas d'en déduire plus de choses que sur n'importe quelle décomposition en sous-problèmes de taille polynomiale.

6.2 Définitions

Je donne ici les définitions nécessaires à la compréhension de la suite de ce chapitre. J'ai autant que possible favorisé la simplicité quitte à parfois perdre un peu de rigueur quand il n'y a pas d'ambiguïté possible sur l'interprétation.

Définition 58 (Réseau de contraintes)**.** Un réseau de contraintes $P = (X, D, C)$ est composé de :

- une séquence finie $X = \{x_1, x_2, \ldots, x_n\}$ de *variables* entières,
- un *domaine* pour X, c'est-à-dire un ensemble $D = D(x_1) \times \ldots \times D(x_n)$, où $D(x_i) \subset \mathbb{Z}$ est fini et donné en extension, et
- un ensemble $C = \{c_1, \ldots, c_e\}$ de *contraintes*. Une contrainte $c_j \in C$ est une relation définie sur une séquence de variables $X(c_j) = (x_{j_1}, \ldots, x_{j_{|X(c_j)|}})$, appelée la *portée* de c_j. c_j est un sous-ensemble de $\mathbb{Z}^{|X(c_j)|}$.

La taille du réseau P sera souvent approximée à l'aide des paramètres $n = |X|$, $d = max_{i \in 1..n}|D(x_i)|$, $e = |C|$ et $r = max_{j \in 1..e}|X(c_j)|$.

Les valeurs listées dans $D(x_i)$ pourraient être de n'importe quel type, mais cela simplifie grandement le discours de considérer qu'elles sont toutes des entiers, ce qui n'est pas une restriction vu que $D(x_i)$ est fini. Un élément de $\mathbb{Z}^{|X(c_j)|}$ est appelé *tuple*. Un tuple de $\mathbb{Z}^{|X(c_j)|}$ est dit *valide* si et seulement s'il appartient à $D^{X(c_j)} = D(x_{j_1}) \times \ldots \times D(x_{j_{|X(c_j)|}})$. Les tuples de c_j sont appelés *autorisés* par c_j, les autres tuples de $\mathbb{Z}^{|X(c_j)|}$ sont *interdits* par c_j.

Exemple 13. L'exemple de coloriage de carte de la figure 1 (gauche) peut être modélisé par le réseau de la figure 1 (droite). Les variables $x_1, \ldots, x_4$ correspondent respectivement aux pays France, Espagne, Suisse, Italie. Les domaines sont $D(x_1) = \{B, V\}, D(x_2) = \{B, J, R, V\}$, $D(x_3) = D(x_4) = \{B, R\}$, où B, J, R, V sont des entiers qui représentent les couleurs. La contrainte entre France et Espagne, appelons-la c_1, qui garantit que France (variable x_1) et Espagne (variable x_2) prendront des couleurs différentes, peut se définir par $X(c_1) = (x_1, x_2)$ et $c_1 = \{(B, J), (B, R), (B, V), (V, B), (V, J), (V, R)\}$ ou plus simplement par $x_1 \neq x_2$.

Le concept d'instanciation est essentiel à la suite de ce chapitre.

Définition 59 (Instanciation). Une *instanciation I sur* un ensemble $Y \subseteq X$ de variables est une fonction qui affecte à chaque variable x_i de Y une valeur de son domaine $D(x_i)$ [1]. Si W est un sous-ensemble de Y, on note $I[W]$ la restriction (ou projection) de I aux variables de W. On dit d'une instanciation qu'elle *satisfait* une contrainte c_j si $X(c_j) \subseteq Y$ et $I[X(c_j)] \in c_j$. Si $X(c_j) \subseteq Y$ et $I[X(c_j)] \notin c_j$, on dit que I *viole* c_j. On dit d'une instanciation qu'elle est *localement cohérente* si et seulement si aucune des contraintes de C n'est violée.

Exemple 14. Sur notre exemple de coloriage de carte, $\{(x_1, B); (x_2, J); (x_4, B)\}$ est une instanciation sur $\{x_1, x_2, x_4\}$ (c'est-à-dire sur France, Espagne et Italie) qui n'est pas localement cohérente car elle viole la contrainte entre x_1 et x_4. $\{(x_1, B); (x_2, J); (x_4, R)\}$ est une instanciation sur $\{x_1, x_2, x_4\}$ localement cohérente.

Définition 60 (Solution). Une *solution* S d'un réseau de contraintes $P = (X, D, C)$ est une instanciation sur X qui ne viole aucune contrainte de C. On note $Sol(P)$ l'ensemble de toutes les solutions de P.

Définition 61 (CSP). On appelle *problème de satisfaction de contraintes* (noté CSP) le problème défini par :

Instance : Un réseau de contraintes $P = (X, D, C)$

Question : $Sol(P) \neq \varnothing$?

Théorème 37. CSP est NP-complet.

Preuve. C'est une conséquence directe du fait que SAT, le premier problème montré NP-complet, est un CSP particulier. □

Maintenant que nous avons cerné le modèle et la question centrale que l'on se pose, le chapitre suivant va recenser les techniques de base qui permettent de traiter cette question.

6.3 *Backtracking*

Comme dans tout problème NP-complet, l'énumération des possibilités est une technique naturelle pour résoudre le problème, c'est-à-dire trouver une solution. La méthode « générer et tester » (qui consiste à générer toutes les instanciations sur X jusqu'à en trouver une qui soit solution) étant quand même un peu bête, je vais directement présenter sa version améliorée qu'est le *backtrack chronologique (BT)* [Golomb et Baumert, 1965].

La fonction BT présentée en algorithme 6.1 prend en paramètre un réseau de contraintes P et une instanciation partielle localement cohérente I. L'appel $BT(P, \varnothing)$ imprime la première solution trouvée et renvoie **true** si P a des solutions, sinon renvoie **false**. Après avoir testé si I n'est pas une instanciation complète (ligne 1), la fonction

1. Dans la suite on représentera une instanciation indifféremment comme une séquence ou comme un ensemble d'affectations de valeurs à des variables. L'affectation de la valeur v_i à la variable x_i est notée (x_i, v_i).

Algorithme 6.1 : Backtrack chronologique

```
function BT (P : problem ; I : instanciation) : Boolean
  begin
1   if |I| = |X| then
      imprimer I ; return true
2   choisir une variable x_i pas encore instanciée dans I
3   foreach v_i ∈ D(x_i) do
4     if I ∪ {(x_i, v_i)} est localement cohérente then
5       if BT(P, I ∪ {(x_i, v_i)}) then
          return true
6   return false
```

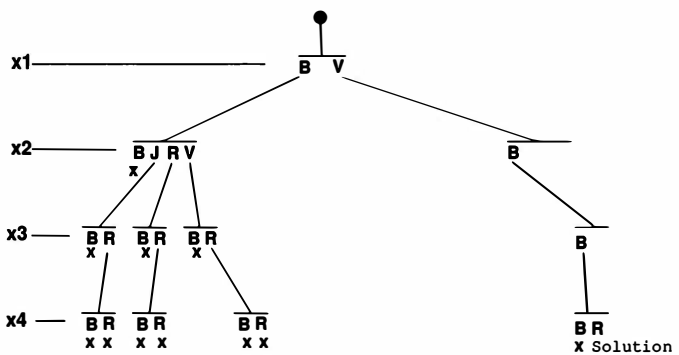


FIGURE 2 – Arbre de recherche de BT sur le réseau de contraintes de l'exemple 13.

sélectionne une variable x_i non instanciée (ligne 2) puis essaie tour à tour toutes ses valeurs jusqu'à en trouver une qui s'étende en une solution. Une fois une valeur v_i choisie, et si l'instanciation I augmentée de l'affectation $x_i \leftarrow v_i$ est localement cohérente (ligne 4), la fonction est rappelée récursivement avec la nouvelle instanciation (ligne 6.1). Si toutes les valeurs de $D(x_i)$ ont été essayées sans succès, la fonction renvoie **false** (ligne 6). On appelle cela un *retour arrière.* L'espace exploré par BT est appelé *arbre de recherche.* Les instanciations en sont les noeuds et ont un arc entrant provenant de la précédente instanciation localement cohérente. La racine de l'arbre est l'instanciation vide. La figure 2 représente l'arbre exploré par BT si on sélectionne les variables (ligne 2) et les valeurs (ligne 3) dans l'ordre lexicographique.

6.4 Propagation de contraintes

Le raisonnement par contraintes utilise plusieurs moyens pour réduire la taille de l'espace de recherche exploré par la procédure BT. La propagation de contraintes est le principal de ces moyens. La propagation de contraintes consiste à explicitement interdire des valeurs ou combinaisons de valeurs pour des variables parce qu'autrement, un ensemble de contraintes ne pourra pas être satisfait. Par exemple, dans un problème contenant les deux variables x_1 et x_2 de domaines 1..10, et une contrainte spécifiant que $|x_1 - x_2| > 5$, si on propage cette contrainte on peut interdire les valeurs 5 et 6 pour x_1 et pour x_2. Supprimer ces valeurs des domaines de x_1 et x_2 est un moyen de réduire l'espace des combinaisons qu'une procédure de recherche doit explorer.

La propagation de contraintes peut être présentée selon deux axes : *les cohérences locales* et *l'itération de règles de réduction.* Une cohérence locale définit une propriété que le problème doit satisfaire *après* la phase de propagation. Le côté opérationnel n'est pas spécifié. A l'inverse, l'approche par règles de réductions décrit le mécanisme de propagation lui-même. Les règles sont des conditions sur le type d'opérations de réduction qui peuvent être appliquées au problème. Je présenterai la propagation de contraintes principalement au travers des cohérences locales.

6.4.1 Cohérence sur une contrainte à la fois

Arc-cohérence

L'arc-cohérence (ou cohérence d'arc) est la plus ancienne et la plus connue des cohérences locales. Elle garantit que chaque valeur de chaque variable est compatible avec chaque contrainte prise séparément. L'article fondateur sur l'arc-cohérence est dû à Mackworth [1977], qui le premier a clairement défini ce concept.

Exemple 15. Soit un réseau P contenant les trois variables x_1, x_2 et x_3, de domaines $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3\}$, et les contraintes $x_1 = x_2$ et $x_2 < x_3$ (voir figure 3). P n'est pas arc-cohérent parce que certaines valeurs sont incompatibles avec certaines contraintes. En examinant la contrainte $x_2 < x_3$ on voit que la valeur 3 pour x_2 doit être supprimée parce qu'il n'y a pas de valeur plus grande que 3 dans $D(x_3)$. On peut aussi supprimer la valeur 1 de $D(x_3)$ à cause de cette même contrainte $x_2 < x_3$. Supprimer 3 de $D(x_2)$ provoque la suppression de la valeur 3 pour x_1 à cause

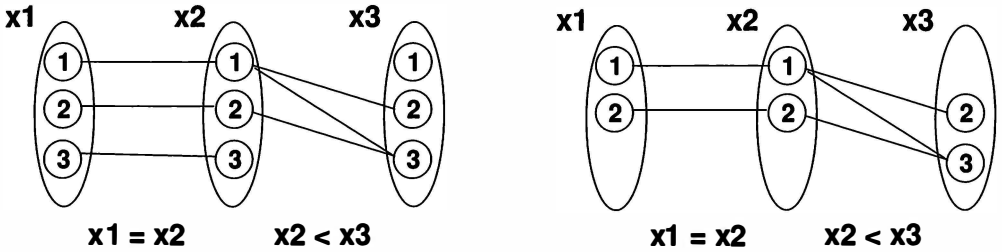


FIGURE 3 – Réseau de l'exemple 15 avant l'arc-cohérence (gauche) et après (droite). Les arêtes représentent les paires de valeurs autorisées pour chaque contrainte.

de $x_1 = x_2$. Maintenant, toutes les valeurs restantes sont compatibles avec toutes les contraintes.

Je donne la définition d'arc-cohérence dans sa forme la plus générale, c'est-à-dire pour les contraintes d'arité quelconque. Dans ce cas, elle est souvent appelée arc-cohérence généralisée.

Définition 62 (Arc-cohérence (AC)). Etant donnés un réseau $P = (X, D, C)$, une contrainte $c \in C$, et une variable $x_i \in X(c)$,

- Une valeur $v_i \in D(x_i)$ est *cohérente avec* c dans D si et seulement si il existe un tuple valide τ satisfaisant c tel que $v_i = \tau[\{x_i\}]$. Un tel tuple est appelé *support* pour (x_i, v_i) sur c.
- La contrainte c est arc-cohérente sur D si et seulement si toutes les valeurs de toutes les variables dans $X(c)$ ont un support sur c.
- Le réseau P est *arc-cohérent* si et seulement si toutes les contraintes de C sont arc-cohérentes sur D.

Appliquer l'arc-cohérence sur le réseau $P = (X, D, C)$ signifie calculer la *fermeture arc-cohérente* $AC(P)$ de P. $AC(P)$ est le réseau (X, D_{AC}, C) où $D_{AC} = \cup\{D' \subseteq D \mid (X, D', C)$ est arc-cohérent$\}$. $AC(P)$ est arc-cohérent et est *unique*. Il a les mêmes solutions que P. $AC(P)$ se calcule en supprimant itérativement les valeurs qui n'ont pas de support sur une contrainte jusqu'à atteindre un point fixe où toutes les valeurs ont des supports sur toutes les contraintes.

On verra par la suite qu'appliquer l'arc-cohérence dans un réseau de contraintes est une tâche essentielle à la recherche de solutions. Proposer des algorithmes efficaces d'arc-cohérence a donc toujours été une préoccupation importante de la communauté. Les idées utilisées pour améliorer les algorithmes d'arc-cohérence sont d'ailleurs souvent utilisées pour améliorer les algorithmes de propagation pour d'autres types de cohérences que nous verrons plus loin.

AC3 est l'algorithme d'arc-cohérence le plus connu et le plus simple, même si ce n'est pas le plus performant. Il a été proposé dans [Mackworth, 1977] pour les contraintes binaires. L'algorithme 6.2 le décrit dans sa forme pour contraintes d'arité quelconque.

Le composant principal d'AC3 est la révision d'un arc, c'est-à-dire la suppression

Algorithme 6.2 : AC3 (aussi noté GAC3 sur les contraintes non binaires)

```
function Revise(in x_i : variable; c : constraint) : Boolean
   begin
1      CHANGE ← false
2      foreach v_i ∈ D(x_i) do
3          if ∄ un tuple valide τ ∈ c avec τ[x_i] = v_i then
4              supprimer v_i de D(x_i)
5              CHANGE ← true
6      return CHANGE

function AC3(in X : set) : Boolean
   begin
7      Q ← {(x_i, c) | c ∈ C, x_i ∈ X(c)}
8      while Q ≠ ∅ do
9          prendre (x_i, c) de Q
10         if Revise(x_i, c) then
11             if D(x_i) = ∅ then
                   return false
12             else
                   Q ← Q ∪ {(x_j, c') | c' ∈ C ∧ c' ≠ c ∧ x_i, x_j ∈ X(c') ∧ j ≠ i}
13     return true
```

des valeurs d'une variable incohérentes sur une contrainte. Le nom « arc » provient du cas binaire. La fonction `Revise`(x_i, c) passe chaque valeur v_i de $D(x_i)$ une par une (ligne 2), et explore $D^{X(c)\setminus\{x_i\}}$ pour trouver un support sur c pour v_i (ligne 3). Si un tel support n'est pas trouvé, v_i est supprimé de $D(x_i)$ et le fait que $D(x_i)$ ait été modifié est mémorisé (lignes 4–5). La fonction renvoie vrai si le domaine $D(x_i)$ a été modifié, faux sinon (ligne 6).

L'algorithme AC3 est une simple boucle qui révise les arcs jusqu'à ce que plus aucune modification ne soit faite. Les domaines sont alors tous arc-cohérents sur toutes les contraintes. Pour éviter trop d'appels inutiles à `Revise`, AC3 maintient une liste Q de toutes les paires (x_i, c) pour lesquelles on n'est pas sûrs que $D(x_i)$ soit arc-cohérent sur c. En ligne 7, Q est initialisée avec toutes les paires (x_i, c) possibles telles que $x_i \in X(c)$. La boucle principale (ligne 8) prend les paires (x_i, c) une par une dans Q (ligne 9) et appelle `Revise`(x_i, c) (ligne 10). Si $D(x_i)$ est vidé, AC3 renvoie faux (ligne 11). Sinon, si $D(x_i)$ est modifié, il est possible qu'une valeur d'une autre variable x_j ait perdu ses supports sur une contrainte c' impliquant à la fois x_i et x_j. Donc, toutes les paires (x_j, c') telles que $x_i, x_j \in X(c')$ doivent être mises à nouveau dans Q (ligne 12). Lorsque Q est vide, AC3 renvoie vrai (ligne 13) puisque tous les arcs ont été révisés et toutes les valeurs restantes de toutes les variables sont cohérentes avec toutes les contraintes.

AC3 est polynomial en l'arité des contraintes. Il est en $O(er^3d^{r+1})$, où r est l'arité la plus grande des contraintes de C. Sur des réseaux de contraintes binaires cela donne

$O(ed^3)$. La complexité d'AC3 n'est pas optimale car `Revise` ne mémorise rien sur les calculs faits pour trouver les supports et doit donc faire et refaire les mêmes tests de contraintes à chaque appel.

De nombreux travaux ont été publiés pour améliorer la complexité d'AC3. Mohr et Henderson [1986] ont proposé AC4, le premier algorithme optimal d'arc-cohérence ($O(ed^2)$ sur les contraintes binaires et $O(erd^r)$ en général), au prix d'une structure de données très coûteuse. Bessiere et al. [2005] ont proposé AC2001, qui a l'avantage d'être basé sur le même schéma qu'AC3 tout en ayant une complexité optimale grâce à une structure de pointeurs facile à implémenter.

Borne-cohérence

Quand une contrainte a une arité trop grande ou quand les domaines sont très grands, l'arc-cohérence peut s'avérer trop coûteuse. Une alternative est d'utiliser la *borne-cohérence (BC)*, une cohérence plus faible qui tire parti du fait que les domaines sont composés d'entiers et héritent donc de l'ordre total sur $\mathbb{Z}$. On peut définir la plus petite et la plus grande valeur d'un domaine $D(x_i)$, notées $min_D(x_i)$ and $max_D(x_i)$ respectivement, et appelées les *bornes* de $D(x_i)$.

Définition 63 (Borne-cohérence)**.** Etant donnés un réseau $P = (X, D, C)$ et une contrainte c, un *support aux bornes* τ sur c est un tuple qui satisfait c et tel que pour tout $x_i \in X(c), min_D(x_i) \leq \tau[x_i] \leq max_D(x_i)$. Une contrainte c est *borne-cohérente* si et seulement si pour tout $x_i \in X(c)$, $(x_i, min_D(x_i))$ et $(x_i, max_D(x_i))$ appartiennent à un support aux bornes sur c. P est borne-cohérent si et seulement si toutes ses contraintes sont borne-cohérentes.

Exemple 16. Soient les variables x, y, z, les domaines $D(x) = \{0, 5\}$, $D(y) = \{0, 2, 5, 12\}$, $D(z) = \{4, 5\}$, et la contrainte $|x - y| = z$. BC supprimera seulement la valeur 12 de $D(y)$ parce que c'est la seule borne qui n'a pas de support aux bornes. La valeur 2 pour y n'a pas de support aux bornes mais n'est pas une borne, et la valeur 4 de z est une borne mais a des supports aux bornes, comme $((x, 5), (y, 1), (z, 4))$ par exemple.

Règles de réduction

Une règle de réduction spécifie une condition suffisante pour supprimer des valeurs. L'algorithme de propagation itère sur les règles jusqu'à ce qu'aucun domaine ne soit modifié, c'est-à-dire que l'on ait atteint un point fixe. L'approche par règles de réduction est utile principalement quand on a des informations sur la sémantique de la contrainte à propager car cela permet de spécifier simplement et efficacement un moyen de la propager. Il est en effet souvent inefficace d'utiliser un algorithme générique comme AC3 quand on a des informations sur la contrainte. Cette approche est très utilisée dans les résolveurs car ils contiennent en général beaucoup de contraintes de base prédéfinies, notamment des contraintes arithmétiques. Sur ces contraintes, une modification dans un domaine n'a souvent pas le même effet sur les autres variables de la contrainte selon que c'est la suppression d'une valeur au milieu du domaine, la suppression de la valeur minimum, la suppression de la valeur maximum ou une affectation. Les résolveurs sont

donc souvent munis de la capacité à différencier ces types de réductions de domaines, appelées *événements*. Les événements reconnus par la majorité des résolveurs sont :

— `RemValue`(x_i) : quand une valeur v est supprimée de $D(x_i)$
— `IncMin`(x_i) : quand la valeur minimum de $D(x_i)$ est supprimée
— `DecMax`(x_i) : quand la valeur maximum de $D(x_i)$ est supprimée
— `Instantiate`(x_i) : quand $D(x_i)$ devient un singleton

A l'aide de ces quatre types d'événements on peut spécifier des règles de réduction pour de nombreuses contraintes simples.

Exemple 17. Soit la contrainte `alldifferent`(x, y, z) avec $D(x) = D(y) = D(z) = \{1, 2, 3, 4\}$. Plutôt que d'appeler la fonction `Revise` pour propager cette contrainte, ce qui coûterait $O(d^3)$, nous pouvons construire l'ensemble de règles suivant :

R1 : if `Instantiate`(x) **then** $D(y) \leftarrow D(y) \setminus D(x)$; $D(z) \leftarrow D(z) \setminus D(x)$
R2 : if `Instantiate`(y) **then** $D(x) \leftarrow D(x) \setminus D(y)$; $D(z) \leftarrow D(z) \setminus D(y)$
R3 : if `Instantiate`(z) **then** $D(x) \leftarrow D(x) \setminus D(z)$; $D(y) \leftarrow D(y) \setminus D(z)$

Appliquer ces règles est très efficace car chacune des règles s'exécute en temps constant. On remarque que la différenciation des types d'événements permet de ne réveiller une règle sur une contrainte que si elle risque d'entraîner d'autres suppressions. Par exemple, si 1 et 3 sont supprimés de $D(z)$, seulement `RemValue`(z) et `IncMin`(z) sont vrais, donc aucune règle de la contrainte `alldifferent`(x, y, z) n'a besoin d'être réveillée. Si 1,2 et 3 sont supprimés de $D(z)$, `Instantiate`(z) est vrai. Donc la règle **R3** est appliquée.

Notons que l'ensemble de règles **R1**, **R2**, **R3** n'est pas suffisant pour garantir l'arc-cohérence. Supposons que 3 et 4 soient supprimés de $D(x)$ et $D(y)$ alors que $D(z) = \{1, 2, 3, 4\}$. Les règles **R1**, **R2**, **R3** ne suppriment aucune valeur alors que 1 et 2 dans $D(z)$ ne sont pas arc-cohérentes.

6.4.2 Cohérences fortes

L'arc-cohérence n'est pas le seul moyen de détecter des incohérences dans un réseau. Dès les années 70, plusieurs auteurs ont proposé des propriétés qui permettent de découvrir plus d'incohérences que l'arc-cohérence. Freuder [1978] a proposé les k-cohérences, toute une classe de cohérences locales plus fortes que l'arc-cohérence.

Définition 64 (k-cohérence)**.** Soit $P = (X, D, C)$ un réseau.

— Etant donné un ensemble de variables $Y \subseteq X$ avec $|Y| = k - 1$, une instanciation localement cohérente I sur Y est k-cohérente si et seulement si pour toute k^{eme} variable $x_{i_k} \in X \setminus Y$ il existe une valeur $v_{i_k} \in D(x_{i_k})$ telle que $I \cup \{(x_{i_k}, v_{i_k})\}$ est localement cohérente.
— Le réseau P est *k-cohérent* si et seulement si pour tout ensemble Y de $k - 1$ variables, toute instanciation localement cohérente sur Y est k-cohérente.

Avant que Freuder ne propose la k-cohérence, Montanari [1974] avait proposé la chemin-cohérence. Elle était définie pour les réseaux de contraintes binaires dans lesquels chaque paire de variables est impliquée dans au plus une contrainte. Sur de tels réseaux, Mackworth [1977] a montré que la chemin-cohérence est équivalente à la 3-cohérence.

Exemple 18. Soit le réseau P avec les variables x_1, x_2, x_3, les domaines $D(x_1) = D(x_2) = D(x_3) = \{1,2\}$, et $C = \{x_1 \neq x_2, x_2 \neq x_3\}$. P n'est pas chemin/3-cohérent parce que ni $((x_1,1),(x_3,2))$, ni $((x_1,2),(x_3,1))$ ne peuvent s'étendre à une valeur de x_2 satisfaisant à la fois c_{12} et c_{23}. Le réseau $P' = (X, D, C \cup \{x_1 = x_3\})$ est chemin/3-cohérent.

Freuder a aussi défini la k-cohérence forte. Cette cohérence locale garantit que le réseau est j-cohérent pour $1 \leq j \leq k$.

Définition 65 (k-cohérence forte)**.** Un réseau est *fortement k-cohérent* si et seulement si il est j-consistant pour tout $j \leq k$.

Lorsque k est grand, rendre un réseau k-cohérent est beaucoup trop cher en pratique, à la fois en temps et en espace. Cependant, la notion de k-cohérence n'est pas totalement inutile car elle permet de donner une compréhension opérationnelle de la notion de *cohérence globale*. Un réseau est globalement cohérent lorsque *toutes* les instanciations localement cohérentes peuvent s'étendre en une solution. Cela veut dire que la fonction BT décrite en algorithme 6.1 trouve une solution ou prouve qu'il n'y en a pas sans avoir à effectuer aucun retour arrière. Un réseau est globalement cohérent si et seulement si il est fortement n-cohérent, avec $n = |X|$.

Freuder [1985] a aussi proposé les (i,j)-cohérences, des cohérences locales basées sur les variables. La (i,j)-cohérence garantit que toute instanciation de taille i localement cohérente peut être étendue à n'importe quelles j variables supplémentaires.

Janssen et al. [1989] et al. ont proposé la première cohérence locale basée sur le nombre de contraintes impliquées et non pas sur le nombre de variables.

Définition 66 (Paire-cohérence)**.** Etant donné un réseau $P = (X, D, C)$, une paire de contraintes c_1 et c_2 dans C est paire-cohérente si et seulement si toute instanciation sur $X(c_1)$ (resp. sur $X(c_2)$) satisfaisant c_1 (resp. c_2) peut être étendu en une instanciation sur $X(c_1) \cup X(c_2)$ satisfaisant c_2 (resp. c_1). P est *paire-cohérent* si et seulement si toute paire de contraintes dans C est paire-cohérente.

Exemple 19. Soit le réseau avec les variables x_1, x_2, x_3, x_4, les domaines $D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{1,2\}$ et les contraintes $c_1(x_1,x_2,x_3) = \{(121),(211),(222)\}$ et $c_2(x_2,x_3,x_4) = \{(111),(222)\}$. Ce réseau est arc-cohérent. Cependant il n'est pas paire-cohérent parce que le tuple (121) de c_1 n'est compatible avec aucun tuple dans c_2.

D'autres auteurs ont proposé des cohérences locales, qui sont paramétrées par le nombre de contraintes impliquées dans le raisonnement. Jégou [1993] a proposé l'hyper k-cohérence, qui peut être vue comme le dual de la k-cohérence basée sur les contraintes. Dechter et van Beek [1997] ont proposé la famille des cohérences *(i,m)-relationnelles.* une forme de cohérence qui se concentre sur les incohérences à l'intérieur des portées des contraintes existantes.

Les cohérences citées ci-dessus ont en commun un défaut majeur pour une intégration facile dans un résolveur. Elles créent de nouvelles contraintes qui n'étaient pas dans le réseau initial ou modifient des contraintes existantes (voir exemples 18 et 19).

Dans les deux cas, ces contraintes sont coûteuses à stocker et à propager.

Certaines cohérences locales permettent de supprimer plus de valeurs que l'arc-cohérence tout en laissant les contraintes inchangées. La *singleton-arc-cohérence (SAC)* proposée par Debruyne et Bessiere [2001] est la plus connue, à la fois parce qu'elle permet un bon compromis entre niveau de propagation et coût en temps, et parce qu'elle peut s'intégrer assez facilement dans l'architecture des résolveurs actuels.

Définition 67 (Singleton-arc-cohérence)**.** Un réseau $P = (X, D, C)$ est *singleton-arc-cohérent* si et seulement si pour tout $x_i \in X$, pour tout $v_i \in D(x_i)$, le sous-problème $P|_{x_i=v_i}$, où le domaine de x_i dans P a été réduit au singleton $\{v_i\}$, peut être rendu arc-cohérent sans vider de domaine.

Appliquer la singleton-arc-cohérence revient donc à essayer à tour de rôle toutes les instanciations de taille 1 $x_i = v_i$ et à tester si elles conduisent à un échec quand on propage l'arc-cohérence sur le réseau obtenu. Un échec veut dire que l'instanciation $x_i = v_i$ n'appartient à aucune solution et donc que v_i peut être supprimée du domaine de x_i. Une approche de ce type limitée au test des bornes d'intervalles continus en ne propageant que la borne-cohérence au lieu de l'arc-cohérence a d'abord été utilisée sur les CSP numériques sous le nom de 3B-cohérence [Lhomme, 1993] puis en ordonnancement sous le nom de *shaving*.

6.5 Cas polynomiaux

Le CSP étant NP-complet, il est naturel de se poser la question de l'existence de classes particulières de réseaux de contraintes pour lesquelles CSP deviendrait polynomial.

Les premiers travaux dans ce sens ont été conduits par Freuder et utilisent la *structure* du réseau de contraintes, que l'on représente par son *graphe associé*. Le graphe associé à un réseau de contraintes $P = (X, D, C)$ est le graphe $G_P = (X, E)$ qui a un sommet par variable de P et une arête $\{x_i, x_j\}$ dans E si et seulement si il existe une contrainte $c \in C$ avec $\{x_i, x_j\} \subseteq X(c)$. Freuder [1982] définit la *largeur*[2] du graphe associé à un réseau de contraintes. La largeur est l'entier k le plus petit tel qu'une procédure gloutonne supprimant un sommet chaque fois qu'il est de degré inférieur ou égal à k pourra supprimer tous les sommets du graphe. L'ordre inverse de l'ordre dans lequel la procédure a supprimé les sommets est un ordre de largeur k.

Théorème 38 (Freuder, 1982)**.** Si un réseau de contraintes P a un graphe associé de largeur k et est fortement $(k+1)$-cohérent, alors appliquer la procédure BT sur un ordre de ses variables de largeur k est sans retour arrière.

Dans le même article, Freuder dénonce les limites de ce théorème. En effet, appliquer la $(k+1)$-cohérence avec $k > 1$ crée de nouvelles contraintes et donc modifie la largeur du graphe associé au réseau, ce qui invalide la précondition du théorème. Freuder caractérise le cas où la structure n'est pas modifiée.

2. La largeur de Freuder est une borne inférieure à la *tree-width* d'un graphe [Arnborg, 1985].

Corollaire 1 (Freuder, 1982). Un réseau ayant un graphe associé de largeur 1 (arbre) peut être résolu en $O(nd^2)$ (arc-cohérence puis BT sans retour arrière).

De nombreux travaux ont essayé d'étendre les résultats de Freuder à des structures de réseaux de contraintes plus générales que les arbres. L'idée sous-jacente est que si la *tree-width* du graphe associé est bornée, alors le réseau est polynomial à résoudre [Freuder, 1990].

Mais il y a une autre approche à la recherche de classes traitables, c'est celle qui au lieu de poser des restrictions sur la structure du réseau, pose les restrictions sur le type de contraintes utilisées dans le réseau. Cooper a proposé les contraintes ZOA, un des premiers exemples de classe traitable de contraintes. Une contrainte binaire c_j avec $X(c) = (x_i, x_j)$ est ZOA (pour zero, one, all) si et seulement si chaque valeur $v_i \in D(x_i)$ a soit aucun, soit un, soit $|D(x_j)|$ supports dans $D(x_j)$.

Théorème 39 (Cooper et al., 1994). Si toutes les contraintes d'un réseau de contraintes binaires sont ZOA, alors la chemin-cohérence est suffisante pour que la procédure BT soit sans retour arrière.

De nombreuses autres classes de contraintes ont été proposées, qui offrent la garantie que le réseau est polynomial à résoudre. On peut citer les *connected row convex* [van Beek et Dechter, 1995 ; Deville *et al.*, 1999] ou les nombreux travaux de Cohen et al. [2006].

6.6 Synthèse de solutions et décompositions

Dechter et Pearl [1988] ont proposé *Adaptive consistency (AdC)*, une technique qui permet d'utiliser le théorème 38 tout en tenant compte des limites soulevées par Freuder (voir ci-dessus). Plutôt que d'appliquer uniformément la k-cohérence à tout le réseau, on n'y applique qu'une forme réduite, unidirectionnelle et adaptée au nombre de voisins de la variable traitée. Soit un ordre total $<_o$ sur les variables. Notons $parents(x_i)$ l'ensemble des parents de x_i dans le graphe associé G_P. Les variables sont rendues AdC une par une en partant de la dernière selon $<_o$ jusqu'à la première. Une variable x_i est rendue AdC en créant une contrainte de portée $parents(x_i)$ qui interdira toute instanciation localement cohérente de $parents(x_i)$ qui ne s'étend pas en une instanciation localement cohérente sur $parents(x_i) \cup \{x_i\}$. L'ensemble des arêtes de G_P est alors augmenté de toutes les arêtes possibles sur $parents(x_i)$. Si aucune des contraintes générées par AdC n'est la contrainte vide, la procédure BT appliquée selon l'ordre $<_o$ trouvera une solution sans retour arrière. Le coût de AdC est en $O(nd^{w+1})$ où w est la largeur induite de l'ordre $<_o$, c'est-à-dire la largeur de $<_o$ après application de AdC.

AdC nécessite d'avoir fixé un ordre avant de lancer la phase de cohérence locale. Si l'ordre est mauvais (largeur induite importante), le processus sera coûteux. Dechter et Pearl [1989] proposent le *tree clustering*, une technique de décomposition qui transforme un réseau quelconque en réseau structuré en arbre. L'idée est de regrouper les variables en paquets de sorte que si l'on considère chaque paquet comme une méta-variable reliée aux méta-variables dont les variables partagent des contraintes avec les siennes, le réseau résultant est structuré en arbre. Toutes les instanciations cohérentes sur chaque

paquet sont calculées et considérées comme les valeurs de la méta-variable représentant le paquet. L'avantage par rapport à AdC est qu'aucun ordre n'est fourni à l'avance, mais le calcul des instanciations cohérentes de chaque paquet peut s'avérer très coûteux en espace. De nombreuses autres techniques de décomposition ont été proposées [Gottlob *et al.*, 2000 ; Jégou et Terrioux, 2003].

On peut aussi utiliser non pas la structure du réseau de contraintes mais la structure de ses solutions. Amilhastre et al. [2002] associent un automate d'états finis à chaque contrainte et les combinent jusqu'à obtenir un automate représentant toutes les solutions du réseau. Si les solutions ne sont pas trop dispersées, l'automate peut être de taille raisonnable et de nombreuses requêtes, habituellement coûteuses, se font alors en temps linéaire ou constant.

6.7 Améliorer le backtrack chronologique

Les techniques de synthèse ou de décomposition ne fonctionnent que dans des cas très particuliers où le réseau a une structure adéquate (par exemple proche d'un arbre pour le tree clustering). La méthode standard de résolution de CSP reste donc la procédure BT, et de nombreux travaux ont consisté à apporter des améliorations à ce backtrack pour le rendre moins inefficace. Suivant Dechter et Pearl [1988], on peut grossièrement classer les types d'amélioration en quatre catégories indépendantes qui peuvent se combiner.

Les méthodes rétrospectives tirent parti des informations implicites contenues dans un échec pour éviter plus tard des branches inutiles. Les *backjumping* sélectionnent un meilleur point de retour arrière [Gaschnig, 1979 ; Dechter, 1990 ; Prosser, 1993]. Le *nogood recording* mémorise des instanciations menant à coup sûr à un échec [Dechter, 1990 ; Schiex et Verfaillie, 1993]. Je ne vais pas décrire ces méthodes par manque de place et parce qu'elles sont peu ou pas utilisées dans la génération actuelle de résolveurs à contraintes. On pourra se reporter au chapitre II.5 pour de plus amples descriptions sur ces notions.

Les méthodes prospectives appliquent un certain niveau de propagation de contraintes à chaque noeud de l'arbre de recherche pour éliminer le plus tôt possible le maximum de branches infructueuses.

Enfin, les heuristiques d'ordonnancement des variables et les heuristiques d'ordonnancement des valeurs modifient l'ordre dans lequel les variables et les valeurs sont instanciées, ce qui a une incidence sur la taille de l'espace à explorer pour trouver une solution.

6.7.1 Méthodes prospectives

Les méthodes prospectives appliquent de la propagation de contraintes à chaque noeud de l'arbre de recherche exploré par la procédure BT. La motivation est qu'il vaut mieux découvrir une incohérence une fois en haut de l'arbre de recherche qu'un nombre exponentiel de fois en bas de l'arbre. Les niveaux de propagation utilisés se contentent en général de supprimer des valeurs dans les domaines. Des niveaux de

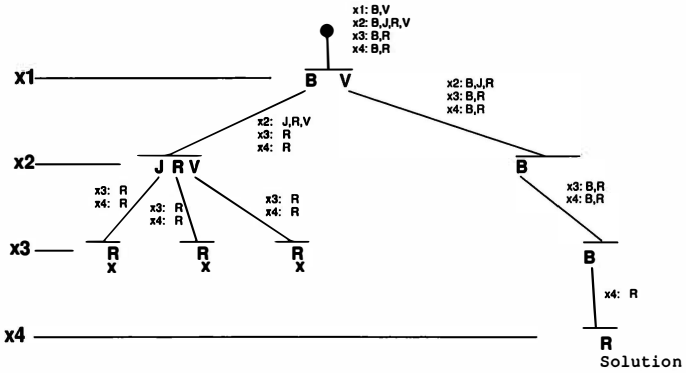


FIGURE 4 – Arbre de recherche de forward-checking sur le réseau de contraintes de l'exemple 13. Les nouveaux domaines des variables non instanciées sont donnés après chaque instanciation.

cohérence plus élevés sont en effet considérés trop coûteux à maintenir à chaque noeud de l'arbre.

Le premier algorithme de cette catégorie est le *forward-checking*, esquissé par Golomb et Baumert [1965] et formellement décrit et étudié par Haralick et Elliott [1980] pour les contraintes binaires. Forward-checking est une procédure d'exploration d'instanciations partielles, tout comme BT. La différence est qu'après chaque affectation d'une variable x_i, forward-checking supprime des domaines des variables x_k non encore instanciées et reliées à x_i par une contrainte c toutes les valeurs qui violent c. Dès qu'une de ces variables x_k a son domaine vide, la branche courante est coupée et les domaines restaurés comme avant l'affectation de x_i. Le fait de supprimer des variables futures toutes les valeurs incompatibles avec les variables déjà instanciées rend inutile la vérification de la compatibilité des valeurs de x_i avec les variables déjà instanciées (voir ligne 4 dans l'algorithme 6.1). La figure 4 représente l'arbre de recherche développé par forward-checking sur le problème de l'exemple 13, à comparer à l'arbre développé par BT en figure 2.

La question de la quantité de propagation à appliquer à chaque noeud de l'arbre de recherche a toujours fait débat. Plus de propagation signifie plus de travail à chaque noeud, mais moins de noeuds à explorer. Forward-checking a longtemps été considéré comme appliquant le bon niveau de propagation. On peut expliquer cela par le fait que les problèmes sur lesquels on testait les algorithmes dans les années 80/début 90 étaient souvent petits et pas très difficiles. Depuis le milieu des années 90, il est entendu que le niveau standard à appliquer pour résoudre des problèmes difficiles est l'arc-cohérence, et c'est ce que font majoritairement les résolveurs sous la forme de la procédure *MAC (Maintaining Arc Consistency)*, proposée par Sabin et Freuder [1994]. MAC est un algorithme qui non seulement applique l'arc-cohérence après chaque affectation, mais aussi après avoir réfuté le choix d'une valeur.

MAC est présenté dans l'algorithme 6.3. MAC applique d'abord l'arc-cohérence sur le problème donné en entrée (ligne 1). Puis, si aucun domaine n'a été vidé (ligne 2) et

Algorithme 6.3 : Maintaining arc consistency (MAC)

```
function MAC(in P : réseau) : Boolean
  begin
1   | appliquer AC sur P
2   | if P contient un domaine vide then
    |   return false
3   | if P est complètement instancié then
    |   imprimer D ; return true
4   | choisir une variable x_i non instanciée et une valeur v_i dans D(x_i)
5   | if MAC(P|_{x_i=v_i}) then
    |   return true
6   | return MAC(P|_{x_i≠v_i})
```

si on n'a pas encore atteint une solution (ligne 3), il sélectionne une variable x_i et une valeur v_i pour cette variable selon l'heuristique utilisée (ligne 4). L'appel récursif de la ligne 5 propagera le choix $x_i = v_i$ en appliquant l'arc-cohérence sur le réseau $P|_{x_i=v_i}$ où x_i a été affecté à v_i, avant de continuer la recherche dans ce sous-problème. Si ce choix se termine par un échec, l'appel récursif de la ligne 6 propagera la réfutation du choix précédent en appliquant l'arc-cohérence sur le réseau $P|_{x_i \neq v_i}$ avant de continuer la recherche. L'arbre de recherche développé par MAC sur le problème de l'exemple 13 est sans retour arrière. Dès l'affectation de B à x_1 un échec est détecté, puis l'algorithme descend directement à la solution en une seule branche.

Lecoutre et Hemery [2007] ont proposé AC3rm, un algorithme très efficace pour maintenir l'arc-cohérence pendant la recherche. Cet algorithme stocke des supports comme AC2001, utilise la multidirectionnalité des supports comme AC7 [Bessiere *et al.*, 1999], mais ne les restaure pas lors des retour arrières. Il perd donc l'optimalité d'AC2001 sur un appel, mais économise le coût des restaurations, ce qui au total est bénéfique. La plupart des résolveurs actuels sont basés sur ce principe.

6.7.2 Heuristiques

Jusqu'à maintenant j'ai considéré que la procédure de recherche de solutions sélectionnait les variables à instancier et les valeurs à leur donner dans l'ordre lexicographique. Cependant, rien ne nous oblige à respecter cet ordre arbitraire. Et il s'avère qu'en pratique, l'ordre de parcours peut avoir un impact énorme sur le temps de résolution.

Ordonnancement des variables

Parmi les heuristiques d'ordonnancement des variables on peut distinguer les ordonnancements *statiques* et les ordonnancements *dynamiques*. Les ordonnancements statiques sont calculés avant la recherche de solutions et sont gardés pendant toute la recherche. Ils sont donc en général basés sur des critères de structure du réseau, puisque celle-ci ne change pas pendant la recherche. Par exemple, *maxdegree* ordonne les variables par ordre décroissant du nombre de contraintes portant sur elles. Cela favorise

la détection d'échecs haut dans l'arbre de recherche, ce qui coûte moins cher que de les détecter au fond de l'arbre. Les ordonnancements dynamiques vont tenir compte des modifications apparues dans le réseau au cours de la recherche, c'est-à-dire en général uniquement des suppressions de valeurs des domaines. Haralick et Elliott [1980] ont proposé l'heuristique *mindom*, qui prend toujours comme prochaine variable celle qui a le plus petit nombre de valeurs restant dans son domaine. L'idée est ici de minimiser le degré de branchement dans l'arbre, et non pas d'aller vers l'échec d'abord. Bessiere et Régin [1996] ont proposé *dom/deg*, qui combine les informations tirées de la taille des domaines avec des informations sur la structure du réseau. *dom/deg* sélectionne en priorité la variable qui a le plus petit ratio taille du domaine courant sur nombre de contraintes portant sur elle.

Boussemart et al. [2004] ont proposé *dom/wdeg*, une heuristique de choix de variable *adaptative*. *dom/wdeg* est inspirée des résolveurs SAT. Comme les autres heuristiques elle tient compte de l'état courant du réseau pendant la recherche, mais aussi elle s'adapte en fonction des calculs que la procédure de recherche a déjà faits. Plus précisément, *dom/wdeg* mémorise par un compteur w_j le nombre de fois qu'une contrainte c_j a été la cause d'un échec, c'est-à-dire le nombre de fois où c'est la propagation de c_j qui a causé un domaine vide. Au départ, $w_j = 1$ pour toutes les contraintes. Pendant la recherche, chaque fois que `Revise`$(-, c_j)$ vide un domaine, w_j est incrémenté de 1. L'heuristique choisit la variable x_i qui minimise le ratio taille du domaine courant sur somme des w_j, où c_j porte sur x_i. On voit que cette heuristique démarre comme *dom/deg* mais va au cours de la recherche se focaliser de plus en plus sur les variables impliquant des contraintes difficiles à satisfaire.

Finalement, je citerai le très récent *last conflict reasoning* qui mime un backjumping simplement en modifiant le choix de la prochaine variable à instancier [Lecoutre *et al.*, 2006].

Ordonnancement des valeurs

L'ordonnancement des valeurs a suscité beaucoup moins de travaux que l'ordonnancement des variables car l'effet sur les performances est bien moindre. Une des raisons est que dès qu'un problème est difficile, on a de grande chances de passer la plus grosse partie du temps sur de gros sous-problèmes incohérents sur lesquels l'ordre de choix des valeurs dans un domaine a peu d'effet puisqu'on doit les essayer toutes. Ce constat sur les ordonnancement de valeurs peut sembler contredit par les travaux de Refalo [2004] sur les heuristiques basées sur les impacts. Mais comme ses travaux combinent ordonnancement de valeurs et de variables, on ne peut être sûr de l'effet exact de l'ordonnancement des valeurs. Tout récemment, Mehta et al. [2011] ont montré que l'ordonnancement des valeurs pouvait avoir un effet plus important que prévu lorsqu'on utilise une heuristique d'ordonnancement de variables adaptative. Ces heuristiques étant influencées par les calculs précédents de la procédure de recherche, une sélection de valeur plutôt qu'une autre aura un effet sur le prochain choix de variable.

6.8 Symétries

Lors de la modélisation d'un problème en CSP il arrive souvent que l'on définisse plusieurs variables pour représenter plusieurs occurrences d'une même entité. Par exemple deux variables x_i et x_j peuvent représenter les deux cours d'histoire qu'une classe doit avoir dans la semaine. Que dans une instanciation x_i prenne la valeur Lundi-8h et x_j la valeur Jeudi-9h ou l'inverse ne changera rien au fait que l'instanciation soit solution ou pas. On dit alors que x_i et x_j sont symétriques. La même chose peut se produire pour les valeurs. Les méthodes de détection de symétries ont pour but d'éviter que la procédure de recherche explore deux sous-arbres qui sont symétriques. Ces méthodes vont en général couper des branches de l'arbre qui contiennent peut-être des solutions mais dont on est sûr qu'il existe une solution symétrique hors des branches coupées. Freuder [1991] a soulevé le problème de la symétrie des valeurs. De nombreux travaux ont depuis traité de toutes sortes de symétries [Benhamou, 1994 ; Gent *et al.*, 2006].

6.9 Contraintes globales

Quand on modélise des problèmes réels en réseaux de contraintes, on s'aperçoit que certains « motifs » se retrouvent. Par exemple, on a souvent besoin d'exprimer le fait que toutes les variables d'un ensemble doivent prendre des valeurs différentes. La taille du motif n'est pas la même dans tous les problèmes. La contrainte `alldifferent`, qui exprime ce motif, peut impliquer un nombre quelconque de variables. Elle représente donc toute une classe de contraintes. Toute contrainte spécifiant que ses variables doivent prendre des valeurs différentes, quel que soit le nombre de variables, est une contrainte `alldifferent`. On appelle contraintes globales ces classes de contraintes définies par une fonction booléenne qui peut prendre n'importe quel nombre de variables en paramètre. Beldiceanu et al. [2005] ont construit un catalogue recensant plus de 300 contraintes globales.

Exemple 20. La contrainte globale `alldifferent`$(x_1, \ldots, x_n)$ est la classe de contraintes qui sont définies sur n'importe quelle séquence de n variables, $n \geq 2$, telles que $x_i \neq x_j$ pour tout $i, j, 1 \leq i, j \leq n, i \neq j$. `NValue`$(y, x_1, \ldots, x_n)$ est la classe de toutes les contraintes définies sur une séquence de $n+1$ variables, $n \geq 1$, telles que $|\{x_i \mid 1 \leq i \leq n\}| = y$.

Si une contrainte globale est disponible dans un résolveur, l'utilisateur peut exprimer facilement le motif associé, qui autrement pourrait être compliqué à exprimer. Ces contraintes pouvant porter sur de grands nombres de variables, il est important d'avoir un moyen de les propager autre que leur appliquer un algorithme générique d'arc-cohérence comme AC3. Rappelons que les algorithmes génériques optimaux d'arc-cohérence sont en $O(erd^r)$ pour des contraintes portant sur r variables (voir Section 6.4.1.)

Une première solution pour propager une contrainte globale à moindre coût est de la décomposer en contraintes plus simples. Décomposer une contrainte globale veut dire remplacer chacune de ses instances par un sous-réesau de contraintes d'arité bornée (et

de nouvelles variables si besoin), en restant de taille polynomiale et en préservant l'ensemble de tuples autorisés sur les variables de la contrainte d'origine. Plus formellement, une décomposition d'une contrainte globale G est une transformation *polynomiale* δ_k (k étant un entier) qui pour tout réseau $P = (X(c), D, \{c\})$ où c est la contrainte de la classe G d'arité $|X(c)|$, retourne un réseau $\delta_k(P) = (X_{\delta_k(P)}, D_{\delta_k(P)}, C_{\delta_k(P)})$ tel que $X(c) \subseteq X_{\delta_k(P)}$, $D(x_i) = D_{\delta_k(P)}(x_i)$ pour tout $x_i \in X(c)$, $|X(c_j)| \leq k$ pour tout $c_j \in C_{\delta_k(P)}$, et $sol(P)$ est égal à la projection de $sol(\delta_k(P))$ sur $X(c)$.

Exemple 21. La contrainte globale $\texttt{atleast}_{p,v}(x_1, \ldots, x_n)$ est satisfaite si et seulement si au plus p variables dans $x_1, \ldots, x_n$ valent v. Cette contrainte peut être décomposée avec $n+1$ variables additionnelles $y_0, \ldots, y_n$. La transformation contient la contrainte ternaire $(x_i = v \ \wedge \ y_i = y_{i-1} + 1) \vee (x_i \neq v \ \wedge \ y_i = y_{i-1})$ pour tout $i, 1 \leq i \leq n$, et les domaines $D(y_0) = \{0\}, D(y_n) = \{p, \ldots, n\}$ et $D(y_i) = \{0, \ldots, n\}$ pour tout $i, 1 \leq i \leq n$.

Toute la question est bien sûr de trouver des décompositions qui *préservent* l'arc-cohérence sur la contrainte d'origine. C'est à dire, pour n'importe quelle instance c de la contrainte globale G sur n'importe quel domaine initial D sur $X(c)$, on veut que pour tout domaine D' inclus dans D, l'arc-cohérence appliquée sur la décomposition supprime de D' les mêmes valeurs que l'arc cohérence sur c. La décomposition de $\texttt{atleast}_{p,v}$ donnée dans l'exemple 21 préserve l'arc-cohérence parce qu'elle a une structure Berge-acyclique.

Il n'est pas toujours possible de trouver une décomposition qui préserve l'arc-cohérence. Par exemple, Bessiere et al. [2007] ont montré que si le problème de l'existence d'un tuple valide satisfaisant la contrainte est NP-complet, alors il n'existe pas de décomposition préservant l'arc-cohérence, à moins que P=NP. C'est le cas de la contrainte globale `NValue`, pour laquelle il est donc inutile de chercher une décomposition préservant l'arc cohérence. Bessiere et al. [2009] ont montré que les contraintes NP-difficiles ne sont pas les seules pour lesquelles il n'existe pas de décomposition préservant l'arc-cohérence. Toute contrainte globale qui peut représenter une fonction non représentable par un circuit booléen monotone de taille polynomiale n'admet pas de décomposition préservant l'arc-cohérence. Un exemple est la contrainte `alldifferent`. Pour de telles contraintes, la solution est d'implémenter un algorithme spécifique qui calcule l'arc-cohérence en temps polynomial. Par exemple, Régin [1994] a utilisé le problème du couplage maximum dans un graphe biparti pour produire un algorithme polynomial qui calcule l'arc-cohérence sur la contrainte `alldifferent`.

6.10 Conclusion

Dans ce chapitre j'ai présenté diverses techniques pour résoudre le CSP. Avec l'utilisation du raisonnement par contraintes pour résoudre des problèmes réels on s'est vite aperçu que répondre oui ou non ou juste donner une solution à un réseau spécifié une fois pour toutes n'est souvent pas suffisant pour satisfaire l'utilisateur. L'utilisateur peut très bien avoir oublié de spécifier certaines contraintes qui lui viennent à l'esprit en voyant une solution qui a des défauts (par exemple un emploi du temps avec quatre

heures de maths d'affilée) ou au contraire avoir imposé trop de préférences qu'il faudra relâcher pour rendre le réseau satisfiable (par exemple enlever la contrainte de fin des cours à 16h pour les classes ayant plus de 32h hebdomadaires). Une première approche pour traiter ce problème a été les CSP dynamiques [Dechter et Dechter, 1988], où l'utilisateur interagit avec le résolveur en ajoutant ou enlevant des contraintes selon la réponse qu'il lui a fournie. Les modèles *maxCSP* [Freuder et Wallace, 1992] et CSP valués [Schiex *et al.*, 1995] peuvent aussi traiter ce problème puisqu'ils cherchent des solutions optimales selon certains critères. Ils seront développés au chapitre II.7. Le problème à décrire peut aussi contenir des incertitudes sur les valeurs de certaines variables dont l'utilisateur n'est pas maître (par exemple liées à la météo). Une solution devra satisfaire les contraintes quelles que soient les valeurs prises par ces variables, ou devra maximiser une probabilité de satisfaction. Les CSP mixtes [Fargier *et al.*, 1996] ou stochastiques [Walsh, 2002] traitent ce cas. Le problème peut être distribué sur plusieurs sites distants communicant par messages et voulant garder privées certaines données du problème. On utilise alors les CSP distribués [Yokoo *et al.*, 1998]. Certains problèmes manipulent le concept d'ensemble. Représenter des ensembles par un ensemble de variables booléennes représentant leur fonction caractéristique est une solution parfois lourde et peu efficace. Des travaux se sont intéressés à leur représentation par des variables spéciales, les variables ensemblistes [Gervet, 1994]. On peut aussi avoir à représenter des informations qui ne se discrétisent pas facilement. Dans ce cas, on utilise les CSP numériques, dans lesquels les variables prennent leurs valeurs dans des domaines composés d'intervalles sur les réels [Benhamou et Granvilliers, 2006]. Finalement, sur certaines problèmes, on peut ne pas avoir toutes les valeurs des domaines dès le début de la résolution, ou bien la portée d'une contrainte va dépendre de la valeur d'autres variables. Les *open CSP* permettent de traiter ces cas [Faltings et Macho-Gonzalez, 2005]. Il existe bien d'autres extensions qui sont plus marginales mais pourraient prendre de l'importance si leur intérêt se confirme.

Références

Amilhastre, J., Fargier, H. et Marquis, P. (2002). Consistency restoration and explanations in dynamic csps -application to configuration. *Artificial Intelligence*, 135 :199–234.

Arnborg, S. (1985). Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25 :2–23.

Beldiceanu, N., Carlsson, M. et Rampon, J. (2005). Global constraint catalog. Rapport technique T2005 :08, Swedish Institute of Computer Science, Kista, Sweden.

Benhamou, B. (1994). Study of symmetry in constraint satisfaction problems. *In Proceedings PPCP'94*, pages 249–257, Seattle WA.

Benhamou, F. et Granvilliers, L. (2006). Continuous and interval constraints. *In* Rossi, F., van Beek, P. et Walsh, T., éditeurs : *Handbook of Constraint Programming*, chapitre 16. Elsevier.

Bessiere, C., Freuder, E. et Régin, J. (1999). Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107 :125–148.

BESSIERE, C., HEBRARD, E., HNICH, B. et WALSH, T. (2007). The complexity of reasoning with global constraints. *Constraints*, 12(2) :239–259.

BESSIERE, C., KATSIRELOS, G., NARODYTSKA, N. et WALSH, T. (2009). Circuit complexity and decompositions of global constraints. *In Proceedings IJCAI'09*, pages 412–418, Pasadena CA.

BESSIERE, C. et RÉGIN, J. (1996). MAC and combined heuristics : two reasons to forsake FC (and CBJ ?) on hard problems. *In Proceedings CP'96*, pages 61–75, Cambridge MA.

BESSIERE, C., RÉGIN, J., YAP, R. et ZHANG, Y. (2005). An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165 :165–185.

BOUSSEMART, F., HEMERY, F., LECOUTRE, C. et SAIS, L. (2004). Boosting systematic search by weighting constraints. *In Proceedings ECAI'04*, pages 146–150, Valencia, Spain.

COHEN, D. et JEAVONS, P. (2006). The complexity of constraint languages. *In* ROSSI, F., van BEEK, P. et WALSH, T., éditeurs : *Handbook of Constraint Programming*, chapitre 6. Elsevier.

COOPER, M., COHEN, D. et JEAVONS, P. (1994). Characterising tractable constraints. *Artificial Intelligence*, 65 :347–361.

DEBRUYNE, R. et BESSIERE, C. (2001). Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230.

DECHTER, R. (1990). Enhancement schemes for constraint processing : Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41 :273–312.

DECHTER, R. et DECHTER, A. (1988). Belief maintenance in dynamic constraint networks. *In Proceedings AAAI'88*, pages 37–42, St Paul MN.

DECHTER, R. et PEARL, J. (1988). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34 :1–38.

DECHTER, R. et PEARL, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, 38 :353–366.

DECHTER, R. et van BEEK, P. (1997). Local and global relational consistency. *Theoretical Computer Science*, 173(1) :283–308.

DEVILLE, Y., BARETTE, O. et VAN HENTENRYCK, P. (1999). Constraint satisfaction over connected row convex constraints. *Artificial Intelligence*, 109(1-2) :243–271.

FALTINGS, B. et MACHO-GONZALEZ, S. (2005). Open constraint programming. *Artificial Intelligence*, 161(1-2) :181–208.

FARGIER, H., LANG, J. et SCHIEX, T. (1996). Mixed constraint satisfaction : a framework for decision problems under incomplete knowledge. *In Proceedings AAAI'96*, pages 175–180, Portland OR.

FIKES, R. (1970). REF-ARF : A system for solving problems stated as procedures. *Artificial Intelligence*, 1 :27–120.

FREUDER, E. (1978). Synthesizing constraint expressions. *Communications of the ACM*, 21(11) :958–966.

FREUDER, E. (1982). A sufficient condition for backtrack-free search. *Journal of the*

ACM, 29(1) :24–32.

FREUDER, E. (1985). A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4) :755–761.

FREUDER, E. (1990). Complexity of k-tree structured constraint satisfaction problems. *In Proceedings AAAI'90*, pages 4–9, Boston MA.

FREUDER, E. (1991). Eliminating interchangeable values in constraint satisfaction problems. *In Proceedings AAAI'91*, pages 227–233, Anaheim CA.

FREUDER, E. et WALLACE, R. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70.

GASCHNIG, J. (1979). Performance measurement and analysis of certain search algorithms. Rapport technique Technical Report CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh PA.

GENT, I., PETRIE, K. et PUGET, J. (2006). Symmetry in constraint programming. *In* ROSSI, F., van BEEK, P. et WALSH, T., éditeurs : *Handbook of Constraint Programming*, chapitre 10. Elsevier.

GERVET, C. (1994). Conjunto : Constraint logic programming with finite set domains. *In Proceedings ILPS'94*, pages 339–358, Ithaca NY.

GOLOMB, S. et BAUMERT, L. (1965). Backtrack programming. *Journal of the ACM*, 12(4) :516–524.

GOTTLOB, G., LEONE, N. et SCARCELLO, F. (2000). A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124(2) :243–282.

HARALICK, R. et ELLIOTT, G. (1980). Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313.

JANSSEN, P., JÉGOU, P., NOUGUIER, B. et VILAREM, M. C. (1989). A filtering process for general constraint-satisfaction problems : Achieving pairewise-consistency using an associated binary representation. *In Proceedings IEEE-ICTAI'89*, pages 420–427, Fairfax VA.

JÉGOU, P. (1993). On the consistency of general constraint-satisfaction problems. *In Proceedings AAAI'93*, pages 114–119, Washington D.C.

JÉGOU, P. et TERRIOUX, C. (2003). Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146(1) :43–75.

LAURIÈRE, J. (1978). A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10 :29–127.

LECOUTRE, C. et HEMERY, F. (2007). A study of residual supports in arc consistency. *In Proceedings IJCAI'07*, pages 125–130, Hyderabad, India.

LECOUTRE, C., SAIS, L., TABARY, S. et VIDAL, V. (2006). Last conflict based reasoning. *In Proceedings ECAI'06*, pages 133–137, Riva del Garda, Italy.

LHOMME, O. (1993). Consistency techniques for numeric CSPs. *In Proceedings IJCAI'93*, pages 232–238, Chambéry, France.

MACKWORTH, A. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8 :99–118.

MEHTA, D., O'SULLIVAN, B. et QUESADA, L. (2011). Value ordering for finding all

solutions : Interactions with adaptive variable ordering. *In Proceedings CP'11*, pages 606–620, Perugia, Italy.

MOHR, R. et HENDERSON, T. (1986). Arc and path consistency revisited. *Artificial Intelligence*, 28 :225–233.

MONTANARI, U. (1974). Networks of constraints : Fundamental properties and applications to picture processing. *Information Science*, 7 :95–132.

PROSSER, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3) :268–299.

REFALO, P. (2004). Impact-based search strategies for constraint programming. *In Proceedings CP'04*, pages 557–571, Toronto, Canada.

RÉGIN, J. (1994). A filtering algorithm for constraints of difference in CSPs. *In Proceedings AAAI'94*, pages 362–367, Seattle WA.

SABIN, D. et FREUDER, E. (1994). Contradicting conventional wisdom in constraint satisfaction. *In Proceedings PPCP'94*, Seattle WA.

SCHIEX, T., FARGIER, H. et VERFAILLIE, G. (1995). Valued constraint satisfaction problems : hard and easy problems. *In Proceedings IJCAI'95*, pages 631–637, Montréal, Canada.

SCHIEX, T. et VERFAILLIE, G. (1993). Nogood recording for static and dynamic CSPs. *In Proceedings IEEE-ICTAI'93*, pages 48–55, Boston MA.

van BEEK, P. et DECHTER, R. (1995). On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, 42(3) :543–561.

WALSH, T. (2002). Stochastic constraint programming. *In Proceedings ECAI'02*, pages 111–115, Lyons, France.

WALTZ, D. (1972). Generating semantic descriptions from drawings of scenes with shadows. Tech.Rep. MAC AI-271, MIT.

YOKOO, M., DURFEE, E., ISHIDA, T. et KUWABARA, K. (1998). The distributed constraint satisfaction problem : formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5) :673–685.

Chapitre 7

Réseaux de contraintes valués

Définissant une extension des réseaux de contraintes, les réseaux de contraintes valués (ou CSP valués) représentent un cadre unificateur pour modéliser tous les problèmes d'optimisation sur domaines finis dans lesquels le domaine des coûts (aussi appelé *structure de valuation*) peut être symbolique ou numérique mais doit être totalement ordonné. Comme les réseaux de contraintes, il s'agit d'une forme de modèle graphique défini par un ensemble de variables ayant chacune un domaine fini associé et par un ensemble de fonctions de coût impliquant un sous-ensemble des variables. Avant de lire ce chapitre, nous conseillons donc vivement au lecteur d'achever la lecture du chapitre II.6 dédié au problème de satisfaction de contraintes.

7.1 Introduction

La modélisation d'un problème combinatoire en réseau de contraintes est généralement assez naturelle : il s'agit d'identifier les variables de décision du problème, et l'ensemble des propriétés qu'elles doivent satisfaire pour former une « solution ». Dans un problème d'ordonnancement par exemple, les variables représentent les dates de démarrages des tâches et les contraintes capturent les dates de disponibilité, de livraison, les précédences et durées de tâches, les capacités des ressources,...

Dans certains cas, l'énumération des propriétés dont on exige la satisfaction ne permet pas d'aboutir à un résultat satisfaisant. C'est en particulier le cas si le problème ainsi défini n'a pas de solutions (on dit qu'il est sur-contraint). De fait, les propriétés représentées sont souvent de natures variées : il peut s'agir de propriétés physiques inviolables (une machine ne peut traiter qu'une tâche à la fois) ou de propriétés dont la satisfaction est seulement souhaitable (pour des raisons économiques par exemple : il faut que le produit soit livré à temps). La modélisation de toutes ces propriétés en contraintes dures peut amener à l'absence de solution. Inversement, négliger les propriétés dont on souhaite seulement la satisfaction mène à des solutions faisables certes, mais peu satisfaisantes.

Auteurs : Martin C. Cooper, Simon de Givry et Thomas Schiex.

Historiquement, la première généralisation du problème de satisfaction de contraintes (ou CSP, pour *Constraint Satisfaction Problem*) utilisant des fonctions de coût remonte sans doute à [Rosenfeld *et al.*, 1976] qui considère des CSP « flous » dans lesquels les contraintes, définies habituellement comme un ensemble de combinaisons de valeurs autorisées, sont remplacées par des « ensembles flous » de combinaisons autorisées, chaque combinaison ayant un degré d'appartenance (entre 0 et 1) à la relation floue. Les ensembles flous généralisant les ensembles classiques, il est ainsi possible d'exprimer des contraintes classiques (degrés d'appartenance 0 et 1 seulement) mais aussi des combinaisons plus ou moins « possibles ». On cherche alors à trouver une valeur des variables du problème de façon à éviter d'utiliser des combinaisons de valeurs peu possibles. Plus précisément, on cherche à maximiser le degré d'appartenance minimum des combinaisons utilisées (un problème max-min). Des algorithmes de propagation de contraintes dédiés ont été ainsi proposés par [Rosenfeld *et al.*, 1976].

Cette extension a été suivie de différentes autres extensions (réseaux de contraintes additifs [Shapiro et Haralick, 1981] ou partiels [Freuder et Wallace, 1992], réseaux de contraintes possibilistes [Schiex, 1992], réseaux de contraintes lexicographiques [Fargier *et al.*, 1993], réseaux de contraintes probabilistes [Fargier et Lang, 1993],...) qui utilisent des échelles de coûts différentes (entiers, réels, symboles) et qui jugent de la qualité d'une solution en combinant les coûts des combinaisons utilisées avec un opérateur spécifique dans chaque cas, donnant ainsi une sémantique particulière aux « coûts » : coûts additifs (financiers, énergétiques,...), priorités, probabilités,... Dans chacun des cas, des algorithmes dédiés ont été proposés, plus ou moins facilement selon les cas.

7.2 Réseaux de contraintes valuées

Pour capturer le cœur des propriétés et des algorithmes communs à toutes ces propositions, mais aussi pour mieux comprendre pourquoi certaines extensions sont plus simples à traiter que d'autres, deux cadres génériques ont été proposés : les CSP à semi-anneaux ou SCSP [Bistarelli *et al.*, 1995, 1997] et les CSP valués ou VCSP [Schiex *et al.*, 1995]. Dans les deux cas, les contraintes des CSP, qui séparent les combinaisons de valeurs en combinaison autorisées ou interdites, sont remplacées par des fonctions de coût (ou contraintes valuées) permettant une comparaison plus fine et graduée des combinaisons de valeurs. Les deux cadres VCSP et SCSP s'appuient sur un domaine de coût abstrait, permettant l'expression de l'autorisation et de l'interdiction mais aussi de niveaux intermédiaires. Le domaine des coûts est équipé dans chaque cas d'une structure algébrique adaptée aux problèmes que l'on souhaite capturer.

Le lien entre VCSP et SCSP est simple : les VCSP sont des SCSP utilisant une structure de coût totalement ordonnée. L'essentiel des propriétés et algorithmes ayant été développés dans ce cas totalement ordonné, qui est aussi le plus fréquent en pratique, nous nous restreindrons donc au cadre VCSP. Dans ce cas, la structure de coûts utilisée est appelée « structure de valuation ».

7.2.1 Structure de valuation

Une structure de valuation V est définie par un quintuplet $V = \langle E, \oplus, \prec, \bot, \top \rangle$ où E est l'ensemble des coûts utilisables, $\oplus$ est un opérateur de combinaison de coûts et où $\prec$ définit un ordre total sur E. $\bot$ et $\top$ dénotent respectivement l'élément minimum et l'élément maximum de E ($\bot \prec \top$). On exige de plus :

- que $\oplus$ soit commutatif ($\forall \alpha, \beta \in E, \alpha \oplus \beta = \beta \oplus \alpha$) et associatif ($\forall \alpha, \beta, \gamma \in E, \alpha \oplus (\beta \oplus \gamma) = (\alpha \oplus \beta) \oplus \gamma$).
- que $\oplus$ soit monotone ($\forall \alpha, \beta, \gamma \in E, (\alpha \preccurlyeq \beta) \Rightarrow ((\alpha \oplus \gamma) \preccurlyeq (\beta \oplus \gamma))$)
- que $\bot$ soit l'élément neutre pour $\oplus$ ($\forall \alpha \in E, \bot \oplus \alpha = \alpha$) et $\top$ l'élément absorbant ($\forall \alpha \in E, \top \oplus \alpha = \top$).

L'associativité et la commutativité capturent le fait que seul l'*ensemble* des coûts combinés par $\oplus$ a un sens : le résultat de la combinaison ne doit pas dépendre de l'ordre dans lequel elle est réalisée. La monotonie garantit que la diminution d'un coût dans une combinaison ne peut mener à une augmentation du coût global. Les deux derniers axiomes (élément neutre et absorbant) permettent de capturer les contraintes dures des réseaux de contraintes comme des fonctions de coût utilisant uniquement les coûts $\bot$ (combinaisons autorisées) et $\top$ (combinaisons interdites).

Cette structure algébrique est proche de la structure de co-norme triangulaire, souvent étudiée dans un contexte numérique, où $E = [0, 1]$ [Klement *et al.*, 2000]. Elle est aussi appelée tomonoïde dans le domaine de l'algèbre.

Le tableau 1 donne quelques exemples de structures de coût qui sont des structures de valuation. Différentes propriétés ont une influence importante sur les algorithmes de traitement des VCSP :

1. l'idempotence de $\oplus$ ($\forall a \in E, a \oplus a = a$) : on sait que le seul opérateur $\oplus$ remplissant cette condition (et les axiomes d'une structure de valuation) est max (pour l'ordre $\succ$) qui correspond aussi au $\wedge$ (et logique) des CSP classiques. Cette propriété simplifie considérablement les procédures d'inférence car, comme le fait la propagation de contraintes dans les CSP classiques, elle permet d'ajouter des informations implicites dans un réseau de façon explicite, en préservant son sens. Elle conduit par contre à des effets désagréables en termes de modélisation (l'« effet de noyade » des VCSP possibilistes [Schiex, 1992]). Les CSP classiques et possibilistes/flous sont idempotents.
2. la monotonie stricte (SM) de $\oplus$ ($\forall a, b, c \in E, (a \prec c) \wedge (b \neq \top) \Rightarrow (a \oplus b) \prec (c \oplus b)$) garantit que dans une situation où les coûts ne sont pas déjà intolérables (égaux à $\top$), toute croissance de coût locale entraîne une croissance de coût globale. C'est une propriété attirante en termes de modélisation mais qui complexifie sensiblement tous les mécanismes d'inférence. En effet, tout ajout d'information à un réseau modifie ici le sens du réseau. Les VCSP additifs, pondérés, probabilistes, lexicographiques sont strictement monotones.

C'est pour lutter contre la « perte d'équivalence » des opérateurs non idempotents qu'un opérateur $\ominus$ de pseudo-différence a été introduit dans les structures de valuation. Ces structures sont alors dites « justes » [Schiex, 2000 ; Cooper et Schiex, 2004]. On exige que pour tout $\beta \preccurlyeq \alpha$, il existe un γ maximum tel que $\beta \oplus \gamma = \alpha$ (un tel γ peut ne pas être défini dans des structures de valuation non finies). Cet élément γ est noté $\alpha \ominus \beta$

et définit l'opérateur $\ominus$. Cet axiome supplémentaire est satisfait par la majorité des structures de coût utilisées en pratique (ou peut être satisfait en plongeant les structures existantes dans des structures plus larges et qui sont justes [Cooper, 2003 ; Cooper et Schiex, 2004]). L'opérateur de pseudo-différence ainsi défini permet de restaurer l'équivalence après l'ajout d'une information implicite dans le réseau (voir section 7.5) et ainsi d'étendre la notion de propagation de contraintes à des cas non idempotents en conservant l'essentiel de ses propriétés (dont la préservation de l'équivalence).

Nom	E	$a \oplus b$	$\prec$	$\bot$	$\top$	SM	Idemp.	$a \ominus b$
Classique	$\{v, f\}$	$a \wedge b$	$v < f$	v	f	oui	oui	a
Additif	$\bar{\mathbb{N}}$	$a + b$	$<$	0	$+\infty$	oui	non	$a - b$
Pondéré	$\{0, \ldots, m\}$	$\min(m, a + b)$	$<$	0	m	non	non	$(a = m ? m : a - b)$
Probabiliste	$[0, 1]$	$a \times b$	$>$	1	0	oui	non	a/b
Possibiliste	$[0, 1]$	$\max(a, b)$	$<$	0	1	non	oui	$\max(a, b)$
Flou	$[0, 1]$	$\min(a, b)$	$>$	1	0	non	oui	$\min(a, b)$
Lexico	$[0, 1]^*$	$\cup$	$\succ_{lex}$	$\varnothing$	$\top$	oui	non	NA

TABLE 1 – Quelques structures de valuation classiques. SM indique le caractère strictement monotone de $\oplus$, Idemp. son caractère idempotent. La définition de $\ominus$ est précisée quand elle existe et est simple. Dans le cas lexicographique, $[0, 1]^*$ représente l'ensemble des multi-ensembles de réels entre 0 et 1, $\cup$ représente l'union de multi-ensembles et $\succ_{lex}$ la comparaison lexicographique des séquences obtenues par le tri décroissant des multi-ensembles. Ainsi $\{1, 0.2, 1\} \succ_{lex} \{0.8, 1, 0.8\}$ car dans l'ordre lexicographique $(1, 1, 0.2)$ est plus grand que $(1, 0.8, 0.8)$. Bien que la structure lexicographique ne soit pas juste, il est possible de la plonger dans une structure plus grande qui elle est juste. Voir [Cooper et Schiex, 2004] pour plus de détails.

Peu de temps après, [Cooper et Schiex, 2004 ; Cooper, 2005] ont montré que toute structure de valuation juste et dénombrable peut s'analyser sous la forme d'un empilement de structures de valuations additives/pondérées, qui interagissent entre elles comme une structure idempotente (et donc utilisant $\oplus = \max$). Le cas $\oplus = \max$ étant très proche du cas des CSP classiques [Meseguer *et al.*, 2006, page 293], l'essentiel des travaux se focalisent depuis sur le cas pondéré.

7.2.2 Réseaux de fonctions de coût

Comme les réseaux de contraintes classiques, les réseaux de contraintes valuées (ou réseaux de fonctions de coût) définissent une classe de modèle graphique.

Un réseau $\mathcal{P}$ est défini par un quadruplet $\langle X, D, F, V \rangle$ où X est une collection de variables, D étant formé de l'ensemble des domaines D_i de chacune des variables $x_i \in X$. V est une structure de valuation et F un ensemble de fonctions de coût locales à valeurs dans l'ensemble E de la structure de valuation V. Chaque fonction de coût $f_S \in F$ (appelée aussi contrainte valuée ou molle, dérivé de l'anglais *soft constraint*) est définie sur un ensemble $S \subseteq X$ de variables et spécifie le coût associé à chacune des combinaisons de valeurs (ou *tuple*) que peuvent prendre les variables de S. S est appelé la portée de f_S et $|S|$ est son arité (nombre de variables qu'elle implique). On

note $\ell(S) = \prod_{x_i \in S} D_i$ le produit cartésien des domaines des variables de S, définissant l'ensemble des combinaisons de valeurs possibles pour les variables de S. Pour une combinaison de valeurs $t \in \ell(S)$ et $T \subseteq S$, on note $t[T]$ la combinaison des valeurs que prennent les variables de T dans t (appelée projection de t sur S'). Par simplicité, une fonction de coût unaire (impliquant une variable x_i) sera notée f_i et une fonction de coût binaire f_{ij}. On introduit également souvent une fonction de coût $f_\varnothing$, n'impliquant aucune variable et définissant un coût constant.

L'ensemble des variables X et l'ensemble des portées des différentes fonctions de coût $f_S \in F$ définissent un hyper-graphe dont les sommets sont les variables de X et les hyper-arêtes les différentes portées. Dans le cas où l'arité maximum est limitée à 2, cet hyper-graphe est un graphe, appelé graphe du problème (d'où le nom de modèle graphique).

Cet hypergraphe ne capturant que la structure du problème mais pas les domaines ou les définitions de fonctions de coût, une seconde représentation plus fine, appelée graphe de micro-structure, est souvent utilisée pour représenter les VCSP (binaires). Ici, les sommets représentent les valeurs des domaines de chacune des variables. Le sommet associé à la valeur $a \in D_i$ est valué par $f_i(a)$ si ce coût est défini et différent de $\bot$. De même une arête relie les sommets $a \in D_i$ et $b \in D_j$ si $f_{ij}(a, b)$ est défini et différent de $\bot$. Cette arête est alors étiquetée par la valuation $f_{ij}(a, b)$.

L'ensemble des fonctions de coût locales f_S d'un tel réseau définit implicitement une fonction de coût sur l'ensemble X des variables du problème. Cette fonction de coût globale (ou jointe) est simplement définie en combinant, via l'opérateur de combinaison $\oplus$, l'ensemble des coûts produits par chacune des fonctions de coût locales. Elle permet de définir la fonction de coût globale d'un VCSP $\mathcal{P}$

$$Val_{\mathcal{P}}(t) = \bigoplus_{f_S \in F} f_S(t[S])$$

Le problème central des VCSP est de trouver une affectation préférée t de toutes les variables i.e., telle que $Val_{\mathcal{P}}(t)$ soit de coût minimum (l'ensemble des coûts est supposé totalement ordonné). Il s'agit d'un problème NP-difficile (avec un problème de décision associé qui est NP-complet).

On dira que deux réseaux $\mathcal{P}$ et $\mathcal{P}'$, définis sur les mêmes variables, sont équivalents s'il définissent la même fonction de coût globale : $Val_{\mathcal{P}} = Val_{\mathcal{P}'}$.

Pour donner du corps à ces définitions, nous considérons par exemple un VCSP additif. Dans cet exemple, l'ensemble des coûts est l'ensemble $\bar{\mathbb{N}}$ des entiers naturels complété avec $+\infty$, l'opérateur de combinaison $\oplus$ est l'addition entière, l'élément minimum de l'ensemble est 0 et l'élément maximum $+\infty$. Nous considérons un VCSP formé de trois variables $X = (x_1, x_2, x_3)$. Le domaine de ces variables est composé de deux valeurs $D_1 = D_2 = D_3 = \{a, b\}$. On préfère que les variables prennent la valeur a si possible et l'on exige que $x_1 = x_3$, que $x_1 \neq x_2$ et que $x_2 \neq x_3$. Ce problème se modélise via six fonctions de coût. Les fonctions unaires $f_1 = f_2 = f_3$ sont identiques et définies par $f_i(a) = 0$ et $f_i(b) = 1$, favorisant ainsi l'utilisation de la valeur a. Trois fonctions f_{13}, f_{12} et f_{23} vont capturer les exigences $x_1 = x_3$, $x_1 \neq x_2$ et $x_2 \neq x_3$. Comme il s'agit d'exigences absolues (des contraintes), les fonction de coût correspondantes utilisent uniquement deux coûts spécifiques : le coût 0 (élément neutre pour $\oplus = +$,

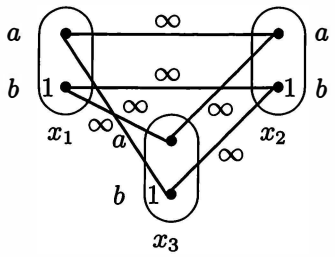


FIGURE 1 – La micro-structure du problème.

minimum dans l'échelle des coûts) sera associé aux combinaisons de valeurs autorisées et le coût $+\infty$ (absorbant pour $\oplus = +$, maximum dans l'échelle des coûts) sera associé aux combinaisons interdites. Les tables ci-dessous décrivent les trois fonctions.

f_{12}	a	b
a	$+\infty$	0
b	0	$+\infty$

f_{23}	a	b
a	$+\infty$	0
b	0	$+\infty$

f_{13}	a	b
a	0	$+\infty$
b	$+\infty$	0

TABLE 2 – Trois fonctions de coûts représentant des contraintes.

Ce même réseau est représenté dans la figure 1 par sa micro-structure.

L'affectation (a, a, a) a un coût global égal à $+\infty$ car $f_{12}((a, a)) = +\infty$. Une solution optimale est (a, b, a), de coût 1 (coût généré par f_2 car $x_2 = b$).

7.2.3 Opérations sur les fonctions de coût

Une boîte à outil très puissante sur les réseaux de fonctions de coût peut être construite en utilisant simplement trois opérations simples. Il s'agit de l'instanciation (ou conditionnement), de la combinaison et de l'élimination de variable[1].

A partir d'une fonction de coût f_S, il est possible d'instancier une variable $x_i \in S$ avec l'une des valeurs a de son domaine D_i. On obtient alors une fonction de coût sur $T = S - \{x_i\}$ et définie par $g_T(t) = f_S(t \cup \{x_i = a\})$. Dans notre exemple précédent, instancier x_1 à la valeur a transforme la fonction f_{12} en une fonction unaire g_2 telle que $g_2(a) = f_{12}(a, a) = +\infty$ et $g_2(b) = f_{12}(a, b) = 0$. Il faut noter que si l'on instancie toutes les variables d'une fonction de coût, on obtient une fonction de portée vide, comme $f_\varnothing$. Cette opération a une complexité négligeable (on accède directement à une partie de la fonction de coût originale).

La seconde opération est une opération sur deux fonctions de coût f_S et $g_{S'}$. C'est l'équivalent de l'opération de jointure sur les relations définissant un CSP : elle combine deux fonctions de coût en une seule fonction, équivalente à la combinaison par $\oplus$ des

1. Cette dernière opération est aussi souvent appelée projection. Nous avons évité d'utiliser cette terminologie du fait des possibles confusions qu'elle peut engendrer avec la notion homonyme introduite dans la section 7.5.

deux fonctions d'origine. Cette fonction porte sur les variables de $S \cup S'$ et est définie par $(f_S \oplus g_{S'})(t) = f_S(t[S]) \oplus g_{S'}(t[S'])$. La production de la combinaison de deux fonctions est exponentielle en temps et en espace (en $O(d^{|S \cup S'|})$). Dans notre exemple précédent, la combinaison de f_1 et de f_{12} définit une fonction g_{12}, définie dans la table 3. Il faut noter que l'on peut remplacer un ensemble de fonctions par leur combinaison sans modifier le sens du réseau de fonctions de coût (la fonction de coût globale $Val_{\mathcal{P}}$).

$f_1 \oplus f_{12}$	a	b	(x_1)
a	$+\infty$	1	
b	0	$+\infty$	
(x_2)			

TABLE 3 – La combinaison de f_1 et f_{12}.

Enfin, l'opération d'élimination, consiste à résumer l'information des coûts d'une fonction f_S sur un sous-ensemble $T \subset S$ de variables en faisant l'hypothèse que les variables de $S - T$ éliminées sont instanciées dans chaque cas au mieux (de façon à minimiser le coût).

La fonction $f_S[T]$ de f_S sur T qui résulte de l'élimination des variables de $S - T$ est une fonction de coût $g_T(u) = \min_{v \in \ell(S-T)} f_S(u \cup v)$. L'élimination nécessitant de parcourir, dans le cas général, tout le domaine de f_S, cette opération a une complexité temporelle en $O(d^{|S|})$. Le résultat occupe un espace en $O(d^{|T|})$. Si $S-T$ est un singleton $\{x_i\}$, on notera $f_S[-x_i] = f_S[S - \{x_i\}]$, le résultat de l'élimination de la seule variable x_i.

Si $f_S[T]$ a un coût α sur une affectation des variables de T, cela signifie qu'il est possible de compléter cette affectation sur les variables de $S - T$ pour construire une affectation des variables de S qui a le même coût α pour f_S. Ainsi, dans notre exemple, l'élimination de x_2 dans $f_{12}[-x_2]$ donne une fonction unaire sur $\{x_1\}$, partout égale à 0. Par exemple, il est possible d'étendre $x_1 = a$ (de coût nul pour $f_{12}[-x_2]$) avec $x_2 = b$ qui forme une affectation de même coût (nul) pour f_{12}.

Il est bien sûr possible d'exploiter dans les VCSP des fonctions de coût qui sont représentées par d'autres moyens que des tables (représentation analytique comme dans $f(x) = x^2$ ou utilisation de structures de données plus compactes comme des automates ou des diagrammes de décision par exemple).

7.2.4 Lien avec d'autres approches

Globalement, les CSP valués définissent donc un cadre générique, permettant d'exprimer et de combiner des préférences (numériques ou symboliques, voir le chapitre I.6) et des propriétés requises (ou contraintes), exprimées de façon locale sur quelques variables, au travers de fonctions de coût. Des liens forts existent avec de nombreux formalismes issus de l'intelligence artificielle mais aussi de la recherche opérationnelle.

Ainsi, tout comme la logique propositionnelle et le problème SAT définissent un cas particulier de CSP, les VCSP permettent de capturer les variantes de SAT telles que le problème MaxSAT (maximiser le nombre de clauses satisfaites par une interprétation) éventuellement pondéré ou partiel (les clauses non pondérées étant considérées

comme dures). Ces problèmes sont eux-mêmes très proches des problèmes d'optimisation pseudo-booléenne quadratique dans lesquels on souhaite trouver le minimum d'un polynôme du second degré à variables 0/1 [Boros et Hammer, 2002].

En tant que modèle graphique, les VCSP permettent aussi de capturer les réseaux bayésiens (voir chapitre II.8 : des fonctions de coût additives peuvent être obtenues en représentant les probabilités conditionnelles par l'opposé de leur logarithme), les champs de Markov [Chellappa et Jain, 1993] et les graphes de facteurs [Kschischang *et al.*, 2001]. Les outils VCSP ont montré d'excellentes performances pour résoudre le problème de recherche d'explications de probabilité maximum (MPE) dans ces domaines lors d'évaluations internationales. Cependant, dans ces formalismes, les problèmes ne se limitent généralement pas à l'optimisation mais concernent aussi le comptage (ou l'intégration discrète) pour lesquels peu de travaux VCSP existent.

Plus largement, d'autres formalismes d'expression de préférences et d'utilités tels que les modèles GAI (*Generalized Additive Independence* [Bacchus et Grove, 1995]) ou les réseaux « *Ceteris Paribus* » (CP-nets [Domshlak *et al.*, 2003]) peuvent se rapprocher des VCSP et parfois profiter des techniques et outils de résolution développés dans ce cadre.

7.3 Programmation dynamique et élimination de variables

Une première approche pour la résolution du problème VCSP consiste à s'appuyer sur des techniques de programmation dynamique non sérielle [Bertelé et Brioshi, 1972], aussi appelées algorithmes d'élimination de variables et plus connues dans la communauté des contraintes sous les noms de *Bucket Elimination* [Dechter, 1999] et *Cluster Tree Elimination* [Dechter, 2003]. Ces techniques, qui ont été initialement adaptées au cadre CSP, s'étendent très naturellement au traitement de VCSP. Nous ne présenterons ici que la technique la plus simple d'élimination de variables procédant variable par variable, connue sous le nom de « *Bucket Elimination* ». Les techniques de programmation dynamique s'étendent naturellement à la résolution d'une variété de problèmes dépassant les seuls problèmes d'optimisation (problèmes de dénombrement par exemple. Voir par exemple [Aji et McEliece, 2000 ; Shafer et Shenoy, 1988] et [Dubois et Prade, 1991] pour le cas possibiliste).

Si l'on considère un VCSP défini par un ensemble de variables X et un ensemble de fonctions de coût F, l'algorithme d'élimination de variable consiste à réduire itérativement le nombre de variables du problème en préservant le coût d'une solution optimale. *In fine*, on sera réduit à un problème ayant 0 variables et une fonction de coût $f_\varnothing$ égale à l'optimum.

Sans perte de généralité, nous supposons que l'algorithme élimine les variables dans l'ordre $x_1, \ldots, x_n$. L'élimination d'une variable x_i peut se décrire comme un processus en trois étapes :

1. on collecte dans un ensemble K_i toutes les fonctions de coût reliant la variable x_i à des variables qui la suivent dans l'ordre d'élimination.
2. on combine toutes ces fonctions de coût en une seule fonction de coût en utilisant

$\oplus$. Le résultat est une fonction de coût g_S impliquant x_i et toutes les variables reliées à x_i et qui la suivent dans l'ordre d'élimination. Cette combinaison est équivalente à l'ensemble de fonctions de coût K_i.

3. on élimine ensuite la variable x_i de g_S et on ajoute la fonction obtenue au réseau. Cette nouvelle fonction de coût résume l'effet de toutes les fonctions de coût de K_i sur ces variables. On retire donc du réseau les fonctions de coût de K_i et la variable x_i qui n'est plus impliquée dans aucune fonction de coût : la variable a été éliminée.

Les deux dernières étapes sont généralement effectuées simultanément ce qui permet de réduire un peu la complexité spatiale d'une élimination. Le problème obtenu après l'élimination de x_i a une nouvelle fonction de coût mais a une variable en moins et l'ensemble des fonctions de coût de K_i a été supprimé. Il a cependant le même coût optimum que le problème d'origine du fait des bonnes propriétés des opérateurs de combinaison et d'élimination.

L'ordre dans lequel les variables sont éliminées peut avoir une forte influence sur l'efficacité du processus complet. Il est d'ailleurs possible d'estimer la complexité globale de la procédure. En effet, chaque élimination crée une fonction de coût nouvelle mais dont la portée est connue. Comme chaque étape est exponentielle dans le nombre de variables voisines de la variable x_i éliminée et qui suivent x_i dans l'ordre d'élimination, on peut simuler le processus et estimer le coût global sans effectuer réellement les calculs. C'est l'étape pour laquelle le nombre de voisins w de la variable x_i éliminée qui suivent x_i dans l'ordre d'élimination est maximum qui fixe la complexité (spatiale et temporelle), exponentielle en w. Ce paramètre w est appelé largeur induite ou largeur d'arbre (*treewidth*) du graphe pour l'ordre d'élimination donné. Sa minimisation définit un problème NP-complet mais de bonnes heuristiques existent [Bodlaender et Koster, 2008].

Ces procédures d'élimination sont utilisées dans de nombreux domaines et permettent de traiter des problèmes d'optimisation mais aussi de dénombrement par exemple. Du fait de leur complexité spatiale, elles restent réservées en général à des problèmes pour lesquels un w pas trop grand existe.

7.3.1 Élimination de variable partielle ou « mini-buckets »

Les procédures d'élimination de variable sont souvent inapplicables du fait de leur complexité spatiale (et temporelle) lorsqu'il faut éliminer une variable ayant un nombre important de voisins (non éliminés). Pour tenter de contrôler cette complexité, en renonçant à l'optimalité de la procédure, il est possible d'utiliser une approche simple : si l'ensemble des fonctions de coût K_i implique un trop grand nombre de variables en sus de x_i, on peut simplement partitionner K_i en une série de sous-ensembles disjoints $K_i = \cup K_{ij}$ qui chacun implique un nombre de variables borné par un entier z fixé a priori.

On pourra ensuite traiter chacun des ensembles K_{ij} séparément (en appliquant les phases de combinaison et d'élimination sur chaque ensemble). Ces ensembles impliquant un nombre de variables borné, les opérations ont une complexité spatiale et temporelle bornée (exponentielle en z). Naturellement, les bonnes propriétés de l'élimination de

variable sont perdues car ce n'est pas forcément la même valeur de x_i qui optimise chacun des K_{ij} : le coût final obtenu après une série de « mini-éliminations » n'est qu'un minorant du coût optimum. Les fonctions de coût intermédiaires générées lors du traitement des K_{ij} permettent donc de construire un minorant sur le coût d'une solution optimum exploitable dans les algorithmes de recherche arborescente [Dechter et Rish, 2003]. La grande force de cette approche est que la force des minorants est ajustable via le paramètre z. Il est donc possible de l'adapter au problème à traiter. C'est aussi un inconvénient en pratique car aucun résultat théorique ou empirique ne permet de choisir *a priori* une bonne valeur de z (compromis temps/qualité).

7.4 Recherche de solutions optimales

Étant donné un VCSP, le problème central habituellement considéré consiste à identifier une affectation des variables de valuation minimum. Une des approches les plus courantes consiste à utiliser, comme dans les CSP, un algorithme de recherche arborescente en profondeur d'abord. L'algorithme « Backtrack » des CSP est remplacé par un algorithme de type « Séparation-Évaluation » ou *Branch and Bound*. Cet algorithme suppose qu'il est possible, par une méthode qu'il faudra spécifier, de calculer un minorant $lb_{\mathcal{P}}$ du coût d'une solution optimale d'un VCSP quelconque. En partant d'un VCSP initial, l'algorithme consiste à comparer le minorant $lb_{\mathcal{P}}$ au coût de la meilleure solution connue. Si $lb_{\mathcal{P}}$ atteint ou dépasse ce coût, il est inutile de poursuivre la recherche car il est impossible de faire mieux que la meilleure solution connue.

Sinon, il faut simplifier le problème. De façon usuelle, on choisit une variable x_i du problème et l'on exploite le fait qu'elle doit prendre une des valeurs de son domaine. On instancie donc x_i avec une valeur choisie dans son domaine (conditionnement). Cela réduit les portées de certaines fonctions de coût et diminue le nombre de variables du problème. On continue récursivement en choisissant une nouvelle variable. Si le minorant atteint le coût de la meilleure solution connue, on reconsidère le dernier choix effectué. S'il est possible d'instancier toutes les variables sans que le minorant n'atteigne le coût de la meilleure solution, on a trouvé une nouvelle meilleure solution et la recherche se poursuit en actualisant le coût de la meilleure solution connue.

Lorsque l'on reconsidère un choix, plusieurs schémas sont possibles. Il est possible d'essayer une nouvelle valeur pour la dernière variable instanciée, on parle alors de branchement k-aire. On peut aussi simplement imposer que la dernière variable affectée ne prenne pas la valeur déjà essayée (en la retirant du domaine). On parle alors de branchement binaire, théoriquement plus puissant. D'autres stratégies sont possibles.

Dans tous les cas, le choix de la prochaine variable à instancier ou de sa prochaine valeur peut avoir une influence extrême sur l'efficacité de l'algorithme et différentes heuristiques ont été proposées. En ce qui concerne le choix de la variable, comme toutes les variables doivent être affectées, on choisira celle qui a le plus de chance de permettre de détecter rapidement l'impossibilité d'amélioration du coût (variable impliquant un nombre important de fonctions de coût, un domaine de petite taille...). Ce choix a une importance considérable et peut permettre de capturer une forme de retour-arrière intelligent [Lecoutre *et al.*, 2006]. Dans le cadre CSP, le choix de valeur reste difficile à faire et est habituellement considéré comme dépendant du domaine d'application. Dans

les VCSP, les coûts générés par les fonctions de coût unaires impliquant la variable courante permettent de guider la recherche et on choisira donc une valeur minimisant ces coûts.

L'efficacité pratique de cet algorithme exact dépend de la qualité du minorant lb_P utilisé. Un minorant simple peut être obtenu en combinant l'ensemble des fonctions de coût de portée vide créées par les instanciations. Cela définit un coût constant qui forme un minorant. Mais ce minorant (correspondant à l'algorithme de Backtrack des CSP) est trop naïf. En pratique, le minorant doit être facile à calculer mais être le plus proche possible de l'optimum, deux objectifs contradictoires puisque le problème d'optimisation des VCSP est NP-difficile.

Historiquement, des procédures de calcul de minorants *ad hoc* et assez variées ont été proposées. Mais depuis une dizaine d'année, on observe une convergence d'approche avec les CSP classiques : ce sont des procédures de « propagation de contraintes » (aussi appelé filtrage par cohérence locale) généralisées aux VCSP qui sont utilisées pour calculer un minorant. Ces méthodes opèrent en reformulant un VCSP donné $\mathcal{P}$ en un autre VCSP $\mathcal{P}'$ sur les mêmes variables et qui doit lui être équivalent ($Val_{\mathcal{P}} = Val_{\mathcal{P}'}$) mais avec une fonction de coût $f_\varnothing$ accrue et qui définit le minorant. Cette approche par reformulation a l'avantage d'être incrémentale : la version reformulée peut à nouveau être utilisée pour d'autres traitements. C'est l'hybridation des algorithmes de recherche arborescente et des algorithmes de propagation de contraintes valuées (pour calculer le minorant) qui donnent en général les meilleurs résultats pratiques pour résoudre le problème VCSP sur des instances variées de problèmes. Dans certains cas, l'utilisation de minorants produit par l'approche « mini-buckets » ou par d'autres algorithmes [Verfaillie *et al.*, 1996] peut fournir une alternative intéressante.

D'autres procédures arborescentes peuvent être définies : au lieu de parcourir l'arbre de recherche en profondeur d'abord, on peut s'intéresser prioritairement à l'exploration de la meilleure solution partielle courante (recherche en meilleur d'abord), une approche qui a de bonnes propriétés en termes de taille de l'arbre exploré mais qui peut être très gourmande en espace. Des schémas plus complexes ont été considérés, inspirés de la programmation dynamique [Verfaillie *et al.*, 1996 ; Terrioux et Jegou, 2003 ; Marinescu et Dechter, 2005 ; de Givry *et al.*, 2006], ou d'autres encore destinés à fournir des méthodes qui donnent rapidement des solutions de bonne qualité (recherche partielle [de Givry et Jeannin, 2006]).

Les méthodes de recherche locale et métaheuristiques [Aarts et Lenstra, 1997] offrent une alternative facile à mettre en œuvre pour tenter de répondre au problème VCSP, sans garantie d'optimalité toutefois, et avec la difficulté que peut générer l'existence de contraintes dures (formant des barrières dans le paysage). Ces méthodes d'optimisation sont très génériques et peu de développements spécifiques aux CSP valués ont été proposés dans le domaine, si ce n'est dans le domaine des méthodes à voisinage large/variable [Loudni et Boizumault, 2006] pour lesquelles les méthodes complètes VCSP peuvent être utilisées pour explorer un grand voisinage.

7.5 La propagation de contraintes valuées

Dans un CSP classique il faut trouver une affectation à n variables qui satisfait un ensemble de contraintes, où chaque contrainte précise les combinaisons de valeurs que l'on peut affecter à un sous-ensemble des variables. La notion de propagation de contraintes permet de réduire la taille des domaines, de renforcer des contraintes ou encore d'élaguer l'arbre de recherche via la détection d'incohérences locales entre les contraintes et les domaines. Cette notion est omniprésente dans les algorithmes de résolution de CSP. Par exemple, pendant la recherche d'un 3-coloriage d'un graphe, dès que l'on détecte l'existence d'un sommet dont trois sommets voisins sont affectés à trois couleurs différentes, on peut élaguer cette branche de l'arbre de recherche.

Les opérations classiques de propagation de contraintes dans les CSP se traduisent par la propagation d'interdictions implicites correspondant à un coût de $\top$ dans les CSP valués. L'ensemble des notions de cohérence locale dans les VCSP inclut donc naturellement des notions héritées du cadre CSP, mais s'enrichit considérablement à travers des notions faisant intervenir un minorant ou un majorant.

Dans les VCSP dits pondérés, lorsque l'opérateur d'agrégation de coûts $\oplus$ est $+_m$, l'addition plafonnée par un majorant m

$$\forall \alpha, \beta \in \{0, 1, \ldots, m\} \qquad \alpha +_m \beta \; = \; \min\{\alpha + \beta, m\}$$

de nouvelles contraintes dures peuvent faire leur apparition lors de la propagation de coûts. Par exemple, si $f_i(a) +_m f_\varnothing = m$, alors augmenter $f_i(a)$ à m (ce qui revient à éliminer a du domaine D_i car $m = \top$) laisse invariant la fonction $Val_\mathcal{P}$. De façon similaire, si $f_{ij}(a, b) +_m f_i(a) +_m f_j(b) +_m f_\varnothing = m$, alors on peut affecter le coût maximal m à $f_{ij}(a, b)$. Le VCSP obtenu définit bien la même fonction de coût globale car la valuation $\top$ est idempotente ($\top \oplus \top = \top$).

La propagation d'un coût non idempotent α doit être compensée pour que la fonction $Val_\mathcal{P}$ reste invariante. Par exemple, si l'opérateur d'agrégation est l'addition d'entiers, alors on peut augmenter le minorant $f_\varnothing$ de $\alpha = \min\{f_i(a) : a \in D_i\}$ à condition de diminuer $f_i(a)$ de α pour toute valeur $a \in D_i$. Cette opération s'appelle la *projection unaire*. Une instance de VCSP est *nœud cohérente* dès qu'il n'est plus possible d'appliquer une projection unaire ni de propager un coût idempotent de $f_\varnothing$ vers une contrainte valuée unaire.

On appelle *projection* tout déplacement de coût d'une contrainte valuée de portée S ($|S| > 1$) vers une contrainte valuée unaire $f_i(a)$ (où $i \in S$ et $a \in D_i$). L'augmentation de $f_i(a)$ par α est compensée par la diminution de chaque $f_S(t)$ (tel que $t[x_i] = a$) par α. Il faut que les nouveaux coûts appartiennent à la structure de valuation. Par exemple, dans le cas de coûts réels, une condition nécessaire pour que l'on puisse utiliser $f_\varnothing$ comme un minorant de $Val_\mathcal{P}$ est d'avoir des coûts non négatifs. Il s'en suit que le coût projeté α ne doit pas dépasser $\min\{f_S(t) : t[x_i] = a\}$. Une instance de VCSP est *arc cohérent* si elle est nœud cohérent et s'il n'est pas possible d'appliquer une projection ni de propager un coût idempotent vers une contrainte f_S. De manière générale, la fermeture par cohérence d'arc n'est pas unique. Ainsi, plusieurs notions différentes de la cohérence d'arc co-existent dans le cadre VCSP selon le temps que l'on est prêt à consacrer à chercher la meilleure fermeture.

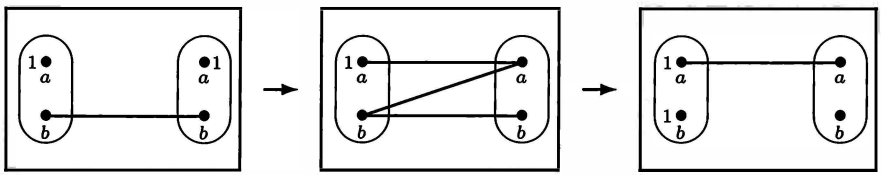


FIGURE 2 – Opérations d'extension et de projection appliquées à un VCSP.

On appelle *extension* tout déplacement de coût α de $f_i(a)$ vers f_S. Dans le cas de coûts réels, une extension équivaut à la projection d'un coût négatif. Un ensemble bien choisi d'opérations de projection et d'extension peut permettre de regrouper des coûts sur une seule variable, ce qui peut déclencher l'augmentation du minorant $f_\varnothing$ via une projection unaire. La figure 2(a) représente un VCSP de deux variables avec domaines $\{a, b\}$ sous forme de graphe. Il y a deux fonctions de coût unaires ($f_1(a) = f_2(a) = 1$), et l'arête qui relie les deux valeurs b représente un coût de 1 : $f_{1,2}(b, b) = 1$. La Figure 2(b) montre le résultat d'une extension de $f_2(a)$ vers la fonction de coût binaire, tandis que la Figure 2(c) montre le résultat d'une projection depuis cette fonction de coût binaire vers $f_1(b)$. Enfin, une projection unaire depuis la fonction de coût unaire f_1 permettrait d'en déduire un minorant $f_\varnothing = 1$. Pour les coûts appartenant à $\mathbb{R}^{\geq 0} \cup \{\infty\}$, il est possible de trouver un ensemble d'opérations menant à la meilleure augmentation possible de $f_\varnothing$ via la résolution d'un programme linéaire [Schlesinger, 1976 ; Cooper *et al.*, 2007, 2010]. Malheureusement, la taille du programme linéaire à résoudre limite pour l'instant cette technique à l'étape de prétraitement.

On peut s'approcher d'une fermeture optimale sans avoir recours à un programme linéaire. Étant donné une instance VCSP $\mathcal{P}$, on peut la transformer en une instance CSP en remplaçant chaque fonction de coût f_S (sauf $f_\varnothing$) par la relation

$$R_P \;=\; \{t \in \ell(S) : f_S(t) = 0\}$$

On appelle Bool($\mathcal{P}$) l'instance CSP qui en résulte car la structure de valuation est devenue booléenne. Les solutions de Bool($\mathcal{P}$) sont les affectations t telles que $Val_\mathcal{P}(t) = f_\varnothing$. L'instance VCSP $\mathcal{P}$ est *arc cohérent virtuel* (VAC) si la fermeture par cohérence d'arc (dans le sens CSP) de Bool($\mathcal{P}$) est non vide. Soit S une séquence d'opérations de propagation de contraintes (dans le sens CSP) qui élimine toutes les valeurs dans un domaine quelconque D_i de Bool($\mathcal{P}$). Il est toujours possible de traduire S en une séquence S' de projections, extensions et projections unaires qui augmente $f_\varnothing$ dans $\mathcal{P}$ [Cooper *et al.*, 2008, 2010]. Pour établir la cohérence d'arc virtuelle, il suffit de chercher une séquence S, d'appliquer la séquence correspondante S' et de recommencer jusqu'à ce que la fermeture par cohérence d'arc de Bool($\mathcal{P}$) soit non vide. Même si, en théorie, la convergence n'est pas garantie, en pratique, cet algorithme est suffisamment rapide pour pouvoir être appliqué pendant la recherche. Assez complexe, il devrait pouvoir faire l'objet de nombreuses améliorations.

Une technique heuristique mais moins gourmande en temps de calcul tente de regrouper des coûts en les envoyant toujours vers les variables qui apparaissent plus tôt

dans l'ordre d'instanciation. On garantit la convergence en ne permettant que les opérations d'extension qui respectent cet ordre. La technique EDAC combine cette approche directionnelle avec une version de VAC sur le voisinage de chaque variable [de Givry *et al.*, 2005].

Jusqu'ici, l'essentiel des travaux ont donc porté autour de la cohérence dite d'arc, effectuant uniquement des mouvements de coût impliquant une unique contrainte d'arité 2 ou plus [Sànchez *et al.*, 2008]. Il reste un travail théorique, algorithmique et expérimental important pour explorer le monde des cohérences de plus haut niveau, impliquant plusieurs contraintes valuées.

7.5.1 Fonctions de coût globales

Une partie du succès de la programmation par contraintes provient de la définition de contraintes dites « globales ». Ce sont des contraintes ayant une sémantique précise et qui peut généralement être exploitée afin de définir des algorithmes de propagation de contraintes plus efficaces que les algorithmes génériques. C'est le cas de certaines fonctions de coût, dont le minimum peut être trouvé plus efficacement que dans le cas général, pour lesquelles des algorithmes efficaces de propagation aux bornes ont été proposés [Zytnicki *et al.*, 2009].

Un autre exemple très connu est la contrainte « `AllDiff` » qui impose à un ensemble de variables de prendre des valeurs toutes différentes (voir le chapitre II.6) et qui possède un algorithme de filtrage efficace basé sur la théorie des couplages.

Si différentes contraintes globales avec des variables de coût ont pu voir le jour dans le passé, ce n'est que très récemment qu'il a été montré qu'il était possible de faire de la propagation de coût sur des fonctions de coût globales comme le `AllDiff` [Lee et Leung, 2009, 2012]. Ce premier résultat ouvre sans doute sur une série de résultats visant à transférer la longue liste de contraintes globales des réseaux de contraintes aux VCSP. Ceci est en particulier aisé lorsque les contraintes globales sont décomposables en réseaux de fonctions d'arité bornée [Allouche *et al.*, 2012a].

7.6 Complexité et classes traitables

La complexité du VCSP dépend de sa structure de valuation. Cependant, toute structure de valuation contient une valuation idempotente $\bot$ (qui représente la satisfaction totale et donc un coût nul) et une valuation absorbante $\top \neq \bot$ (qui représente l'insatisfaction totale et donc l'interdiction). Puisqu'il est toujours possible de coder toutes les instances CSP via les valuations $\bot$ et $\top$, le VCSP est NP-difficile pour toute structure de valuation.

Néanmoins, il existe des algorithmes de complexité polynomiale pour certains sous-problèmes de VCSP et pour certaines structures de valuation. Si l'opérateur d'agrégation est idempotent (ce qui est le cas dans les CSP flous et possibilistes), l'étude de la complexité du VCSP se ramène à l'étude de la complexité du CSP, car résoudre le VCSP revient à résoudre ses problèmes de coupe [Meseguer *et al.*, 2006, page 293], que l'on obtient en remplaçant chaque fonction de coût f_P par la relation

$$R_P^\alpha \;=\; \{t \in l(P) : f_P(t) \leq \alpha\}.$$

Lorsque l'opérateur d'agrégation n'est pas idempotent, l'étude de la complexité du VCSP ne peut s'inspirer directement de l'étude de la complexité du CSP. Par exemple, 2SAT $\in P$ mais MAX-2SAT est NP-difficile. Il existe une seule classe non triviale de fonctions de coût qui définit un sous-problème traitable des VCSP issu de MAX-SAT : la classe de fonctions sous-modulaires. Soit $f_P : D^r \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ une fonction de coût où le domaine D possède un ordre total. La fonction f_P est *sous-modulaire* [Fujishige, 2005] si $\forall x = (x_1, \ldots, x_r)$, $y = (y_1, \ldots, y_r) \in D^r$,

$$f_P(\min(x,y)) \ + \ f_P(\max(x,y)) \ \leq \ f_P(x) \ + \ f_P(y)$$

où $\min(x,y) = (\min(x_1,y_1), \ldots, \min(x_r,y_r))$ et $\max(x,y)$ est défini de façon similaire. La classe de fonctions sous-modulaires comprend toute fonction d'un seul argument, des fonctions binaires $\sqrt{x^2+y^2}$, $((x \geq y)?(x-y)^t : m)$ (pour $t \geq 1$), $K - xy$, la fonction de coupe d'un graphe, la fonction de rang d'un matroïde [Gondran et Minoux, 2009, chapitre 9] ainsi que la fonction η_a^ρ (pour $\rho > 0$), où

$$\eta_a^\rho(x) \ = \ \begin{cases} 0 & \text{si } (x_1 < a_1 \wedge \ldots \wedge x_r < a_r) \vee (x_{r+1} > a_{r+1} \wedge \ldots \wedge x_s > a_s) \\ \rho & \text{sinon.} \end{cases}$$

Un algorithme issu de la Recherche Opérationnelle permet de résoudre un VCSP avec contraintes valuées sous-modulaires en temps $O(n^5 e)$ où n est le nombre de variables et e le nombre de contraintes [Orlin, 2009]. Une autre approche [Cooper *et al.*, 2008] consiste à d'abord établir la cohérence d'arc (ce qui préserve la sous-modularité des contraintes valuées) et à ensuite établir la cohérence d'arc dans le CSP Bool($\mathcal{P}$) où $\mathcal{P}$ est maintenant VAC. Par la définition de VAC, la fermeture $\mathcal{Q}$ par cohérence d'arc de Bool($\mathcal{P}$) est non vide et, par la définition de la sous-modularité, ses contraintes sont *max-closed* [Jeavons et Cooper, 1995] ; pour trouver une solution de $\mathcal{Q}$ (qui est nécessairement une solution optimale de $\mathcal{P}$), il suffit de choisir la valeur maximale dans chaque domaine.

Dans le cas du VCSP avec coûts appartenant à $\mathbb{R}^{\geq 0}$ et qui admet toute contrainte unaire, la seule restriction sur les types de contraintes valuées qui garantisse l'existence d'un algorithme polynomial est la sous-modularité ; sans cette restriction, le problème est APX-complet (ce qui implique la non-existence d'un schéma d'approximation en temps polynomial si $P \neq NP$) [Deineko *et al.*, 2008]. L'étude des propriétés algébriques des fonctions de coût a permis de caractériser toutes les classes traitables sur domaines booléens dans le cas de coûts appartenant à $\mathbb{R}^{\geq 0} \cup \{\infty\}$ [Cohen *et al.*, 2006]. Ces résultats indiquent l'importance primordiale de la notion de sous-modularité dans l'identification de classes traitables de contraintes valuées.

7.7 Outils de résolution et applications

Que ce soit dans le domaine des réseaux de fonctions de coût, ou dans le domaine proche des problèmes de satisfaisabilité pondérés partiels (MaxSAT, correspondant à des réseaux de fonctions de coût avec des domaines booléens formulés via l'utilisation de clauses), différents outils de résolution sont actuellement développés dans le monde de la recherche.

Au niveau des outils propositionnels, la compétition internationale MaxSAT regroupe les outils les plus connus. On pourra se reporter sur ce sujet sur le site web des compétitions MaxSAT (`http://maxsat.ia.udl.cat/`). Ces outils sont en général capables de traiter des problèmes modélisés en logique propositionnelle pondérée, sous forme normale conjonctive, comme un ensemble de clauses pondérées ou non (les clauses non pondérées étant considérées comme dures). Mais un certain nombre d'entre eux peuvent être limités aux clauses non pondérées, ou ne pas permettre l'expression de clauses dures.

Pour ce qui est du traitement des VCSP, la majorité des outils se concentrent maintenant sur les VCSP pondérés (avec des coûts additifs, bornés ou non) car ils ont une cible applicative assez large. Les compétitions MaxCSP sont encore une fois un endroit idéal pour accéder à ces outils (`http://www.cril.univ-artois.fr/CPAI08/`).

Une large collection d'instances de problèmes réels, académiques et aléatoires, modélisés sous la forme de VCSP additifs, est maintenue et accessible à l'adresse `http://costfunction.org`. Ce site permet aussi d'accéder au logiciel open-source `toulbar2` (`mulcyber.toulouse.inra.fr/projects/toulbar2/`), qui offre un large éventail de méthodes, de l'élimination de variables aux algorithmes de recherche arborescente s'appuyant sur la propagation de fonctions de coût. `toulbar2` est également capable de traiter des problèmes exprimés sous forme de réseau probabiliste (réseau bayésien ou champ markovien) (`http://www.cs.huji.ac.il/project/UAI10/`).

7.7.1 Applications

La cible applicative de ces méthodes est large et recouvre tout problème pouvant se ramener à l'optimisation d'une somme de fonctions de coût locales sur des variables discrètes. Ils ont ainsi été appliqués, entre autres, à des problèmes de gestion de ressources (affectation de fréquences [Cabon *et al.*, 1999]), à des problèmes d'ordonnancement avec sélection de tâche (sélection de prises de vues dans un satellite d'observation [Verfaillie *et al.*, 1996]), à des problèmes d'analyse de pedigree en génétique [Sànchez *et al.*, 2008], de planification avec coûts [Cooper *et al.*, 2006], à des problèmes de localisation de gènes d'ARN [Zytnicki *et al.*, 2008] ou de design de protéines [Allouche *et al.*, 2012b].

Au delà de l'optimisation combinatoire pure, le traitement du problème de recherche d'explication de probabilité maximum (MPE, Maximum Probability Explanation) dans les réseaux bayésiens ou de maximum *a posteriori* (MAP) dans les champs markoviens se modélise et se traite directement comme des problèmes VCSP additifs. Ils permettent donc aussi de traiter des problèmes sur des formalismes probabilistes voisins ou inclus (chaînes de Markov cachées, champs aléatoires conditionnels sur des horizons finis...).

En dehors des critères probabilistes, certains critères possibilistes peuvent aussi se réduire à un problème VCSP additif. C'est en particulier le cas du critère dit « lexicographique » [Schiex *et al.*, 1995].

7.8 Conclusion

Cette rapide présentation n'a évidemment pas vocation à être exhaustive. Nous n'avons pas, par exemple, abordé les techniques de compilation de connaissances (*know-*

ledge compilation), bien connues dans le monde CSP et en logique propositionnelle (voir le chapitre II.5) et qui ont également été étendues aux VCSP [Darwiche et Marquis, 2004; Fargier et Marquis, 2007] ou aux CSP à semi-anneau, dans une version étendue [Wilson, 2005].

Néanmoins, ce chapitre montre que le formalisme des CSP valués s'est, en une dizaine d'années, doté d'un ensemble de résultats fondamentaux (classes polynomiales), de techniques, d'algorithmes (recherche arborescente, cohérences locales, élimination de variables...) et d'outils logiciels qui en font un modèle solide pour formuler et résoudre des problèmes d'optimisation combinatoire sur les modèles graphiques (déterministes ou stochastiques). Les récentes extensions de la notion de contrainte globale (ou de fonction de coût globale) basées sur les cohérences locales du domaine devraient permettre d'ici peu d'offrir au modélisateur un outil aussi riche que la programmation par contraintes classique, mais avec un pouvoir d'expression et une efficacité accrue pour les problèmes d'optimisation. D'autres domaines, tels que l'exploitation de symétries dans les VCSP par exemple, sont encore peu étudiés.

La notion de modèle graphique valué a, de fait, été utilisée au-delà de la communauté de la programmation par contraintes. Les notions de modèles graphiques stochastiques discrets tels que les réseaux bayésiens ou plus encore les champs de Markov [Chellappa et Jain, 1993] ou les graphes de facteurs [Kschischang *et al.*, 2001] constituent autant de formalismes qui sont essentiellement équivalents aux réseaux de fonctions de coût mais dont l'interprétation probabiliste pose naturellement des problèmes de dénombrement de solutions — liés à des calculs de « marginales » — plutôt que des problèmes d'optimisation. Les notions de cohérence locale des VCSP, préservant les distributions jointes, pourraient trouver ici un domaine d'application nouveau, tout comme les fonctions de coût globales pourront augmenter le pouvoir d'expression de ces formalismes, souvent limités, dans le cas discret, à l'expression de fonctions de coût (probabilités conditionnelles, potentiels, facteurs) impliquant peu de variables et définies par des tables.

Les réseaux de fonctions de coût offrent ainsi un outil idéal pour exprimer et traiter simultanément préférences/utilités, incertitudes/probabilités I.3 et plus classiquement faisabilités et contraintes. Dans ce cadre, on distingue habituellement les variables contrôlables (par le décideur) des variables non contrôlables (représentant l'environnement et la source d'éventuelles incertitudes). On pourra se reporter à [Dubois *et al.*, 1996] pour le cas possibiliste, le traitement algébrique de tels problèmes ayant été étudié en particulier dans [Pralet *et al.*, 2007].

Références

Aarts, E. et Lenstra, J. K. (1997). *Local Search in Combinatorial Optimization.* Interscience Series in Discrete Mathematics and Optimization. John Wiley and sons.

Aji, S. et McEliece, R. (2000). The generalized distributive law. *Information Theory, IEEE Transactions on*, 46(2) :325–343.

Allouche, D., Bessiere, C., Boizumault, P., de Givry, S., Métivier, J., Gutierrez, P., Loudni, S. et Schiex, T. (2012a). Filtering decomposable global cost

functions. *In Proceedings of the 26th National Conference on Artificial Intelligence*, pages 407–413, Toronto, Canada. AAAI Press.

ALLOUCHE, D., TRAORÉ, S., ANDRÉ, I., de GIVRY, S., KATSIRELOS, G., BARBE, S. et SCHIEX, T. (2012b). Computational protein design as a cost function network optimization problem. *In Principles and Practice of Constraint Programming*, Québec, Canada. Springer Verlag.

BACCHUS, F. et GROVE, A. (1995). Graphical models for preference and utility. *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 3–10. Morgan Kaufmann.

BERTELÉ, U. et BRIOSHI, F. (1972). *Nonserial Dynamic Programming.* Academic Press.

BISTARELLI, S., MONTANARI, U. et ROSSI, F. (1995). Constraint solving over semirings. *In Proc. of IJCAI-95*, Montréal, Canada.

BISTARELLI, S., MONTANARI, U. et ROSSI, F. (1997). Semiring-based constraint satisfaction and optimization. *Journal of the ACM (JACM)*, 44(2) :201–236.

BODLAENDER, H. L. et KOSTER, A. M. C. A. (2008). Treewidth Computations I. Upper Bounds. Rapport technique UU-CS-2008-032, Utrecht University, Department of Information and Computing Sciences, Utrecht, The Netherlands.

BOROS, E. et HAMMER, P. (2002). Pseudo-Boolean Optimization. *Discrete Appl. Math.*, 123 :155–225.

CABON, B., DE GIVRY, S., LOBJOIS, L., SCHIEX, T. et WARNERS, J. (1999). Radio link frequency assignment. *Constraints*, 4 :79–89.

CHELLAPPA, R. et JAIN, A. (1993). *Markov Random Fields : Theory and Applications.* Academic Press.

COHEN, D., COOPER, M., JEAVONS, P. et KROKHIN, A. (2006). The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11) :983 – 1016.

COOPER, M., CUSSAT-BLANC, S., ROQUEMAUREL, M. D. et RÉGNIER, P. (2006). Soft arc consistency applied to optimal planning. *In Proc. of CP'06*, numéro 4204 de LNCS, pages 680–684, Nantes, France. Springer.

COOPER, M., DE GIVRY, S., SANCHEZ, M., SCHIEX, T. et ZYTNICKI, M. (2008). Virtual arc consistency for valued CSP. *In Proc. AAAI'08*, pages 253–258.

COOPER, M., de GIVRY, S., SÁNCHEZ, M., SCHIEX, T., ZYTNICKI, M. et WERNER, T. (2010). Soft arc consistency revisited. *Artificial Intelligence*, 174(7) :449–478.

COOPER, M. C. (2003). Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342.

COOPER, M. C. (2005). High-order consistency in Valued Constraint Satisfaction. *Constraints*, 10 :283–305.

COOPER, M. C., de GIVRY, S. et SCHIEX, T. (2007). Optimal soft arc consistency. *In Proc. of the 20th IJCAI*, pages 68–73, Hyderabad, India.

COOPER, M. C. et SCHIEX, T. (2004). Arc consistency for soft constraints. *Artificial Intelligence*, 154 :199–227.

DARWICHE, A. et MARQUIS, P. (2004). Compiling propositional weighted bases. *Arti-*

ficial Intelligence, 157(1) :81–113.

de GIVRY, S. et JEANNIN, L. (2006). A unified framework for partial and hybrid search methods in constraint programming. *Computer & Operations Research*, 33(10) :2805–2833.

de GIVRY, S., SCHIEX, T. et VERFAILLIE, G. (2006). Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. *In Proc. of the National Conference on Artificial Intelligence, AAAI-2006*, pages 22–27.

de GIVRY, S., ZYTNICKI, M., HERAS, F. et LARROSA, J. (2005). Existential arc consistency : getting closer to full arc consistency in weighted CSPs. *In Proc. of the 19th IJCAI*, pages 84–89, Edinburgh, Scotland.

DECHTER, R. (1999). Bucket elimination : A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2) :41–85.

DECHTER, R. (2003). *Constraint Processing.* Morgan Kaufmann Publishers.

DECHTER, R. et RISH, I. (2003). Mini-buckets : A general scheme for approximating inference. *Journal of ACM*, 50(2) :1–61.

DEINEKO, V., JONSSON, P., KLASSON, M. et KROKHIN, A. (2008). The approximability of MAX CSP with fixed-value constraints. *Journal of the ACM*, 55(4).

DOMSHLAK, C., ROSSI, F., VENABLE, K. et WALSH, T. (2003). Reasoning about soft constraints and conditional preferences : complexity results and approximation techniques. *In Proc. of IJCAI-03*, pages 215–220, Acapulco, Mexico.

DUBOIS, D., FARGIER, H. et PRADE, H. (1996). Possibility theory in constraint satisfaction problems : Handling priority, preference and uncertainty. *Applied Intelligence*, 6(4) :287–309.

DUBOIS, D. et PRADE, H. (1991). Inference in possibilistic hypergraphs. *Uncertainty in Knowledge Bases*, pages 249–259.

FARGIER, H. et LANG, J. (1993). Uncertainty in constraint satisfaction problems : a probabilistic approach. *In Proc. of ECSQARU '93*, volume 747 de *LNCS*, pages 97–104, Grenade, Spain.

FARGIER, H., LANG, J. et SCHIEX, T. (1993). Selecting preferred solutions in Fuzzy Constraint Satisfaction Problems. *In Proc. of the 1st European Congress on Fuzzy and Intelligent Technologies.*

FARGIER, H. et MARQUIS, P. (2007). On valued negation normal form formulas. *In Proc. of the 20th IJCAI*, pages 360–365, Hyderabad, India.

FREUDER, E. et WALLACE, R. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70.

FUJISHIGE, S. (2005). *Submodular Functions and Optimisation.* Numéro 58 de Annals of Discrete Mathematics. Elsevier, Amsterdam, 2nd édition.

GONDRAN, M. et MINOUX, M. (2009). *Graphes et algorithmes.* Lavoisier, EDF R&D.

JEAVONS, P. G. et COOPER, M. C. (1995). Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2) :327–339.

KLEMENT, E., MESIAR, R. et PAP, E. (2000). *Triangular Norms.* Kluwer cademic Publishers.

KSCHISCHANG, F., FREY, B. et LOELIGER, H. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2) :498–519.

LECOUTRE, C., SAIS, L., TABARY, S. et VIDAL, V. (2006). Last conflict based reasoning. *In ECAI 2006 : 17th European Conference on Artificial Intelligence*, page 133, Riva del Garda, Italy.

LEE, J. et LEUNG, K. (2009). Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. *In Proc. of the 21th IJCAI*, Pasadena (CA), USA.

LEE, J. et LEUNG, K. (2012). Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *Journal of Artificial Intelligence Research*, 43 :257–292.

LOUDNI, S. et BOIZUMAULT, P. (2006). Combining VNS with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191(3) :705–735.

MARINESCU, R. et DECHTER, R. (2005). AND/OR branch-and-bound for graphical models. *In Proc. of the 19th IJCAI*, page 224, Edinburgh, Scotland.

MESEGUER, P., ROSSI, F. et SCHIEX, T. (2006). *Soft Constraints*, chapitre 9, pages 281–328. Elsevier.

ORLIN, J. B. (2009). A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming Ser. A*, 118(2) :237 – 251.

PRALET, C., VERFAILLIE, G. et SCHIEX, T. (2007). An algebraic graphical model for decision with uncertainties, feasibilities, and utilities. *Journal of Artificial Intelligence Research*, 29 :421–489.

ROSENFELD, A., HUMMEL, R. et ZUCKER, S. (1976). Scene labeling by relaxation operations. *IEEE Trans. on Systems, Man, and Cybernetics*, 6(6) :173–184.

SÀNCHEZ, M., de GIVRY, S. et SCHIEX, T. (2008). Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. *Constraints*, 13(1-2) : 130–154.

SCHIEX, T. (1992). Possibilistic constraint satisfaction problems or "How to handle soft constraints ?". *In Proc. of the 8th Int. Conf. on Uncertainty in Artificial Intelligence*, pages 269–275, Stanford, CA.

SCHIEX, T. (2000). Arc consistency for soft constraints. *In Proc. of CP-00*, volume 1894 de *LNCS*, pages 411–424, Singapore.

SCHIEX, T., FARGIER, H. et VERFAILLIE, G. (1995). Valued constraint satisfaction problems : hard and easy problems. *In Proc. of IJCAI-95*, pages 631–637, Montréal, Canada.

SCHLESINGER, M. (1976). Sintaksicheskiy analiz dvumernykh zritelnikh signalov v usloviyakh pomekh (Syntactic analysis of two-dimensional visual signals in noisy conditions). *Kibernetika*, 4 :113–130. In Russian.

SHAFER, G. et SHENOY, P. (1988). Local computations in hyper-trees. Working paper 201, School of business, University of Kansas.

SHAPIRO, L. et HARALICK, R. (1981). Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3 :504–519.

TERRIOUX, C. et JEGOU, P. (2003). Bounded backtracking for the valued constraint satisfaction problems. *In Proc. of the Ninth International Conference on Principles and Practice of Constraint Programming (CP-2003).*

VERFAILLIE, G., LEMAÎTRE, M. et SCHIEX, T. (1996). Russian doll search. *In Proc. of AAAI'96*, pages 181–187, Portland, OR.

WILSON, N. (2005). Decision diagrams for the computation of semiring valuations. *In Proc. of the 19th IJCAI*, page 331, Edinburgh, Scotland.

ZYTNICKI, M., GASPIN, C., de GIVRY, S. et SCHIEX, T. (2009). Bounds arc consistency for weighted CSPs. *Journal of Artificial Intelligence Research.*

ZYTNICKI, M., GASPIN, C. et SCHIEX, T. (2008). DARN ! A soft constraint solver for RNA motif localization. *Constraints*, 13(1-2) :91–109.

Chapitre 8

Modèles graphiques pour l'incertitude : inférence et apprentissage

Les modèles graphiques pour l'incertitude regroupent un ensemble de formalismes graphiques servant à représenter et raisonner avec des connaissances et informations incertaines, complexes, imprécises ou manquantes. De manière générale, un modèle graphique englobe une composante graphique qui est, selon le type du modèle, un graphe orienté ou non orienté, et une composante quantitative. La composante graphique met en évidence les variables du domaine et les relations entre ces variables (causalité, dépendance conditionnelle, etc.). La composante quantitative sert, quant à elle, à quantifier dans la théorie d'incertitude utilisée les relations dans le modèle. Concernant les utilisations des modèles graphiques, on trouve d'abord la représentation de connaissances incertaines, imprécises et complexes ; le raisonnement et l'aide à la décision ainsi que l'extraction de connaissances à partir de données. Les modèles graphiques disposent d'algorithmes d'inférence efficaces pour répondre à des questions d'intérêt afin de réaliser certaines tâches comme l'explication, le diagnostic, la classification, etc.
Ce chapitre donne un panorama des modèles graphiques les plus répandus. En particulier, il donne un aperçu complet sur les différents aspects relatifs aux modèles graphiques orientés pour l'incertitude : représentation, inférence, apprentissage et, enfin, les applications.

8.1 Introduction

Les modèles graphiques sont des outils importants pour la raisonnement et la décision dans l'incertain. Il existe plusieurs modèles graphiques dont les plus connus sont

Auteurs : Salem Benferhat, Philippe Leray et Karim Tabia.

les réseaux bayésiens [Jensen, 1996 ; Lauritzen et Spiegelhalter, 1988 ; Pearl, 1988 ; Darwiche, 2009], les diagrammes d'influence [Howard et Matheson, 1984 ; Shachter, 1986] (extensions des réseaux bayésiens aux problèmes de décision), les arbres de décision [Raiffa, 1968], les CP-nets (« Ceteris Paribus networks » dédiés pour la représentation des préférences) [Hoos et Poole, 2004], les GAI (« Generalized additive independence ») [Braziunas et Boutilier, 2005 ; Gonzales *et al.*, 2008], les réseaux possibilistes [Fonck, 1994 ; Ben Amor et Benferhat, 2005 ; Benferhat et Smaoui, 2007], les réseaux à base de fonctions de croyances [Shenoy, 1989, 1993a ; Ben Yaghlane et Mellouli, 2008 ; Xu et Smets, 1994], etc.

La majorité des modèles graphiques font clairement référence à la théorie des probabilités. Les réseaux bayésiens sont des modèles graphiques très utilisés. Ils sont représentés par des graphes orientés acycliques (DAG), où chaque nœud correspond à une variable et chaque arc représente une relation d'influence entre deux variables. L'incertitude est exprimée par des probabilités conditionnelles pour chaque nœud dans le contexte de ses parents. Les modèles graphiques ont été également développés dans d'autres théories de l'incertain comme la théorie des possibilités ou les fonctions de croyances. La théorie des possibilités est une théorie de l'incertain qui fournit un cadre adapté pour un traitement quantitatif et ordinal des informations incertaines.

Les applications de plus en plus nombreuses des réseaux bayésiens utilisent toutes le mécanisme d'inférence, c'est-à-dire la propagation des informations dans le réseau, afin d'évaluer les influences entre les différents éléments du système. L'inférence dans les réseaux bayésiens dépend directement de la structure graphique. Pour les graphes simplement connectés [Pearl, 1988] (un seul chemin entre deux nœuds), l'inférence est réalisée en un temps polynomial. Dans le cas de graphes à connexions multiples [Lauritzen et Spiegelhalter, 1988] (plusieurs chemins peuvent exister entre deux nœuds), l'algorithme de propagation est généralement coûteux. Les algorithmes les plus utilisés se basent sur des méthodes de compilation qui consistent à traduire un probléme décrit dans un langage, dont le problème d'inférence est difficile, vers un autre langage, dont le problème d'inférence est facile. Deux types de transformations ou compilations ont été proposés. Le premier type consiste en une transformation du graphe initial, à une autre structure graphique tel que l'arbre de jonction. Un nœud au niveau de cet arbre de jonction est dit une clique. Cette clique est formée d'un ensemble de variables. Le deuxième type de compilation consiste en un codage de l'ensemble des paramètres (degrés de probabilités ou de possibilités conditionnelles) vers une formule propositionnelle sous forme normale conjonctive. Cette formule conjonctive est ensuite traduite vers un langage cible [Cadoli et Donini, 1998], comme le langage DNNF (Decomposable Negation Normal Form) [Darwiche, 2001] qui supporte un certain nombre d'opérations, utilisées dans les réseaux bayésiens, en un temps polynomial.

La première partie de ce chapitre présentera les définitions de base des réseaux bayésiens ainsi que deux exemples d'algorithmes de propagation de l'incertitude. Dans la deuxième partie, nous présenterons le problème d'apprentissage de réseaux bayésiens à partir de données. Ce chapitre contient également une section qui contient quelques applications des réseaux bayésiens ainsi que les plates-formes les plus utilisées.

8.2 Modèles graphiques

8.2.1 Définitions de base et relations d'indépendance

Cette section donne un bref rappel sur les notions de base relatives aux réseaux bayésiens. Soient $V = \{A_1, A_2, ..., A_n\}$ un ensemble fini de variables et E un sous-ensemble du produit cartésien $V \times V$.

Dans ce qui suit, un *graphe* correspond à une paire $G = (V, E)$ où V constitue l'ensemble des *sommets*, aussi connus sous le nom *nœuds*, d'un graphe et E est l'ensemble de *liens* entre les nœuds dans V. Un *arc* est une arête orientée. Un arc de A_i vers A_j est noté $A_i \rightarrow A_j$. A_i est appelée *origine* et A_j l'*extrémité*.

Un graphe constitué d'arcs est appelé *graphe orienté*. Pour les graphes orientés, il existe la notion de parents et d'enfants. S'il existe un arc $A_i \rightarrow A_j$, A_i est appelé *parent* de A_j et A_j l'*enfant* de A_i. L'ensemble des parents de A_i est noté par U_{A_i} ou U_i. L'ensemble des enfants de A_i est noté par Y_{A_i} ou Y_i. Une *racine* est un nœud qui n'a pas de parents. Une *feuille* est un nœud qui n'a pas d'enfants.

Les graphes orientés ne possédant pas de cycles sont appelés *DAG*s (Directed Acyclic Graphs). Un DAG simplement connecté [Kim et Pearl, 1983] ou un polyarbre [Rebane et Pearl, 1987] est un DAG où entre deux nœuds il existe au plus un chemin. Un DAG à connexions multiples est un DAG où il existe au moins une paire de nœuds qui sont reliés par plus qu'un chemin.

Pearl et ses collègues [Geiger *et al.*, 1989, 1990 ; Pearl, 1988] ont établi que les informations incertaines peuvent être gérées plus efficacement en tirant profit de la notion d'indépendance conditionnelle. La *d-séparation* est un critère qu'ils ont proposé, dans le cadre des réseaux bayésiens, permettant de calculer toutes les indépendances conditionnelles sur un DAG [Pearl, 1988]. La d-séparation implique l'indépendance. En effet, deux variables sont *d-séparées* par un sous-ensemble de variables Z dans un graphe orienté implique qu'ils sont indépendants dans le contexte de Z dans toutes les distributions de probabilité associées à ce graphe. Un modèle graphique permet donc de représenter les relations d'indépendance conditionnelle qui existent entre les variables (ou ensemble de variables). Soient X, Y, W et Z des sous ensembles des variables dans V. Ces relations d'indépendance conditionnelle sont gouvernées (mais pas totalement) par les axiomes suivants, appelés axiomes des graphoïdes [Pearl et Paz, 1986 ; Studeny, 1990] :

1 (Symétrie) $X \perp Y|Z \Rightarrow Y \perp X|Z$
2 (Décomposition) $X \perp YW|Z \Rightarrow X \perp Y|Z$
3 (Faible union) $X \perp YW|Z \Rightarrow X \perp Y|ZW$
4 (Contraction) $X \perp Y|Z \& X \perp W|ZY \Rightarrow X \perp YW|Z$
5 (Intersection) $X \perp W|ZY \& X \perp Y|ZW \Rightarrow X \perp YW|Z$

où $X \perp Y|Z$ signifie que X est indépendant de Y dans le contexte de Z.

Lorsqu'un modèle satisfait les quatre premiers axiomes $[1-4]$ alors il est appelé semi-graphoïde. S'il satisfait aussi l'axiome 5 alors il est appelé graphoïde. La relation d'indépendance probabiliste est un semi-graphoïde.

8.2.2 Les réseaux probabilistes et extensions

Les réseaux bayésiens sont des cas particuliers de modèles graphiques. Ces modèles se basent sur la théorie des graphes et la théorie des probabilités.

Un réseau bayésien est défini par :

— Un composant graphique : Les variables et les arcs forment un graphe orienté acyclique (DAG).

— Un composant numérique : Chaque lien dans le DAG est quantifié par une distribution de probabilité conditionnelle $p(A_i|U_{A_i})$ de tout nœud A_i dans le contexte de ses parents U_{A_i}.

Un réseau bayésien est un formalisme qui permet de représenter de façon compacte une distribution jointe de probabilités sur un ensemble de variables en utilisant la notion d'indépendance. En effet, une distribution jointe sur n variables binaires requiert 2^n entrées. Les indépendances entre les variables permettent de décomposer cette large distribution en un ensemble de distributions locales associées à un petit nombre de variables. Cette approche permet une économie en mémoire et surtout le traitement de problèmes de tailles importantes. Cette représentation permet en outre de répondre de façon cohérente et efficace à diverses questions sur cette distribution de probabilité notamment l'inférence à partir d'observations.

Les distributions de probabilité locales (a priori) doivent satisfaire les contraintes de ***normalisation*** suivantes :

— Si $U_{A_i} = 0$ (A_i est une racine) alors la distribution marginale associée à A_i doit satisfaire :

$$\forall a_i \in D_{A_i}, \sum_{a_i} P(a_i) = 1. \tag{8.1}$$

— Si $U_{A_i} \neq 0$ alors la distribution conditionnelle associée à A_i doit satisfaire :

$$\forall u_{A_i} \in \times D_{A_j \in U_{A_i}}, \sum_{a_i} P(a_i|u_{A_i}) = 1. \tag{8.2}$$

Un réseau bayésien tel qu'il a été défini possède la propriété de Markov, c'est-à-dire toute variable est indépendante des ses non-descendants dans le contexte de ses parents. Cette propriété permet d'écrire la probabilité jointe sous une forme factorisée en utilisant la règle de Bayes :

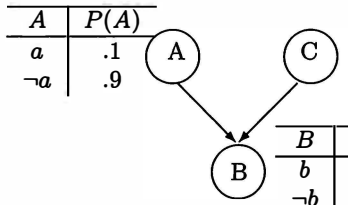
$$P(A_1, ..., A_n) = \prod_{i=1}^{n} P(A_i|U_{A_i}). \tag{8.3}$$

Cette forme compacte de la distribution jointe est aussi appelée ***règle de chaînage.***

L'exemple suivant permet d'illustrer la notion de règle de chaînage dans les réseaux bayésiens.

Exemple 22. Soit le réseau bayésien représenté par le graphe donné par la figure 1.

La distribution de probabilité jointe est donnée par :
$P(A, B, C) = P(A).P(B|AC).P(C)$.



A	$P(A)$
a	.1
$\neg a$	.9

C	$P(C)$
c	.6
$\neg c$	.4

B	A	C	$P(B\|AC)$
b	a	c	.2
$\neg b$	a	c	.8
b	a	$\neg c$	.7
$\neg b$	a	$\neg c$	.3
b	$\neg a$	c	.5
$\neg b$	$\neg a$	c	.5
b	$\neg a$	$\neg c$	.4
$\neg b$	$\neg a$	$\neg c$	.6

FIGURE 1 – Exemple de réseau bayésien

Ainsi le degré de probabilité associé à l'interprétation $a \wedge b \wedge \neg c$ est égal à :
$P(a, b, \neg c) = P(a).P(b|a\neg c).P(\neg c) = 0.1 * 0.7 * 0.4 = 0.028.$

La principale utilisation des modèles graphiques est le raisonnement et l'inférence. Cela concerne la déduction de certaines informations d'intérêt étant données les informations codées par le réseau et d'autres informations disponibles (des observations par exemple). Les modèles graphiques sont dotés de mécanismes d'inférence pour répondre à des questions d'intérêt afin de réaliser certaines tâches comme l'explication, le diagnostic (voir le chapitre I.18), la classification, etc.
De manière générale, l'inférence consiste à propager une ou plusieurs informations certaines (observées) et d'évaluer la modification dans les croyances représentées par le modèle graphique. Il s'agit par exemple de calculer la probabilité a posteriori des variables de question Q (variables non observées) étant donnée une évidence E (un ensemble de variables observées). Le résultat est la probabilité $P(Q|E)$. Il existe trois principales catégories de requêtes (donc d'inférence) :

— **Probabilité d'un événement :** Dans ces requêtes, on veut mesurer la probabilité d'un événement donné e. Il s'agit de calculer $P(e)$ depuis le réseau. Les mécanismes utilisés pour ce type de requêtes sont généralement la règle de chaînage et la marginalisation.

— **Explication la plus probable (MPE) :** L'objectif ici est d'identifier la plus *probable* instanciation v^* (tuple de valeurs) de toutes les variables V du réseau qui soit cohérente avec une instanciation e d'un sous-ensemble de variables E. On cherche ici la meilleure explication pour E=e, calculée comme suit :

$$v^* = argmax_v(P(v, e)) \tag{8.4}$$

— **Maximum a posteriori :** Dans les requêtes de type maximum a posteriori (MAP), il s'agit de calculer l'hypothèse h^*, instance de H (variables d'hypothèse ou variables non observées), la plus probable pour une observation donnée e (instance des variables observées E). Les sous-ensembles E et H sont disjoints. Par exemple, dans les problèmes de classification, il s'agit de trouver la classe (variable non observable) la plus probable étant donné les caractéristiques de

l'objet à classifier (l'évidence ici concerne les valeurs des variables caractérisant les objets étudiés). L'inférence MAP est basée sur la formule suivante :

$$h^* = argmax_h(P(h|e)). \tag{8.5}$$

De manière générale, les mécanismes utilisés pour l'inférence dans les réseaux bayésiens sont le conditionnement, la règle de Bayes, la marginalisation et la règle de chaînage. Les réseaux bayésiens permettent, grâce à ces mécanismes, de calculer n'importe quelle probabilité d'intérêt qu'on peut utiliser en classification, explication, etc. Notons que les algorithmes d'inférence, de par la nature du résultat qu'ils fournissent, se répartissent en algorithmes d'inférence exacte (qui fournissent des résultats exacts) et des algorithmes d'inférence approchée (car leurs résultats, pour des considérations de complexité de calcul, sont des approximations des probabilités réelles). Les algorithmes d'inférence dans les réseaux bayésiens peuvent être répartis selon leurs approches en trois catégories :

1. *Inférence par élimination de variables* : Le principe de ces algorithmes est d'éliminer de proche en proche et par marginalisation certaines variables jusqu'à répondre à la requête. L'ordre d'élimination dépend de la requête et de la structure du réseau. La complexité en temps de ce type d'algorithmes est exponentielle dans la largeur (en terme de nombre de variables) des facteurs (tables relatives à de sous ensembles de variables) rencontrés lors de l'élimination.
2. *Inférence par compilation* : On peut mettre dans cette catégorie tous les algorithmes qui passent par des structures ou langages intermédiaires pour répondre (efficacement) aux requêtes posées. Parmi ces algorithmes, certains sont basés sur des structures d'arbres. Le plus connu de ces algorithmes est l'arbre de jonction [Jensen, 1996] oú le réseau bayésien de départ est compilé en arbre de clusters et séparateurs (dans la section suivante, la version possibiliste de l'algorithme d'arbre de jonction est présentée. Un exemple de langage cible DNNF est également présenté dans le cadre possibiliste). Après cette compilation, certains calculs se font de manière locale aux clusters et séparateurs. La complexité de l'inférence dépend ici de la largeur de l'arbre (consistant en la largeur du plus grand cluster).
3. *Inférence par conditionnement* : L'idée principale dans cette approche est d'exploiter l'évidence sur laquelle porte la requête pour simplifier le réseau initial en arbre ou poly-arbre. A titre d'exemple, les observations sont intégrées en déconnectant les variables observées de leurs descendants directs et en mettant à jour les tables locales de ces derniers. Des algorithmes d'inférence dits par instanciation exploitent également cette idée en instanciant certaines variables et réalisant des inférences sur les réseaux résultants [Pearl, 1988].

Le lecteur intéressé pourra trouver des détails récents et une synthèse des principaux résultats de complexité de l'inférence dans les réseaux bayésiens dans [de Campos, 2011].

Diagrammes d'influence

Les diagrammes d'influence [Howard et Matheson, 1984 ; Shachter, 1986] sont une extension des réseaux bayésiens pour les problèmes de décision sous incertitude (voir

le chapitre I.14). Ainsi, un diagramme d'influence contient trois types de nœuds :

— *Noeuds de chance* : Ils représentent des variables aléatoires comme dans les réseaux bayésiens.

— *Noeuds de décision* : Un nœud de décision représente les différentes alternatives ou actions pouvant être choisies. Chaque décision procure une certaine utilité.

— *Noeuds d'utilité* : Ces nœuds quantifient l'utilité (ou la satisfaction) associée à chaque décision possible.

Exemple 23. Comme le montre l'exemple (académique) de la figure 2, les nœuds de chance **Temps** et **Prévisions météo** sont des variables aléatoires qui décrivent le problème traité. Ces variables sont associées à des tables de probabilités locales. La variable **Prendre parapluie** est une variable de décision alors que la variable **Satisfaction** est le nœud d'utilité. Généralement, les arcs sont interprétés comme des relations d'influence causale. Dans cet exemple, le temps qu'il fait un jour donné influence les prévisions météo du lendemain. De même, les prévisions météo influencent sur la décision de prendre ou laisser son parapluie. La satisfaction procurée par la décision prise dépend à la fois du fait qu'on a pris son parapluie ou pas et du temps qu'il fait.

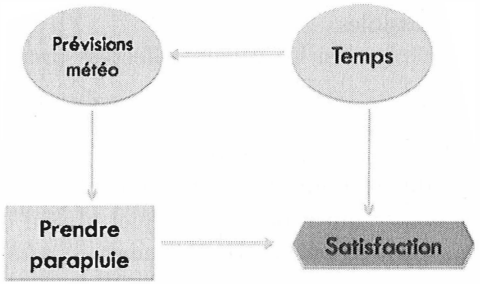


FIGURE 2 – Exemple de diagramme d'influence

Les diagrammes d'influence sont utilisés pour modéliser de manière compacte des problèmes de décision dans l'incertain. Leur principale utilisation est le calcul de la meilleure décision dans un contexte donné. Le critère de décision utilisé est le maximum d'utilité espérée. Des diagrammes d'influence possibilistes ont été proposés dans [Garcia et Sabbadin, 2008].

Systèmes à base de valuation VBS

Les modèles graphiques ont été également étudiés dans le cadre des fonctions de croyances [Shafer, 1976 ; Smets, 1998]. Les premiers travaux se trouvent dans un système connu sous le nom de VBS (*Valuation Based Systems*) [Shenoy, 1993b] conçu pour représenter et raisonner avec des informations incertaines dans les systèmes experts. Les fonctions de croyances utilisées dans ce formalisme sont basées sur la théorie de Dempster-Shafer. Les VBS n'utilisent pas des structures de DAG et se basent sur les propriétés algébriques des opérations du conditionnement et de la fusion pour la propagation de l'incertitude associée aux nœuds du graphe. Une version possibiliste

des VBS a été proposée dans [Shenoy, 1992]. Dans [Simon *et al.*, 2008], une adaptation (assez directe) des réseaux bayésiens a été proposée. Dans [Ben Yaghlane et Mellouli, 2008 ; Xu et Smets, 1994] une structure de DAG est également utilisée, cependant les fonctions de masses conditionnelles sont définies au niveau de chaque parent au lieu de tous les parents comme dans les réseaux bayésiens.

Réseaux possibilistes

Les modèles graphiques possibilistes ont été étudiés dans plusieurs travaux. La composante graphique a exactement la même syntaxe et sémantique que dans les réseaux bayésiens. La composante numérique, quant à elle, est basée sur la théorie des possibilités au lieu des probabilités.
Un élément de base de la théorie des possibilités est la notion de *distribution de possibilité* π qui est une application de Ω vers l'intervalle [0,1]. Une distribution de possibilités code des connaissances incertaines. $\pi(\omega) = 1$ signifie que ω est complètement possible et $\pi(\omega) = 0$ signifie qu'il est impossible que ω soit le monde réel. L'échelle possibiliste peut être interprétée de deux manières différentes :

- d'une manière *ordinale*, si les valeurs affectées ne reflètent que l'ordre entre les différents mondes possibles.
- d'une manière *numérique*, si les valeurs affectées prennent un sens sur l'intervalle.

Etant donnée une distribution de possibilité π définie sur l'univers de discours Ω, on peut définir deux mesure duales pour chaque évènement $\varphi \subseteq \Omega$:

- la *mesure de possibilité* qui représente à quel niveau φ est cohérent avec nos connaissances représentées par π : $\Pi(\varphi) = \max_{\omega \in \varphi} \pi(\omega)$;
- la *mesure de nécessité* qui correspond au degré de certitude associé à φ à partir des informations codées par π : $N(\varphi) = 1 - \Pi(\bar{\varphi})$.

La section suivante présentera une adaptation de l'algorithme d'arbre de jonction dans le cadre de la théorie des possibilités.

8.2.3 Algorithmes de propagation dans les modèles graphiques

Il est possible de déterminer l'effet de la réalisation des valeurs spécifiques de quelques variables sur le reste des variables en utilisant le processus de marginalisation dans le cas où la distribution de probabilité jointe est connue.

L'algorithme de propagation dans les arbres de jonction [Jensen, 1996] est basé sur 4 étapes (S1-S4) décrites ci-dessous. Ces étapes s'appliquent aussi bien aux réseaux bayésiens probabilistes qu'aux réseaux possibilistes. Nous les présentons ici dans le cadre de la théorie des possibilités.

Etape S1 : Construction de l'arbre de jonction

Les principales étapes pour la construction d'un arbre de jonction à partir d'un DAG peuvent être résumées comme suit [Huangn et Darwiche, 1994] :

— *Moralisation du graphe initial* : permet de créer un graphe non orienté à partir du graphe initial en rajoutant des liens entre les parents de chaque variable.
— *Triangulation du graphe moral* : Cette étape a pour but d'identifier des ensembles de variables qui peuvent être regroupées sous forme de *cliques* notées C_i. Notons qu'il est important de trouver une triangulation optimale qui va minimiser la taille des cliques afin de rendre les calculs locaux possibles. Ce problème est connu comme étant *difficile* [Cooper, 1990] et plusieurs heuristiques ont été proposés pour le résoudre [Kjaerulff, 1990].
— *Construction de l'arbre de jonction* $\mathcal{JT}$: Cette étape consiste à connecter les cliques identifiées lors de l'étape précédente, avec la condition que toutes les cliques se trouvant dans un chemin entre cliques C_i et C_j doivent contenir $C_i \cap C_j$. Une fois les cliques adjacentes identifiées, on insère un séparateur, noté S_{ij}, entre chaque paire de cliques C_i et C_j, contenant leurs variables communes.

Exemple 24. Soit le réseau bayésien de la figure 3.

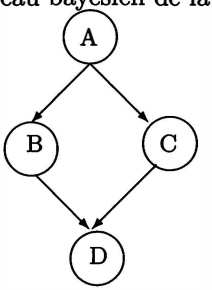


FIGURE 3 – Example de DAG d'un réseau bayésien à quatre variables.

La figure 4 donne un exemple d'un arbre de jonction associé au DAG de la figure 3 contenant les deux cliques suivantes : $C_1 = \{ABC\}$ et $C_2 = \{BCD\}$ ainsi que leur séparateur $S_{12} = \{BC\}$.

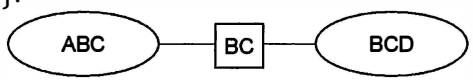


FIGURE 4 – Arbre de jonction associé au graphe $\Pi\mathcal{G}$ de la figure 3.

Etape S2 : Initialisation

Une fois l'arbre de jonction construit, cette étape permet de le quantifier en transformant les distributions de possibilité conditionnelles initiales en des distributions jointes locales rattachées aux cliques et aux séparateurs.

Plus spécifiquement, pour chaque clique C_i (resp. séparateur S_{ij}) de $\mathcal{JT}$, on affecte un potentiel $\pi^t_{C_i}$ (resp. $\pi^t_{S_{ij}}$) où t est relatif à l'étape de propagation. En particulier, $t = I$ (resp. $t = C$) correspond à l'étape d'initialisation (resp. de cohérence globale). Ces potentiels permettent d'associer à l'arbre de jonction $\mathcal{JT}$ une distribution de possibilité jointe, unique, notée $\pi^t_{\mathcal{JT}}$ définie par :

$$\pi^t_{\mathcal{JT}}(A_1, .., A_N) = \min_{i=1..m} \pi^t_{C_i}, \tag{8.6}$$

où m est le nombre de cliques dans $\mathcal{JT}$.

Les grandes lignes de la procédure d'initialisation peuvent être résumées comme suit :

— Pour chaque clique, C_i affecter une distribution uniforme : $\pi^I_{C_i} \leftarrow 1$ [1].

— Pour chaque séparateur, S_{ij} affecter une distribution uniforme : $\pi^I_{S_{ij}} \leftarrow 1$.

— Pour chaque variable $A_k \in V$, choisir une clique C_i contenant $\{A_k\} \cup U_{A_k}$ et mettre à jour ses distributions de possibilité locales : $\pi^I_{C_i} \leftarrow min(\pi^I_{C_i}, \Pi(A_k \mid U_{A_k}))$.

Il est à noter que si une clique contient un nombre important de nœuds alors le calcul de sa distribution jointe peut devenir impossible à cause de sa taille.

Etape S3 : Propagation globale

Avant de présenter l'étape de propagation globale, nous introduisons la notion de cohérence globale dans les arbres de jonction.
Soient C_i et C_j deux cliques adjacentes dans l'arbre de jonction $\mathcal{JT}$ et soit S_{ij} leur séparateur. Le lien entre C_i et C_j (ainsi que le séparateur S_{ij}) est dit *stable* ou *cohérent* si :

$$\max_{C_i \setminus S_{ij}} \pi^t_{C_i} = \pi^t_{S_{ij}} = \max_{C_j \setminus S_{ij}} \pi^t_{C_j}, \tag{8.7}$$

où $\max_{C_i \setminus S_{ij}} \pi^t_{C_i}$ est la distribution marginale de S_{ij} définie à partir de $\pi^t_{C_i}$.

Si tous les liens dans un arbre de jonction sont cohérents alors l'arbre de jonction est dit *globalement cohérent.*

Une fois l'arbre de jonction initialisé, le processus de propagation globale permettra d'assurer sa cohérence globale à travers un passage de messages entre les cliques. La première étape consiste à choisir arbitrairement une clique *pivot* qui va démarrer ce processus caractérisé par deux phases :

— phase de *collecte de l'évidence*, dans laquelle chaque clique envoie un message à sa clique adjacente dans la direction du pivot (dans cette direction chaque clique a une et une seule clique adjacente),

— phase *distribution de l'évidence*, dans laquelle chaque clique envoie un message à ses cliques adjacentes dans le sens contraire du pivot en démarrant par le pivot lui-même jusqu'à atteindre les feuilles du graphe.

Lorsque l'arbre de jonction est globalement cohérent, alors le potentiel de chaque clique correspond à sa distribution locale calculée à partir du réseau initial.

Etape S4 : Réponse aux requêtes

L'arbre de jonction résultant de la phase précédente est globalement cohérent ce qui signifie que le potentiel de chaque clique code $\Pi_{\mathcal{PG}}(C_i)$. Donc, le calcul de la marginale relative à toute variable $A_k \in V$, peut se faire en marginalisant le potentiel de n'importe quelle clique C_i comme suit :
$\Pi_{\mathcal{PG}}(A_k) = \max_{C_i \setminus A_k} \pi^C_{C_i}$.

1. 1 signifie la distribution de possibilité où tous les éléments sont égaux à 1

Les algorithmes d'inférence dans les arbres de jonction possibilistes, bien qu'efficaces, atteignent leurs limites lorsque les cliques générées sont de tailles importantes.

Il existe différents algorithmes alternatifs à celui basé sur les arbres de jonctions. Parmi ces alternatives, on trouve les approches basées sur la compilation de bases de connaissances décrites en langage propositionnel. De manière informelle, la compilation consiste à transformer un ensemble de clauses propositionnelles en un autre ensemble de formules écrites dans un langage dit « cible » dont l'inférence se réalise en un temps polynomial. Un exemple simple de langage cible est celui basé sur les formes normales disjonctives. La compilation des modèles graphiques se fait en deux étapes : i) coder le modèle graphique par une base de clauses propositionnelles, et ii) compiler la base de connaissances obtenue vers un langage cible qui supporterait les requêtes standard des modèles graphiques comme par exemple le calcul des probabilités a postériori.

La compilation des modèles graphiques a été largement étudiée dans les réseaux bayésiens probabilistes par Darwiche et ses collègues [Darwiche, 2002 ; Chavira et Darwiche, 2005, 2007 ; Pipatsrisawat et Darwiche, 2008] et récemment étudiée dans le cadre de la théorie des possibilités dans [Ayachi *et al.*, 2011].

Il existe plusieurs langages cibles beaucoup plus compacts ou succints que la forme normale disjonctive. Ces langages compacts sont des cadres particuliers des formes normales négatives NNF où le symbole de négation apparaît uniquement au niveau des symboles propositionnels. Les langages dits DNNF *Decomposable Negation Normal Form (DNNF)* ou d-DNNF *Deterministic Decomposable Negation Normal Form (DNNF)* [Darwiche, 2001] [Darwiche et Marquis, 2002] sont particulièrement utilisés pour la compilation des modèles graphiques.

Le codage d'un modèle graphique est simple. L'idée est d'associer à chaque instante x_i d'une variable X, un nouveau symbole propositionnel $\lambda_{x_{ij}}$ et d'associer à chaque paramètre, un degré de probabilité $P(x_i \mid u_{i1}, \ldots u_{im})$ ou un degré de possibilité $\Pi(x_i \mid u_{i1}, \ldots u_{im})$, un symbole propositionnel noté $\theta_{x_i|u_i}$. Ensuite, il faut décrire la base de connaissances qui consiste d'une part à dire que les variables associées aux instances sont mutuellement exclusives et que les degrés d'incertitude associés à $x_i \mid u_{i1}, \ldots u_{im}$ sont égaux aux degrés associés aux variables. La mise en forme clausale donne deux ensembles de clauses :

— **Clauses d'exclusions mutuelles** : $\forall X_i \in V$,

$$\lambda_{x_{i1}} \vee \lambda_{x_{i2}} \vee \cdots \lambda_{x_{in}}$$
$$\neg\lambda_{x_{ij}} \vee \neg\lambda_{x_{ik}}, j \neq k$$

— **Clauses associées aux paramètres** :$\forall X_i \in V, \forall\ \theta_{x_i|u_i}$,

$$\lambda_{x_i} \wedge \lambda_{u_{i1}} \wedge \ldots \wedge \lambda_{u_{im}} \rightarrow \theta_{x_i|u_i}$$
$$\theta_{x_i|u_i} \rightarrow \lambda_{x_i}$$
$$\theta_{x_i|u_i} \rightarrow \lambda_{u_{i1}}, \cdots, \theta_{x_i|u_i} \rightarrow \lambda_{u_{im}}$$

Plusieurs variantes de ce codage ont été proposées, en particulier dans le cadre de la

théorie des possibilités [Ayachi *et al.*, 2010]. Par ailleurs, dans [Ayachi *et al.*, 2011] une extension du codage est proposée pour la prise en compte du concept d'interventions essentiel dans la définition de la causalité dans les modèles graphiques.

8.3 Apprentissage et classification avec les réseaux bayésiens

Un réseau bayésien est défini par un graphe représentant un ensemble de dépendances et d'indépendances conditionnelles, et par un ensemble de probabilités conditionnelles. Apprendre un réseau bayésien revient donc à trouver le graphe (i.e. la structure) et les paramètres des distributions conditionnelles associées. Un réseau bayésien pouvant servir à la fois de modèle génératif ou de modèle discriminant par rapport à une variable cible, l'apprentissage d'un tel modèle pourra lui aussi varier, que ce soit du côté de la structure - en utilisant des familles de structures spécifiques pour la classification par exemple - ou de la fonction optimisée au cours de l'apprentissage.

Nous allons passer tout d'abord en revue les principales méthodes **d'apprentissage des paramètres**, où nous supposerons que le graphe est déjà connu, puis nous aborderons le problème plus complexe de **l'apprentissage de la structure**. Nous aborderons ensuite le cas particulier de la classification et de l'apprentissage discriminant.

Par la suite, nous supposerons que les variables du réseau bayésien sont discrètes et que les données utilisées pour l'apprentissage sont complètes. Pour plus d'informations sur l'apprentissage des réseaux bayésiens avec données incomplètes, nous conseillons la lecture de [Francois, 2006].

8.3.1 Apprentissage des paramètres

Si le graphe est déjà connu, apprendre un réseau bayésien revient à déterminer les différentes tables de probabilités conditionnelles associées à chaque variable X_i $\{P(X_i = x_k | pa(X_i) = x_j)\}_{jk}$.

Apprentissage statistique

La méthode d'estimation statistique la plus simple est celle du *maximum de vraisemblance (MV)*) :

$$\hat{\theta}^{MV}_{i,j,k} = \frac{N_{i,j,k}}{\sum_k N_{i,j,k}} \tag{8.8}$$

où $N_{i,j,k}$ est le nombre d'occurences de l'événement $\{ X_i = x_k$ et $Pa(X_i) = x_j\}$ dans la base de données.

Apprentissage bayésien

Lorsque la base de données est de taille réduite, ou lorsqu'une expertise est disponible concernant les valeurs des paramètres, les méthodes d'estimation bayésienne

comme le *maximum a posteriori (MAP)* ou l'*espérance a posteriori (MAP)* s'avèrent être plus intéressantes.

Ces méthodes nécessitent la définition d'une distribution a priori sur les paramètres à estimer. Dans le cas discret classique, la distribution a priori conjuguée est la distribution de Dirichlet dont les cœfficients α peuvent être interprétés comme des nombres d'occurence a priori de chaque événement.

L'approche de *maximum a posteriori (MAP)* nous donne ainsi :

$$\hat{\theta}_{i,j,k}^{MAP} = \frac{N_{i,j,k} + \alpha_{i,j,k} - 1}{\sum_k (N_{i,j,k} + \alpha_{i,j,k} - 1)} \tag{8.9}$$

où $\alpha_{i,j,k}$ sont les paramètres de la distribution de Dirichlet associée à la loi a priori $P(X_i = x_k | pa(X_i) = x_j)$.

L'*espérance a posteriori (EAP)* nous donne une formule assez similaire :

$$\hat{\theta}_{i,j,k}^{EAP} = \frac{N_{i,j,k} + \alpha_{i,j,k}}{\sum_k (N_{i,j,k} + \alpha_{i,j,k})} \tag{8.10}$$

8.3.2 Apprentissage de la structure

L'apprentissage de la structure des réseaux bayésiens est un problème NP-difficile [Chickering *et al.*, 1994] qui a donné lieu à de très nombreux travaux et états de l'art [Heckerman, 1998 ; Daly *et al.*, 2011].

Le nombre de structures possibles est super-exponentiel en fonction du nombre n de variables [Robinson, 1977]. Même si certains travaux s'intéressent à la recherche de la solution exacte lorsque le nombre de variables est faible [Koivisto et Sood, 2004 ; Koivisto, 2006 ; Parviainen et Koivisto, 2009 ; Malone *et al.*, 2011], les méthodes existantes proposent des heuristiques permettant de trouver un bon modèle avec un nombre de variables de plus en plus grand.

La première grande famille d'approches est dénommée *à base de contraintes.* Ces méthodes utilisent le fait qu'un réseau bayésien est un modèle graphique d'indépendances. Elles utilisent des tests d'indépendance conditionnelle pour retrouver les informations nécessaires à la reconstruction graphique du modèle.

La seconde famille *à base de score* va essayer de trouver une structure maximisant une fonction de score, approximation de la vraisemblance marginale, en parcourant l'espace des solutions de manière heuristique.

La dernière famille *hybride* tire partie des avantages des deux familles précédentes en combinant utilisation de mesures de dépendances statistiques et optimisation à base de score.

Dans tous les cas, se pose le problème de l'identifiabilité. En effet, un même modèle d'indépendance peut correspondre à plusieurs structures de réseaux bayésiens. Cette notion dénommée *équivalence de Markov* ou *équivalence de vraisemblance* fait que les algorithmes d'apprentissage classiques ne peuvent identifier une structure qu'à sa classe d'équivalence près.

8.3.3 Apprentissage à base de contraintes

Cette famille d'algorithmes est issue des travaux des initiateurs des réseaux bayésiens avec Pearl et Verma d'un côté pour l'algorithme IC [Pearl et Verma, 1991 ; Pearl, 2000] et Spirtes, Glymour et Scheines de l'autre pour l'algorithme PC [Spirtes *et al.*, 1993, 2000]. Ces algorithmes sont basés sur un principe identique :

- construire un graphe non dirigé contenant les relations de dépendance entre variables, à partir de tests d'indépendance conditionnelle,
- identifier les V-structures, sous-structures dirigées ayant des propriétés de dépendances conditionnelles que n'ont pas les autres sous-structures,
- compléter l'orientation d'autres arêtes en utilisant le fait que (1) toutes les V-structures ont déjà été détectées et (2) que le graphe orienté ne doit pas contenir de circuit. Cette étape applique un ensemble de règles décrites dans [Meek, 1995].

Le nombre de tests statistiques à effectuer explosant de manière combinatoire, plusieurs heuristiques ont été proposées. La plus connue, utilisée dans l'algorithme PC [Spirtes *et al.*, 1993], consiste à effectuer tout d'abord les tests d'indépendance, puis les tests conditionnellement à une seule variable, puis deux variables, etc ... en réduisant à chaque étape le nombre de tests à effectuer.

Il faut noter que la fiabilité des tests statistiques utilisés diminue de manière exponentielle par rapport au nombre de variables considérées, ce qui fait que ces méthodes sont souvent limitées à des problèmes de moins d'une centaine de variables.

8.3.4 Algorithmes à base de score

Cette deuxième famille de méthodes a pour but de parcourir de manière heuristique l'espace super-exponentiel des solutions, en maximisant une fonction de score déterminée.

Fonctions de score

Les fonctions de score utilisées sont des approximations de la vraisemblance marginale $P(\mathcal{D}|\mathcal{B})$ [Chickering et Heckerman, 1996].

Une première approximation du calcul de cette vraisemblance mène aux scores AIC et BIC, où l'on retrouve les principes très généraux de sélection de modèle proposés par [Akaike, 1970] et [Schwartz, 1978] :

$$ScoreAIC(\mathcal{B}, \mathcal{D}) = \log L(\mathcal{D}|\theta^{MV}, \mathcal{B}) - Dim(\mathcal{B}) \tag{8.11}$$

$$ScoreBIC(\mathcal{B}, \mathcal{D}) = \log L(\mathcal{D}|\theta^{MV}, \mathcal{B}) - \frac{1}{2} Dim(\mathcal{B}) \log N \tag{8.12}$$

où N est la taille de la base de données et $Dim(\mathcal{B})$ la dimension du réseau bayésien, nombre de paramètres indépendants permettant de décrire l'ensemble des distributions de probabilité conditionnelle associé au graphe.

$$Dim(\mathcal{B}) = \sum_{i=1}^{n} (r_i - 1) q_i \tag{8.13}$$

avec r_i cardinalité de X_i et $q_i = \prod_{X_j \in pa(X_i)} r_j$ nombre de configurations des parents de X_i.

Des hypothèses sur la distribution a priori des paramètres nous permettent d'arriver à une seconde approximation de la vraisemblance marginale, le score BDe (*Bayesian Dirichlet Equivalent*) [Heckerman *et al.*, 1994] :

$$ScoreBDe(\mathcal{B}, \mathcal{D}) = p(\mathcal{B}) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \tag{8.14}$$

avec $\alpha_{ijk} = N' \times P(X_i = x_k, pa(X_i) = x_j | \mathcal{B}_c)$ où $\mathcal{B}_c$ est la structure a priori n'encodant aucune indépendance conditionnelle (graphe complètement connecté) et N' est un nombre d'exemples « équivalent » défini par l'utilisateur.

Si cette distribution de probabilité estimée dans la structure $\mathcal{B}_c$ est uniforme, nous retrouvons un cas d'a priori uniforme non informatif $\alpha_{ijk} = \frac{N'}{r_i q_i}$ proposé initialement par [Buntine, 1991] et souvent appelé score BDeu dans la litérature.

Ces différents scores vérifient deux propriétés importantes : *score equivalence* et *décomposabilité*. La première propriété correspond au fait que deux structures équivalentes au sens de Markov doivent obtenir le même score [Chickering, 1995 ; Heckerman *et al.*, 1994]. La seconde propriété indique que'une fonction de score (globale) *Score* peut s'écrire comme la somme (ou le produit, à une transformation logarithmique près) de scores locaux s ne faisant intervenir qu'une variable X_i et ses parents dans le graphe : $Score(\mathcal{B}, \mathcal{D}) = \sum_{i=1}^{n} s(X_i, pa_i)$.

Recherche dans l'espace des DAG

A l'aide des fonctions de score décrites précédemment, l'apprentissage de la structure d'un réseau bayésien peut se reformuler comme un problème d'optimisation. Une recherche exhaustive dans l'espace des graphes orientés sans circuit (DAG) étant impossible, de nombreuses heuristiques ou méta-heuristiques ont alors été proposées. Citons par exemple la recherche de l'arbre de recouvrement maximal (*MWST Maximum Weight Spanning Tree*) [Chow et Liu, 1968 ; Heckerman *et al.*, 1994], l'algorithme K2 [Cooper et Herskovits, 1992] ou ses variantes [Bouckaert, 1993] qui utilisent la connaissance a priori d'un ordonnancement des nœuds, ou la recherche gloutonne dans l'espace des DAG [Chickering *et al.*, 1995] ou dans l'espace des représentants des classes d'équivalences [Auvray et Wehenkel, 2002 ; Chickering, 2002].

D'autres méta-heuristiques sont elles aussi possibles : recuit simulé, algorithmes génétiques [Larrañaga *et al.*, 1996 ; Delaplace *et al.*, 2007 ; Auliac *et al.*, 2007 ; Muruzabal et Cotta, 2007], optimisation par essaims particulaires [Wang et Yang, 2010] ou par colonies de fourmis.

De part la taille des voisinages considérés (quadratique pour une recherche gloutonne), ces méthodes sont souvent limitées à des problèmes d'un millier de variables.

8.3.5 Apprentissage hybride

Les méthodes hybrides combinent les avantages des méthodes précédentes. Parmi ces méthodes, les plus récentes permettent de traiter des problèmes avec plusieurs

milliers voire centaines de milliers de variables. Le principe de ces méthodes réside en deux étapes. La première consiste à déterminer le voisinage local de chaque variable. Ce voisinage peut être soit l'ensemble des parents et enfants (sans distinction) de la variable, soit sa couverture de Markov (parents, enfants et parents des enfants). Plusieurs travaux se sont spécifiquement penchés sur cette identification locale, avec par exemple MMPC [Tsamardinos *et al.*, 2006] pour les parents-enfants ou IAMB [Tsamardinos *et al.*, 2003], PCMB [Peña *et al.*, 2007] ou MBOR [Rodrigues De Morais et Aussem, 2008] pour la couverture de Markov.

La seconde étape illustrée par exemple dans l'algorithme MMHC [Tsamardinos *et al.*, 2006] consiste à effectuer une optimisation globale de type recherche gloutonne, en parcourant un espace des DAG contraint par les voisinages locaux découverts précédemment.

8.3.6 Classification

Apprentissage génératif versus discriminant

Lorsque les méthodes d'apprentissage classiques cherchent à trouver un réseau bayésien qui soit un bon modèle génératif $P(X_1...X_n)$, les méthodes d'apprentissage utilisées en classification cherchent à construire un bon modèle prédictif $P(C|X_1...X_n)$ où C est la variable de classe à prédire.

La maximisation de la vraisemblance « discriminante » liée au modèle prédictif est malheureusement plus complexe que pour le cas génératif. La solution n'a pas d'expression analytique simple et doit être obtenue par des méthodes de type descente de gradient [Friedman *et al.*, 1997 ; Grossman et Domingos, 2004 ; Greiner *et al.*, 2002 ; Pernkopf et Bilmes, 2005].

En pratique, le classifieur est donc obtenu en cherchant une structure spécifique tenant compte du rôle précis de la variable C, mais en appliquant les algorithmes d'apprentissage de structure classiques qui optimisent la vraisemblance marginale. Ensuite les paramètres du réseau peuvent être estimés, soit classiquement comme dans la section précédente, soit en optimisant la vraisemblance « discriminante ».

Structures pour la classification

Le premier modèle proposé pour la classification est le réseau bayésien naïf NB. Ce réseau fait l'hypothèse que la variable de classe C et les autres variables X_1 à X_n sont directement dépendantes, mais que les observables X_i sont indépendantes conditionnellement à C. Cette hypothèse forte mène à la structure décrite par la figure 5.

Ce modèle est encore très utilisé de part ses avantages : une structure complètement déterminée et un nombre de paramètres réduits et estimables très simplement par maximum de vraisemblance. Cette simplicité correspond parfaitement au principe de parcimonie, qui fait que même si les hypothèses de départ ne sont pas vérifiées, le réseau bayésien naïf obtiendra très souvent de bonnes performances.

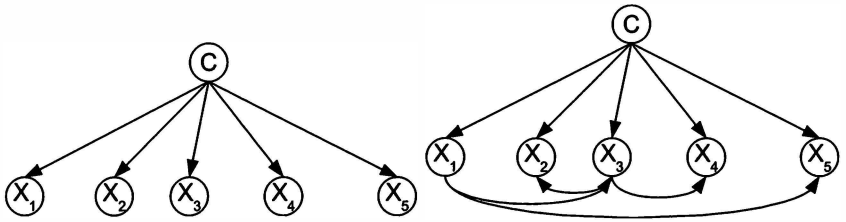


FIGURE 5 – Réseaux bayésiens naïf et naïf augmenté par un arbre

Plusieurs extensions ont été proposées pour lever l'hypothèse d'indépendance conditionnelle des variables X_i. Trouver le meilleur modèle de dépendance entre les X_i (conditionnellement à la classe) est tout aussi complexe que trouver la structure d'un réseau bayésien. Les heuristiques les plus courantes sont l'algorithme TANB (Tree Augmented Naive Bayes) [Friedman *et al.*, 1997] où les X_i sont reliés par un arbre de recouvrement maximal, ou FANB (Forest Augmented Naive Bayes) [Keogh et Pazzani, 1999] où les X_i sont reliés par une forêt (ensemble de sous-arbres non connectés) .

Lorsque les données sont complètes, il est aussi possible d'utiliser la propriété de la couverture de Markov de la variable C. En effet, ce sous-ensemble MB des variables X (X_1 à X_n) est tel que $P(C|MB, X \backslash MB) = P(C|MB)$. De plus, cet ensemble est défini par l'ensemble des parents, des enfants, et des autres parents des enfants de la variable C. Les méthodes d'identification locale décrites dans la section précédente permettent alors de déterminer cet ensemble.

Dans le cadre possibiliste, l'apprentissage des modèles graphiques est peu développé et la majorité des travaux sont des adaptations des techniques d'apprentissage de réseaux bayésiens. La classification avec des réseaux possibilistes a été étudiée dans plusieurs travaux. A titre d'exemple, la classification et l'inférence avec des données incertaines est étudiée dans [Benferhat et Tabia, 2012]. La classification avec option de rejet dans les classifieurs à base de réseaux possibilistes a été proposée dans [Tabia, 2011].

8.4 Applications

Les modèles graphiques pour l'incertitude ont été largement adoptés dans différents domaines d'application. On les retrouve en détection de fraude, diagnostic et aide à la décision médicale [Long, 1989], analyse forensique, recherche d'information, détection de cibles militaires, planification sous incertitude, bio-informatique [Mourad *et al.*, 2011], reconnaissance des formes [Zaarour *et al.*, 2004], détection de spams (comme dans le système SpamAssassin) et d'intrusions informatiques, analyse des risques, etc. Les raisons d'un tel succès sont multiples. En particulier, ils conviennent aussi bien pour la représentation des connaissances relatives aux problèmes à modéliser que pour les tâches de raisonnement et de prise de décision lors de la phase opérationnelle

du système. Le caractère modulaire et intuitif ainsi que la facilité que confèrent ces outils pour la représentation des connaissances incertaines et complexes, les possibilités de modélisation et d'apprentissage automatiques, l'efficacité de l'inférence, etc. constituent des avantages très importants.

Comme exemples d'applications basées sur des modèles graphiques probabilistes, opérationnelles depuis quelques années déjà, on trouve d'abord le projet VISTA [Horvitz et Barry, 1995] de l'agence spatiale américaine NASA visant à sélectionner parmi plusieurs milliers d'informations disponibles en temps réel uniquement celles qui pourraient être pertinentes afin de les afficher sur les consoles des différents opérateurs. Dans le domaine de la navigation automatique de sous-marins, Lockheed Martin UUV [Martin, 1996] est un système intelligent pour le contrôle de véhicule sous-marin autonome, développé par Hugin au profil de Lockheed. Dans le domaine des logiciels informatiques grand public, le projet Lumière [Horvitz *et al.*, 1998] de MicroSoft, initié en 1993, vise à anticiper les besoins et problèmes des utilisateurs de logiciels (l'assistant de la suite de bureautique MicroSoft Office est le produit le plus populaire de ce projet). Dans le domaine médical, le système PathFinder/Intellipath [Heckerman *et al.*, 1992] est un système expert bayésien pour l'assistance à l'identification d'anomalies à partir d'échantillons de tissus lymphatiques. Plusieurs publications et livres présentent des cas d'études d'utilisation de modèles graphiques en pratique. A titre d'exemple, dans [Pourret *et al.*, 2008], le lecteur peut trouver des cas pratiques dans plusieurs domaines tels que le diagnostic et l'aide à la décision médicale, analyse forensique, etc. Nous donnerons ci-après quelques exemples d'application de ces modèles dans le domaine de la sécurité informatique.

En sécurité informatique (où il est question de détecter et prévenir toute action pouvant porter atteinte à la disponibilité ou confidentialité ou disponibilité des informations et services), plusieurs problèmes ont été modélisés et des solutions ont été mises en œuvre en utilisant des modèles graphiques. L'un des premiers travaux ayant utilisé un réseau bayésien en détection d'intrusions [Kumar et Spafford, 1994] proposait de modéliser les dépendances entre plusieurs mesures d'anomalie relatives à différents aspects d'activité d'un système informatique (comme le nombre de processus en cours, nombre de connexions, temps CPU, etc.). eBayes [Valdes et Skinner, 2000], l'un des composants du système de détection d'intrusions comportemental EMERALD [Porras et Neumann, 1997], utilise un réseau bayésien naïf. Dans eBayes, le nœud racine représente la classe des sessions TCP tandis que les attributs (comme le nombre d'adresses IP uniques, nombre de ports uniques, etc.) décrivent ces sessions. Durant la phase de détection, les attributs de la session à analyser sont extraits et utilisés par le classifieur bayésien pour déterminer la classe la plus probable pour cette session parmi les classes *Normal* et *Anormal* correspondant respectivement aux sessions normales et sessions anormales. Parmi les systèmes ayant utilisé un modèle graphique pour associer un score d'anomalie à un événement d'audit, l'exemple le plus connu est SPADE [Staniford *et al.*, 2002] qui est un plugin développé par Silicon Defense. SPADE fait partie de SPICE qui contient un deuxième module pour la corrélation d'alertes. Il s'installe sur le système de détection d'intrusions

Snort [2] et permet de détecter certaines anomalies dues aux scans de ports en analysant les entêtes des paquets de synchronisation TCP-SYN et les paquets UDP entrants.

Quant aux travaux académiques, les modèles graphiques pour l'incertitude ont été utilisés pour la détection d'intrusions réseaux, applicatives et systèmes comme ils ont été utilisés dans les approches comportementales ainsi que dans les approches par scénarios. Parmi les utilisations les plus répandues des modèles graphiques en détection d'intrusions, la classification vient en tête et la majorité de ces classifieurs sont des réseaux bayésiens naïfs ou semi-naïfs. Le problème de détection d'intrusions peut être modélisé comme un problème de classification dans la mesure où il s'agit de classer chaque trace d'audit analysée comme normale ou malveillante. Ainsi, on trouve l'utilisation des réseaux bayésiens en détection d'intrusions dans [Benamor *et al.*, 2003] [Kenaza *et al.*, 2010] [Sebyala *et al.*, 2002] [Abouzakhar *et al.*, 2003].

On trouve les classifieurs de type multi-net dans [Tabia et Leray, 2010], les modèles de Markov cachés dans [Sung-Bae, 2002] [Ye *et al.*, 2004] et les réseaux bayésiens dynamiques dans [An *et al.*, 2006]. Dans les approches comportementales, on trouve dans [Kruegel *et al.*, 2005] un modèle utilisanr un réseau bayésien pour combiner six mesures d'anomalie locales afin de déterminer si l'événement analysé (un appel système) est normal ou anormal. Ce modèle modélise, d'une part, les dépendances entre plusieurs modèles de détection utilisés (longueur de la chaıne, distribution des caractères, tec.) et, d'autre part, la pertinence de chaque mesure d'anomalie locale. En corrélation d'alertes (où l'objectif est soit de détecter des scénarios d'attaques complexes ou d'éliminer les fausses alertes), on peut citer la détection de plans d'attaques (séquences d'actions faisant partie d'attaques complexes) en utilisant des modèles probabilistes causaux [Kenaza *et al.*, 2010]. Dans un autre travail [Tabia et Leray, 2010], un modèle visant à tenir compte de la fiabilité de plusieurs systèmes de détection d'intrusions dans un but de corréraltion d'alertes est basé sur un modèle bayésien. Il est important de souligner enfin, que les modèles graphiques pour l'incertitude peuvent être combinés avec d'autres types de modèles et formalismes. A titre d'exemple, dans le projet ANR PLACID [3], des modèles graphiques probabilistes sont combinés avec des formalismes logiques de représentation et raisonnement avec des connaissances et préférences des utilisateurs pour le problème de corrélation d'alertes.

Pour ce qui est des applications et plates-formes de construction et utilisation de modèles graphiques pour l'incertitude, on trouve plusieurs produits. L'un des acteurs incontournables dans le domaine des plates-formes et applications des modèles probabilistes, Hugins [4] est sans doute en tête. Cet éditeur et consultant développe des plates-formes généralistes et des solutions dans de nombreux domaines tels que la médecine, finance, industrie, etc. L'autre plate-forme ayant imposé son nom durant les deux dernières décennies est Netica de la société Norsys [5]. Netica propose une plate-forme

2. www.snort.org
3. http ://placid.insa-rouen.fr/
4. http ://www.hugin.com/
5. http ://www.norsys.com/

complète pour la modélisation et le raisonnement avec des réseaux bayésiens et des diagrammes d'influence. Elle propose également plusieurs bibliothèques et interfaces de programmation pour utiliser des modèles graphiques depuis d'autres applications. Analytica [6] est une autre plate-forme offrant les mêmes types de solutions. D'autres plates-formes se spécialisent dans certains types d'utilisation comme Agenarisk [7] offrant des solutions pour l'analyse de risques. On trouve également des boîtes à outils pour certains environnements comme BN toolbox [8] pour Matlab, JavaBayes [9] pour Java, etc. On peut encore citer d'autres boîtes à outils pour réseaux bayésiens comme *MensXMachina* [10], *Causal Explorer* [11], PMTK [12], etc. En France, deux plates-formes deviennent de plus en plus leaders dans leurs domaines. La plate-forme BayesiaLab de Bayesia [13] et les solutions de ProBayes [14] avec notamment sa librairie ProBT en C++.

8.5 Conclusion

Les modèles graphiques pour l'incertitude sont des outils compacts et puissants pour la représentation et le raisonnement avec des informations complexes et incertaines. En modélisation, ils offrent l'avantage d'être intuitifs, modulaires et se déclinent en plusieurs variantes qui conviennent pour la modélisation des différents types de dépendances (conditionnelles, causales, séquentielles, etc.). En inférence, ils sont efficaces et permettent de multiples usages comme la classification, le diagnostic, l'explication, la planification (voir le chapitre II.9 pour l'utilisation des réseaux bayésiens dynamiques en planification), etc.
Les modèles graphiques pour l'incertitude peuvent être construits directement par un expert ou construits automatiquement à partir de données. La construction d'un modèle graphique par un expert est facilitée par le fait que le processus d'élicitation procède d'abord par une étape qualitative où l'on s'intéresse uniquement aux variables d'intérêts et leurs relations. Dans un deuxième temps, l'expert quantifiera les relations localement (pour chaque variable dans le contexte de ses parents), ce qui facilite énormément le travail de modélisation et d'élicitation. Un modèle graphique est interprétable par un expert notamment dans un but de validation et il peut servir de support pour la communication entre plusieurs experts. De plus, il existe plusieurs cadres pour l'incertitude pouvant être utilisés pour la composante quantitative et pour l'inférence sur le modèle construit. En présence de données représentative du problème à modéliser, il existe plusieurs techniques d'apprentissage pouvant construire automatiquement un modèle depuis ces données. Depuis les premiers travaux sur les systèmes experts probabilistes, la littérature concernant les modèles graphiques est abondante et plusieurs problématiques sont encore l'objet d'intenses travaux dans certaines communautés d'in-

6. http ://www.lumina.com/
7. http ://www.agenarisk.com/
8. http ://code.google.com/p/bnt/
9. http ://www.cs.cmu.edu/ javabayes/Home/
10. http ://www.mensxmachina.org/software/pgm-toolbox/
11. http ://www.dsl-lab.org/causal_explorer/index.html
12. http ://code.google.com/p/pmtk3/
13. http ://www.bayesia.com/
14. http ://www.probayes.com/

telligence artificielle. Les modèles graphiques pour l'incertitudes apparaissent souvent comme thème principal dans les plus prestigieuses conférences en IA et plusieurs numéros de revues scientifiques leurs sont dédiés. Le meilleur indice de la maturité de ces formalismes et de leur intérêt est, sans doute, leur utilisation dans plusieurs domaines d'application depuis les filtres anti-spam jusqu'aux applications les plus sensibles dans le domaine médical, militaire, etc.

Références

ABOUZAKHAR, N., A. GANI, G. M., ABUITBEL, M. et KING, D. (2003). Bayesian learning networks approach to cybercrime detection. *In 2003 PostGraduate Networking Conference (PGNET 2003)*, Liverpool, United Kingdom.

AKAIKE, H. (1970). Statistical predictor identification. *Ann. Inst. Statist. Math.*, 22 :203–217.

AN, X., JUTLA, D. et CERCONE, N. (2006). Privacy intrusion detection using dynamic bayesian networks. *In ICEC '06 : Proceedings of the 8th international conference on Electronic commerce*, pages 208–215, New York, NY, USA. ACM.

AULIAC, C., D'ALCHÉ-BUC, F. et FROUIN, V. (2007). Learning transcriptional regulatory networks with evolutionary algorithms enhanced with niching. *In* MASULLI, F., MITRA, S. et PASI, G., éditeurs : *Applications of Fuzzy Sets Theory*, volume 4578 de *Lecture Notes in Computer Science*, pages 612–619. Springer Berlin / Heidelberg.

AUVRAY, V. et WEHENKEL, L. (2002). On the construction of the inclusion boundary neighbourhood for markov equivalence classes of bayesian network structures. *In* DARWICHE, A. et FRIEDMAN, N., éditeurs : *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 26–35, S.F., Cal. Morgan Kaufmann Publishers.

AYACHI, R., BEN AMOR, N. et BENFERHAT, S. (2011). Compiling min-based possibilistic causal networks : A mutilated-based approach. *In ECSQARU*, pages 700–712, Belfast-UK.

AYACHI, R., BEN AMOR, N., BENFERHAT, S. et HAENNI, R. (2010). Compiling possibilistic networks : Alternative approaches to possibilistic inference. *In Proceedings of 26th Conference on UAI*, pages 40–47. AUAI Press.

BEN AMOR, N. et BENFERHAT, S. (2005). Graphoid properties of qualitative possibilistic independence. *International Journal of Uncertainty, Fuzziness and Knowledge-Based*, 13 :59–96.

BEN YAGHLANE, B. et MELLOULI, K. (2008). Inference in directed evidential networks based on the transferable belief model. *Int. J. Approx. Reasoning*, 48(399-418).

BENAMOR, N., BENFERHAT, S. et ELOUEDI, Z. (2003). Naive bayesian networks in intrusion detection systems. *In ACM*, Cavtat-Dubrovnik, Croatia.

BENFERHAT, S. et SMAOUI, S. (2007). Hybrid possibilistic networks. *Int. J. Approx. Reasoning*, 44(3) :224–243.

BENFERHAT, S. et TABIA, K. (2012). Inference in possibilistic network classifiers under uncertain observations. *Annals of Mathematics and Artificial Intelligence*, pages

1–41. 10.1007/s10472-012-9290-1.

BOUCKAERT, R. R. (1993). Probabilistic network construction using the minimum description length principle. *Lecture Notes in Computer Science*, 747 :41–48.

BRAZIUNAS, D. et BOUTILIER, C. (2005). Local utility elicitation in gai models. *In Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 42–49, Arlington, Virginia. AUAI Press.

BUNTINE, W. (1991). Theory refinement on bayesian networks. *In* D'AMBROSIO, B., SMETS, P. et BONISSONE, P., éditeurs : *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, pages 52–60, San Mateo, CA, USA. Morgan Kaufmann Publishers.

CADOLI, M. et DONINI, F. M. (1998). A survey on knowledge compilation. *AI Communications–The European Journal for Artificial Intelligence*, 10(3-4) :137–150.

CHAVIRA, M. et DARWICHE, A. (2005). Compiling bayesian networks with local structure. *In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1306–1312.

CHAVIRA, M. et DARWICHE, A. (2007). Compiling Bayesian networks using variable elimination. *In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2443–2449.

CHICKERING, D. (1995). A transformational characterization of equivalent Bayesian network structures. *In* BESNARD, P. et HANKS, S., éditeurs : *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI'95)*, pages 87–98, San Francisco, CA, USA. Morgan Kaufmann Publishers.

CHICKERING, D., GEIGER, D. et HECKERMAN, D. (1994). Learning bayesian networks is NP-hard. Rapport technique MSR-TR-94-17, Microsoft Research Technical Report.

CHICKERING, D., GEIGER, D. et HECKERMAN, D. (1995). Learning bayesian networks : Search methods and experimental results. *In Proceedings of Fifth Conference on Artificial Intelligence and Statistics*, pages 112–128.

CHICKERING, D. et HECKERMAN, D. (1996). Efficient Approximation for the Marginal Likelihood of Incomplete Data given a Bayesian Network. *In UAI'96*, pages 158–168. Morgan Kaufmann.

CHICKERING, D. M. (2002). Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3 :507–554.

CHOW, C. et LIU, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3) :462–467.

COOPER, G. et HERSKOVITS, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9 :309–347.

COOPER, G. F. (1990). Computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42 :393–405.

DALY, R., SHEN, Q. et AITKEN, S. (2011). Learning bayesian networks : approaches and issues. *The Knowledge Engineering Review*, 26 :99–157.

DARWICHE, A. (2001). Decomposable negation normal form. *Journal of the ACM*, 48(4) :608–647.

DARWICHE, A. (2002). A logical approach to factoring belief networks. *In Proceedings of KR*, pages 409–420.

DARWICHE, A. (2009). *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, New York, NY, USA, 1st édition.

DARWICHE, A. et MARQUIS, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17 :229–264.

de CAMPOS, C. P. (2011). New complexity results for map in bayesian networks. *In IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain*, pages 2100–2106.

DELAPLACE, A., BROUARD, T. et CARDOT, H. (2007). Two evolutionary methods for learning bayesian network structures. *In* WANG, Y., CHEUNG, Y.-m. et LIU, H., éditeurs : *Computational Intelligence and Security*, volume 4456 de *Lecture Notes in Computer Science*, pages 288–297. Springer Berlin / Heidelberg.

FONCK, P. (1994). *Réseaux d'inférence pour le raisonnement possibiliste.* Thèse de doctorat, Université de Liège, Faculté des Sciences.

FRANCOIS, O. (2006). *De l'identification de structure de réseaux bayésiens à la reconnaissance de formes à partir d'informations complètes ou incomplètes.* Thèse de doctorat, INSA Rouen.

FRIEDMAN, N., GEIGER, D. et GOLDSZMIDT, M. (1997). Bayesian network classifiers. *Machine Learning*, 29(2-3) :131–163.

GARCIA, L. et SABBADIN, R. (2008). Complexity results and algorithms for possibilistic influence diagrams. *Artif. Intell.*, 172(8-9) :1018–1044.

GEIGER, D., VERMA, T. et PEARL, J. (1989). d-separation : From theorems to algorithms. *In Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI'89)*, pages 139–148, New York, N. Y. Elsevier Science Publishing Company, Inc.

GEIGER, D., VERMA, T. S. et PEARL, J. (1990). Identifying independence in bayesian networks. *Networks*, 20 :507–534.

GONZALES, C., PERNY, P. et QUEIROZ, S. (2008). Preference aggregation with graphical utility models. *In Proceedings of the 23rd AAAI conference on Artificial Intelligence*, pages 1037–1042.

GREINER, R., SU, X., SHEN, B. et ZHOU, W. (2002). Structural extension to logistic regression : Discriminative parameter learning of belief net classifiers. *In In Proceedings of the Eighteenth Annual National Conference on Artificial Intelligence (AAAI-02*, pages 167–173.

GROSSMAN, D. et DOMINGOS, P. (2004). Learning bayesian network classifiers by maximizing conditional likelihood. *In In ICML2004*, pages 361–368. ACM Press.

HECKERMAN, D. (1998). A tutorial on learning with bayesian network. *In* JORDAN, M. I., éditeur : *Learning in Graphical Models.* Kluwer Academic Publishers, Boston.

HECKERMAN, D., GEIGER, D. et CHICKERING, M. (1994). Learning Bayesian networks : The combination of knowledge and statistical data. *In* de MANTARAS, R. L. et POOLE, D., éditeurs : *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 293–301, San Francisco, CA, USA. Morgan Kaufmann Publishers.

HECKERMAN, D. E., HORVITZ, E. J. et NATHWANI, B. N. (1992). Toward normative expert systems : Part i. the pathfinder project. *Methods of information in medicine*, 31(2) :90–105.

HOOS, C. B. R. I. B. C. D. H. H. et POOLE, D. (2004). Cp-nets : A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research (JAIR)*, 21.

HORVITZ, E. et BARRY, M. (1995). Display of information for time-critical decision making. *In In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 296–305. Morgan Kaufmann.

HORVITZ, E., BREESE, J., HECKERMAN, D., HOVEL, D. et ROMMELSE, K. (1998). The lumiere project : Bayesian user modeling for inferring the goals and needs of software users. *In In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 256–265. Morgan Kaufmann.

HOWARD, R. A. et MATHESON, J. E. (1984). Influence diagrams. *The Principles and Applications of Decision Analysis*, 2 :720–761.

HUANGN, C. et DARWICHE, A. (1994). Inference in beleif networks : a procedural guide. *International Journal of Approximate Reasoning*, 11 :1–158.

JENSEN, F. V. (1996). *Introduction to Bayesien networks.* UCL Press, University college, London.

KENAZA, T., TABIA, K. et BENFERHAT, S. (2010). On the use of naive bayesian classifiers for detecting elementary and coordinated attacks. *Fundam. Inform.*, 105(4) :435–466.

KEOGH, E. et PAZZANI, M. (1999). Learning augmented bayesian classifiers : A comparison of distribution-based and classification-based approaches. *In Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, pages 225–230.

KIM, J. H. et PEARL, J. (1983). A computational model for causal and diagnostic reasoning in inference systems. *In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'83)*, pages 190–193, Karlsruhe (Germany).

KJAERULFF, U. (1990). Triangulation of graphs - algorithms giving small total state space. *In Technical report R-90-09.* Department of mathematical and computer science, Aalborg University (Denmark).

KOIVISTO, M. (2006). Advances in exact bayesian structure discovery in bayesian networks. *In Proc. of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*, pages 241–248.

KOIVISTO, M. et SOOD, K. (2004). Exact bayesian structure discovery in bayesian networks. *Journal of Machine Learning*, 5 :549–573.

KRUEGEL, C., VIGNA, G. et ROBERTSON, W. (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5) :717–738.

KUMAR, S. et SPAFFORD, E. H. (1994). An application of pattern matching in intrusion detection. Rapport technique CSD–TR–94–013, Department of Computer Scien'ces, Purdue University, West Lafayette.

LARRAÑAGA, P., POZA, Y., YURRAMENDI, Y., MURGA, R. et KUIJPERS, C. (1996).

Structure learning of bayesian networks by genetic algorithms : A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9) :912–926.

LAURITZEN, S. L. et SPIEGELHALTER, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50 :157–224.

LONG, W. (1989). Medical diagnosis using a probabilistic causal network. *Appl. Artif. Intell.*, 3 :367–383.

MALONE, B. M., YUAN, C., HANSEN, E. A. et BRIDGES, S. (2011). Improving the scalability of optimal bayesian network learning with external-memory frontier breadth-first branch and bound search. *In* COZMAN, F. G. et PFEFFER, A., éditeurs : *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 479–488. AUAI Press.

MARTIN, L. (1996). Autonomous control logic to guide unmanned underwater vehicle. Rapport technique, Lockheed Martin.

MEEK, C. (1995). Causal inference and causal explanation with background knowledge. *In Proceedings of 11th Conference on Uncertainty in Artificial Intelligence*, pages 403–418.

MOURAD, R., SINOQUET, C. et LERAY, P. (2011). A hierarchical bayesian network approach for linkage disequilibrium modeling and data-dimensionality reduction prior to genome-wide association studies. *BMC Bioinformatics*, 12 :16.

MURUZABAL, J. et COTTA, C. (2007). A study on the evolution of bayesian network graph structures. *In Advances in Probabilistic Graphical Models*, volume 214 de *Studies in Fuzziness and Soft Computing*, pages 193–213. Springer Berlin / Heidelberg.

PARVIAINEN, P. et KOIVISTO, M. (2009). Exact structure discovery in bayesian networks with less space. *In* BILMES, J. et NG, A. Y., éditeurs : *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 436–443. AUAI Press.

PEARL, J. (1988). *Probabilistic reasoning in intelligent systems : networks of plausible inference.* Morgan Kaufmman, San Francisco (California).

PEARL, J. (2000). *Causality : models, reasoning, and inference.* Cambridge University Press, New York, NY, USA.

PEARL, J. et PAZ, A. (1986). Graphoids : A graph-based logic for reasoning about relevance relations. *In Technical report 850038 (R-53-L).* Cognitive systems laboratory, University of California.

PEARL, J. et VERMA, T. S. (1991). A theory of inferred causation. *In* ALLEN, J. F., FIKES, R. et SANDEWALL, E., éditeurs : *Proceeding of the Second International Conference on Knowledge Representation and Reasoning (KR'91)*, pages 441–452, San Mateo, California. Morgan Kaufmann.

PEÑA, J. M., NILSSON, R., BJÖRKEGREN, J. et TEGNÉR, J. (2007). Towards scalable and data efficient learning of markov boundaries. *Int. J. Approx. Reasoning*, 45(2) : 211–232.

PERNKOPF, F. et BILMES, J. (2005). Discriminative versus generative parameter and

structure learning of bayesian network classifiers. *In Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 657–664, New York, NY, USA. ACM.

PIPATSRISAWAT, K. et DARWICHE, A. (2008). New compilation languages based on structured decomposability. *In AAAI*, pages 517–522, Chicago, USA.

PORRAS, P. A. et NEUMANN, P. G. (1997). EMERALD : Event monitoring enabling responses to anomalous live disturbances. *In Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, Maryland, USA. NIST, National Institute of Standards and Technology/National Computer Security Center.

POURRET, O., NAIM, P. et MARCOT, B. (2008). *Bayesian Networks : A Practical Guide to Applications.* Wiley.

RAIFFA, H. (1968). *Decision analysis.* Addison-Welsley Publishing Company, Toronto.

REBANE, G. et PEARL, J. (1987). The recovery of causal poly-trees from statistical data. *In Proceedings of the Third Annual Conference on Uncertainty in Artificial Intelligence (UAI'87)*, pages 175–182, New York, NY. Elsevier Science.

ROBINSON, R. W. (1977). Counting unlabeled acyclic digraphs. *In* LITTLE, C. H. C., éditeur : *Combinatorial Mathematics V*, volume 622 de *Lecture Notes in Mathematics*, pages 28–43, Berlin. Springer.

RODRIGUES DE MORAIS, S. et AUSSEM, A. (2008). A novel scalable and data efficient feature subset selection algorithm. *In Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*, ECML PKDD '08, pages 298–312, Berlin, Heidelberg. Springer-Verlag.

SCHWARTZ, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2) :461–464.

SEBYALA, A. A., OLUKEMI, T. et SACKS, L. (2002). Active platform security through intrusion detection using naive Bayesian network for anomaly detection. *In In London Communications Symposium.*

SHACHTER, R. D. (1986). Evaluating influence diagrams. *Operations Research*, 34 :871–882.

SHAFER, G. (1976). *A Mathematical Theory of Evidence.* Princeton Univ. Press. Princeton, NJ.

SHENOY, P. (1989). A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 3(5) :383–341.

SHENOY, P. (1993a). Valuation networks and conditional independence. *In UAI*, pages 191–199.

SHENOY, P. P. (1992). Using possibility theory in expert systems. *Fuzzy Sets and Systems*, 52(2) :129 – 142.

SHENOY, P. P. (1993b). Valuation networks and conditional independence. *In* HECKERMAN, D. et MAMDANI, A., éditeurs : *Uncertainty in Artificial Intelligence 93*, pages 191–199. Morgan Kaufmann, San Mateo, Ca, USA.

SIMON, C., WEBER, P. et EVSUKOFF, A. (2008). Bayesian networks inference algorithm to implement dempster shafer theory in reliability analysis. *Reliability Engineering*

and System Safety, 93 :950–963.

SMETS, P. (1998). *The transferable belief model for quantified belief representation*, volume 1, pages 267–301. Kluwer Academic Publisher.

SPIRTES, P., GLYMOUR, C. et SCHEINES, R. (1993). *Causation, prediction, and search*. Springer-Verlag.

SPIRTES, R., GLYMOUR, C. et SCHEINES, R. (2000). *Causation, Prediction, and Search*. MIT Press, Cambridge, MA.

STANIFORD, S., HOAGLAND, J. A. et MCALERNEY, J. M. (2002). Practical automated detection of stealthy portscans. *J. Comput. Secur.*, 10(1-2) :105–136.

STUDENY, M. (1990). Conditional independence relations have no finite completeness characterization.

SUNG-BAE, C. (2002). Incorporating soft computing techniques into a probabilistic intrusion detection system. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 32(2) :154–160.

TABIA, K. (2011). Possibilistic network-based classifiers : On the reject option and concept drift issues. *In Scalable Uncertainty Management*, Lecture Notes in Computer Science, pages 460–474. Springer.

TABIA, K. et LERAY, P. (2010). Bayesian network-based approaches for severe attack prediction and handling idss' reliability. *In Information Processing and Management of Uncertainty in Knowledge-Based Systems. Applications - 13th International Conference, IPMU 2010, Dortmund, Germany, June 28 - July 2, 2010. Proceedings, Part II*, pages 632–642. Springer.

TSAMARDINOS, I., ALIFERIS, C. F. et STATNIKOV, A. (2003). Time and sample efficient discovery of markov blankets and direct causal relations. *In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 673–678, New York, NY, USA. ACM.

TSAMARDINOS, I., BROWN, L. et ALIFERIS, C. (2006). The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1) :31–78.

VALDES, A. et SKINNER, K. (2000). Adaptive, model-based monitoring for cyber attack detection. *In Recent Advances in Intrusion Detection*, pages 80–92.

WANG, T. et YANG, J. (2010). A heuristic method for learning bayesian networks using discrete particle swarm optimization. *Knowl. and Info. Sys.*, 24 :269–281.

XU, H. et SMETS, P. (1994). Evidential reasoning with conditional belief functions. *In* et AL., D. H., éditeur : *UAI'94*, pages 598–606.

YE, D., HUIQIANG, W. et YONGGANG, P. (2004). A hidden markov models-based anomaly intrusion detection method. *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*, 5.

ZAAROUR, I., HEUTTE, L., LERAY, P., LABICHE, J., ETER, B. et MELLIER, D. (2004). Clustering and bayesian network approaches for discovering handwriting strategies of primary school children. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(7) :1233–1251.

Chapitre 9

Planification en intelligence artificielle

Dans ce chapitre nous proposons une revue, non exhaustive, des travaux de la communauté de l'IA autour de la planification classique et de la planification dans l'incertain. Nous présentons tout d'abord le cadre de la planification classique STRIPS propositionnelle, puis ses extensions basées sur le langage de description de problèmes PDDL devenu un standard dans la communauté. Nous traiterons ensuite brièvement de la thématique de l'analyse structurelle de problèmes, à l'origine du développement de planificateurs performants, et des principaux algorithmes de planification classique et planificateurs associés. Ensuite, nous décrirons le cadre des processus décisionnels markoviens (PDM), issu du domaine de la recherche opérationnelle, mais que la communauté de l'intelligence artificielle a mobilisé pour la planification sous incertitude. Nous décrirons enfin des algorithmes novateurs de résolution (approchée ou non) de PDM proposés récemments, ainsi que certains progrès de l'IA en termes de représentation des connaissances (logique, réseaux bayésiens) ayant été mis à profit afin d'améliorer le pouvoir d'expression du modèle PDM traditionnel, au service de la planification dans l'incertain.

9.1 Introduction

La communauté de la *planification* en intelligence artificielle s'est intéressée depuis les années 1960 à la génération de *plans* (séquences d'actions) visant à atteindre un objectif fixé pour des problèmes exprimés dans un langage concis d'opérateurs de transformation d'état, souvent proche de la logique propositionnelle (voir [Ghallab *et al.*, 2004], par exemple, pour une synthèse plus complète de cette famille d'approches). Depuis le premier système de planification à base d'opérateurs, le système GPS (« General Problem Solver ») de Newell et Simon [Newell et Simon, 1963] dont l'ambitieux objectif était de donner à une machine la capacité de simuler le raisonnement humain, le

Auteurs : Régis Sabbadin, Florent Teichteil-Königsbuch et Vincent Vidal.

domaine de la *planification classique* a connu un essor considérable (tout en revoyant ses ambitions à la baisse...) ; tant pour la richesse de modélisation des problèmes de planification que pour l'efficacité des systèmes de génération de plans (aussi appelés *planificateurs*). La planification classique repose sur deux hypothèses simplificatrices fortes : d'une part on dispose d'une connaissance parfaite, à tout instant, de l'état du système et des effets des actions ; et d'autre part les seules modifications de l'état du système proviennent de l'exécution des actions du plan. Ces hypothèses ont permis la conception de planificateurs capables de résoudre des problèmes de grande taille largement hors de portée de l'être humain.

Plus récemment, le domaine de la *planification dans l'incertain* s'est développé, proposant d'intégrer des actions à effet probabiliste puis des fonctions d'utilité additives sur les buts, conduisant à une famille d'approches pour la planification basées sur la théorie de la décision [Blythe, 1999]. Le cadre des processus décisionnels de Markov (PDM) est alors naturellement devenu l'approche privilégiée pour la représentation et la résolution de problèmes de planification dans l'incertain en intelligence artificielle. De nombreux travaux récents ont visé à améliorer ce cadre en le dotant du pouvoir expressif des langages de représentation traditionnellement utilisés en intelligence artificielle : logique, contraintes ou réseaux bayésiens. L'utilisation de tels langages de représentation fait « exploser » la complexité des algorithmes de résolution classiques des PDM. La résolution de ces problèmes *structurés* est ainsi devenue un défi pour la communauté de l'intelligence artificielle et de nombreuses approches, centralisées [Boutilier *et al.*, 2000 ; Jensen, 2001 ; Givan *et al.*, 2003] ou distribuées [Guestrin *et al.*, 2003] ont été développées ces dernières années.

Dans ce chapitre nous offrons un bref panorama des approches développées ces dernières années pour la planification, en France et à l'étranger. Nous avons centré ce panorama sur les approches basées sur le modèle STRIPS [Fikes et Nilsson, 1971] et ses extensions pour la planification classique, et sur une représentation probabiliste de l'incertitude en planification et des critères issus de la *théorie de la décision*. Dans un premier temps, nous définirons le cadre de modélisation STRIPS pour la planification classique, et décrirons ses principales extensions basées sur le langage de description de problèmes de planification PDDL [McDermott *et al.*, 1998]. Après avoir décrit les principales avancées dans la thématique de l'analyse structurelle des problèmes de planification, qui permet entre autres la conception d'heuristiques efficaces, nous établirons un panorama des principales techniques de résolution et planificateurs associés. Dans un second temps nous décrirons le cadre mathématique des *Processus Décisionnels de Markov* (PDM) [Puterman, 1994], cadre classique de représentation de problèmes de décision séquentielle dans l'incertain. Ensuite, nous montrerons comment la communauté de la planification s'est emparée de ce cadre et l'a étendu, afin de modéliser et résoudre des problèmes de planification dans l'incertain. Nous finirons ce chapitre par une brève description d'extensions du cadre des PDM, proposées ou adoptées par la communauté de l'intelligence artificielle : prise en compte de la non-observabilité de l'état du monde, apprentissage, représentations non probabilistes de l'incertitude...

9.2 La planification classique

9.2.1 Le cadre de la planification STRIPS propositionnelle

Le système fondateur STRIPS (« STanford Research Institute Problem Solver ») [Fikes et Nilsson, 1971] utilisé pour la contrôle du robot SHAKEY a posé les fondations sur lesquelles reposent encore aujourd'hui l'essentiel des travaux en planification classique. Nous nous plaçons ici dans le cadre de la planification STRIPS propositionnelle, ne faisant intervenir que des actions définies à l'aide d'un ensemble fini de symboles propositionnels atomiques. Un problème STRIPS propositionnel peut éventuellement être obtenu en instanciant des schémas d'actions décrits dans un langage plus riche, par exemple PDDL, à l'aide des constantes définies pour un problème particulier représentant les objets de l'univers.

Dans ce cadre, un *état du système* est un ensemble d'atomes issus d'un ensemble fini A représentant l'ensemble des faits de l'univers. Une *action* (ou *opérateur*, ce terme étant plus généralement utilisé pour désigner des schémas d'action contenant des variables pouvant être instanciées) est un triplet $a = \langle pr, ad, de \rangle$ où pr, ad et de dénotent des ensembles finis d'atomes de A ; $\text{prec}(a)$, $\text{add}(a)$, $\text{del}(a)$ dénotent respectivement les ensembles pr, ad, de et représentent les *préconditions*, *ajouts* et *retraits* de a. Les préconditions définissent les conditions requises dans un état donné pour pouvoir y appliquer une action : une action a est applicable sur un état s si et seulement si $\text{prec}(a) \subseteq s$. Les ajouts sont les faits que l'action va créer dans cet état, et les retraits sont ceux détruits par l'action ; l'état s' résultant de l'application d'une action a dans un état s étant défini par $s' = (s \setminus \text{del}(a)) \cup \text{add}(a)$. Un *problème de planification STRIPS propositionnel* est alors défini comme un quadruplet $\Pi = \langle A, O, I, G \rangle$ où A dénote un ensemble fini d'atomes, O dénote un ensemble fini d'actions construites à partir des atomes de A, $I \subseteq A$ dénote un ensemble fini d'atomes qui représente l'*état initial* du problème, et $G \subseteq A$ dénote un ensemble fini d'atomes qui représente le *but* du problème. Un *plan solution* est une sequence d'actions $(a_1, \ldots, a_n)$ telle que pour $s_0 = I$ et pour tout $i \in \{1, \ldots, n\}$, les états intermédiaires définis par $s_i = (s_{i-1} \setminus \text{del}(a_i)) \cup \text{add}(a_i)$ sont tels que $prec(a_i) \subseteq s_{i-1}$ et $G \subseteq s_n$.

L'objectif d'un planificateur va donc être de trouver un plan solution pour un problème donné, sachant que l'espace des états atteignables à partir d'un état initial est en général trop vaste pour être entièrement calculé et représenté en extension sous forme de graphe par exemple, où les noeuds représenteraient les états et les arcs représenteraient les actions permettant la transition d'un état vers un autre (un même état pouvant être atteint par différentes séquences d'actions). Par ailleurs, la complexité théorique du problème de décision concernant l'existence d'un plan solution pour la planification STRIPS propositionnelle est PSPACE-complet [Bylander, 1994] ; des classes polynomiales ont cependant été déterminées [Bäckström et Nebel, 1995], ainsi que des classes de problèmes pour lesquels certains algorithmes ont un comportement polynomial [Helmert, 2003, 2008]. De nombreux algorithmes ont été conçus pour réaliser une exploration partielle de l'espace des solutions, et sont généralement basés sur des techniques issues d'autres domaines de recherche : recherche heuristique, programmation par contraintes, satisfaction de bases de clauses, model checking, algorithmes évolutionnaires, etc. La *qualité* d'un plan solution est aussi un point important,

qui influence fortement le choix d'une technique donnée. Pour la planification classique, l'objectif est en général de minimiser la longueur du plan solution ; nous verrons par la suite que d'autres critères existent, liés à l'amélioration de l'expressivité du langage de description de problèmes de planification.

9.2.2 Le langage de description de problèmes PDDL

Conçu pour permettre une représentation commune des problèmes de planification pour la première compétition internationale de planification [1], le langage PDDL (« *Planning Domain Definition Language* ») a connu depuis de nombreuses évolutions. Ce langage et les compétitions associées ont permis de fédérer les recherches en planification classique, et grandement facilité la conception et le partage des meilleures techniques. Nous décrivons ci-dessous les principaux jalons de l'évolution du langage PDDL devenu un standard, liés aux compétitions successives :

PDDL 1.2 [McDermott *et al.*, 1998] : version utilisée pour les compétitions de 1998 [McDermott, 2000] et 2000 [Bacchus, 2001]. Elle est inspirée du langage utilisé par le planificateur UCPOP [Penberthy et Weld, 1992] et définit la représentation de base des opérateurs (préconditions, effets) et des problèmes (état initial, but). La représentation STRIPS des opérateurs, où les préconditions, ajouts et retraits se réduisent à des conjonctions d'atomes, est étendue par les constructions du langage ADL [Pednault, 1989] : préconditions négatives et disjonctives, quantificateurs existentiels en précondition et universels en précondition et effets, effets conditionnels. Ces extensions permettent une plus grande concision dans la représentation des problèmes, mais l'expressivité reste la même par rapport au langage STRIPS de base, vers laquelle une transformation (potentiellement coûteuse) est possible [Gazen et Knoblock, 1997].

PDDL 2.1 [Fox et Long, 2003] : version utilisée pour la compétition de 2002 [Long et Fox, 2003]. Elle introduit deux extensions majeures : la prise en compte du temps, et les variables numériques. La finesse de prise en compte des aspects temporels se situe à mi-chemin de la représentation complexe à base de chroniques temporelles utilisée par exemple dans les planificateur IxTeT [Ghallab et Laruelle, 1994] et ASPEN [Chien *et al.*, 2000] ou plus récemment dans le cadre CNT (« Constraint Networks on Timelines ») [Verfaillie *et al.*, 2010 ; Pralet et Verfaillie, 2010], et de la représentation simplifiée du planificateur ZENO [Smith et Weld, 1999] étendant la relation d'exclusion mutuelle de GRAPHPLAN [Blum et Furst, 1997]. En PDDL, les préconditions des actions dont on définit la durée d'exécution doivent être vérifiées à l'instant de début ou de fin d'une action, et/ou dans l'intervalle de temps ouvert compris entre ces instants ; les effets sont produits soit à l'instant de début, soit à l'instant de fin. La concurrence temporelle entre actions est possible à condition qu'à tout instant de l'exécution d'un plan, un même atome ne soit pas à la fois produit ou requis par une action et retiré par une autre. La concurrence peut même être nécessaire pour l'obtention d'un plan dans certains problèmes, comme en témoignent divers travaux sur la notion de planification dite temporellement expressive [Cushing *et al.*, 2007a,b]

1. International Planning Competition : http ://ipc.icaps-conference.org

pour laquelle la complexité théorique du problème de l'existence d'un plan est EXPSPACE-complet [Rintanen, 2007]. Les variables numériques sont des quantités dont une valeur (minimale, maximale ou exacte) est requise en précondition, et qui sont modifiées par l'exécution de l'action par augmentation, diminution ou assignation. Ces variables numériques servent notamment à représenter des ressources, notion couramment utilisée dans le domaine de l'ordonnancement [Laborie et Ghallab, 1995 ; Laborie, 2003]. Le langage permet aussi de définir une fonction objectif sur la qualité du plan, exprimée comme une combinaison linéaire de la durée globale d'exécution du plan (le *makespan*) et des variables numériques. L'extension de PDDL 2.1 dénommée PDDL+ [Fox et Long, 2006], inutilisée lors des compétitions, étend les notions temporelles et numériques de PDDL 2.1 à la modélisation de processus continus, permettant entre autres la représentation d'événements exogènes et une prise en compte plus fine de la modification des variables numériques (vue comme un processus continu linéaire) augmentant la possibilité de concurrence entre les actions.

PDDL 2.2 [Hoffmann et Edelkamp, 2004] : version utilisée pour la compétition de 2004 [Hoffmann et Edelkamp, 2005]. Deux extensions mineures sont proposées : les prédicats dérivés et les littéraux temporels initiaux. Les prédicats dérivés sont des atomes non affectés par les actions, dont l'apparition se produit lorsqu'une formule donnée (similaire à une précondition d'action) est vérifiée dans un état, permettant éventuellement le déclenchement de nouvelles actions. Ils fournissent entre autres un moyen concis pour représenter la fermeture transitive d'une relation, et augmentent strictement l'expressivité du langage : il peut s'avérer impossible de les compiler vers une version antérieure du langage [Thiébaux *et al.*, 2003]. Les littéraux temporels initiaux permettent la représentation d'une forme réduite d'évènements exogènes, dont la date d'apparition est connue à l'avance et exprimée dans l'état initial du problème.

PDDL 3.0 [Gerevini et Long, 2005] : version utilisée pour la compétition de 2006 [Gerevini *et al.*, 2009]. Elle introduit deux extensions importantes : les contraintes sur les trajectoires et les préférences sur les buts. Ces extensions sont motivées par le fait que dans les applications réelles, d'une part la forme d'un plan est au moins aussi importante que son obtention, et d'autre part il est souvent impossible de satisfaire entièrement tous les buts d'un problème. Les contraintes sur les trajectoires permettent de contraindre certains enchaînements d'actions et d'états intermédiaires atteints lors de l'exécution du plan, et peuvent être dures (à satisfaire obligatoirement) ou molles (leur satisfaction est associée à un poids entrant dans le calcul de la qualité globale du plan). Les préférences sur les buts permettent de différencier les buts dont l'obtention est obligatoire, des buts dont l'obtention ne l'est pas ; mais entrent, comme pour les contraintes de trajectoire molles, dans le calcul de la qualité du plan. Si les contraintes molles sont étudiées depuis longtemps en CSP [Meseguer *et al.*, 2006], leur prise en compte en planification classique est plus récente et fragmentaire.

PDDL 3.1 [Helmert *et al.*, 2008a] : version utilisée pour les compétitions de 2008 [Helmert *et al.*, 2008b] et 2011. Elle introduit les variables d'état binaires, permettant une représentation plus concise des faits de l'univers, non plus énumérés

comme des atomes de la logique propositionnelle ou du premier ordre mais comme des valeurs du domaine d'une variable d'état. Ces variables sont inspirées d'une part du formalisme SAS+ [Bäckström et Nebel, 1995] ayant permis la découverte de classes polynomiales de problèmes de planification, et d'autre part de leur intégration dans un langage plus flexible et expressif [Geffner, 2000]. L'engouement récent et fort de la communauté pour ces variables d'état, généralement extraites par une pré-compilation de problèmes exprimés en PDDL classique, est dû au fait qu'elles permettent une analyse plus fine des problèmes de planification, pour en tirer notamment des heuristiques et autres éléments comme les *landmarks* (points de passage obligatoires) [Helmert, 2004 ; Richter *et al.*, 2008 ; Karpas et Domshlak, 2009]. Un autre apport de cette version du langage est une forme réduite de variables numériques permettant d'associer un coût positif à chaque action, l'objectif étant alors de minimiser la somme des coûts des actions du plan.

Le schema d'action ci-dessous, venant du domaine TPP (« *Travelling and Purchase Problem* ») écrit pour la compétition de 2006 et inspiré d'un problème étudié en recherche opérationnelle, illustre certains des concepts que nous venons d'introduire :

```
(:durative-action drive
 :parameters (?t - truck ?from ?to - place)
 :duration (= ?duration (drive-time ?from ?to))
 :condition (and (at start (at ?t ?from)) (over all (connected ?from ?to))
                 (preference p-drive
                   (at start (forall (?g - goods)
                                (< (ready-to-load ?g ?from) 1)))))
 :effect (and (at start (not (at ?t ?from))) (at end (at ?t ?to))
              (at end (increase (total-cost) (drive-cost ?from ?to)))))
```

L'action `drive` déplace le camion `?t` du lieu `?from` vers le lieu `?to`. Les variables `?t`, `?from` et `?to` sont instanciées par des constantes représentant les objets d'un problème donné. La durée de ce déplacement dépend des lieux en question, et prend la valeur de la variable numérique `(drive-time ?from ?to)` définie dans l'état initial. Pour pouvoir appliquer cette action, le camion `?t` doit être au lieu `?from` à l'instant de début de l'action (`at start`) et les lieux `?from` et `?to` doivent rester connectés dans l'intervalle ouvert compris entre les instants de début et de fin (`over all`). La préférence `p-drive` est formulée pour représenter le fait que toutes les marchandises `?g` présentes sur le lieu de départ doivent avoir été vendues (`(ready-to-load ?g ?from)` a la valeur 0), donc ne pas être en attente d'un chargement dans un camion. Cette action a pour effet de déplacer le camion (`(at ?t ?from)` est retiré et `(at ?t ?to)` est ajouté), et le coût total du plan `(total-cost)` est augmenté du coût `(drive-cost ?from ?to)` du déplacement. La fonction objectif à minimiser peut être augmentée d'une valeur à définir si la préférence `p-drive` est violée durant l'exécution du plan.

9.2.3 L'analyse structurelle de problèmes en planification classique

L'analyse structurelle en planification consiste à extraire automatiquement des informations à partir de la description d'un problème, permettant de guider ou contraindre la recherche d'une solution vers les portions de l'espace de recherche les

plus prometteuses. On trouve les prémisses de ces techniques dans [Pearl, 1983], où une simplification manuelle de la description d'un problème est proposée pour calculer l'heuristique de Manhattan pour le problème du taquin (cf. chapitre II.1) ; plus tard, une technique automatique implémentée dans le planificateur UNPOP [McDermott, 1996] calcule par une régression à partir des buts ignorant les retraits des actions une estimation de la longueur d'un plan à partir de l'état courant de la recherche ; enfin, les idées introduites par le planificateur GRAPHPLAN [Blum et Furst, 1997] ont littéralement révolutionné le domaine.

La principale innovation apportée par GRAPHPLAN est la construction d'un graphe en chaînage avant à partir de l'état initial, dans lequel les interactions négatives entre actions (lorsqu'une action retire une précondition ou un ajout d'une autre action) ou atomes (lorsque deux atomes ne peuvent être produits que par des actions en interaction négative) ne sont considérées qu'entre paires d'actions ou d'atomes, et non pas sur l'ensemble des actions pouvant être appliquées sur un état donné. Ces paires d'actions ou d'atomes en interaction négative, aussi appelés *exclusions mutuelles* ou *mutex*, sont propagées dans le graphe et retardent l'insertion d'actions et d'atomes, fournissant là une estimation minorante de la longueur des plans pouvant les atteindre. Ce graphe étant construit en temps polynomial par rapport au nombre total d'actions et d'atomes et de leurs relations, il peut être pré-calculé avant la recherche d'une solution, voire même calculé à chaque étape d'une recherche en chaînage avant dans les espaces d'états. Le lecteur intéressé trouvera une description plus détaillée de GRAPHPLAN et des planificateurs qui s'en inspirent directement comme IPP [Koehler *et al.*, 1997] et STAN [Long et Fox, 1999] dans [Vidal, 2001] ou [Ghallab *et al.*, 2004].

Ces calculs d'heuristique ont ensuite été repris et généralisés dans [Bonet *et al.*, 1997], qui proposent une méthode polynomiale d'estimation de l'heuristique, pour le problème relaxé par la suppression des listes de retrait des actions. Cette heuristique estime la longueur d'un plan qui produit un ensemble d'atomes, en définissant le coût d'un atome a à partir d'un état s par une fonction h_s qui ajoute 1 au minimum des coûts des ensembles de préconditions des actions qui produisent a. Le coût d'un ensemble d'atomes est alors défini à l'aide d'une fonction d'agrégation des coûts des atomes qui le composent. Le coût d'un atome peut être défini récursivement, pour un état s et un atome a, par :

$$h_s(a) = \begin{cases} 0 & \text{si} \quad a \in s \\ i+1 & \text{si} \quad min_{o \in O \mid a \in add(o)}[H_s(prec(o))] = i \\ \infty & \text{sinon} \end{cases} \tag{9.1}$$

où H_s est l'heuristique calculant le coût d'un ensemble d'atomes G. Plusieurs possibilités peuvent ainsi être envisagées : l'heuristique des systèmes ASP [Bonet *et al.*, 1997], HSP, HSPr [Bonet et Geffner, 1999] et HSP2 [Bonet et Geffner, 2001] additionne le coût de chaque atome de l'ensemble G :

$$H_s^{add}(G) = \sum_{a \in G} h_s(a) \tag{9.2}$$

[Hoffmann et Nebel, 2001] proposent une amélioration de cette heuristique additive basée sur le fait que la résolution par GRAPHPLAN du problème relaxé est polyno-

miale. En effet, puisque les retraits des actions ont été suprimés, le graphe de planification ne contient aucune exclusion mutuelle, donc la procédure d'extraction de la solution est triviale (sans aucun retour arrière). Le coût d'un ensemble d'atomes est alors constitué par la somme des actions du plan parallèle qui produit ces atomes. Elle prend donc en compte les effets positifs des actions, contrairement à l'heuristique additive qui suppose que tous les sous-buts sont indépendants. Les excellentes performance du planificateur FF lors de la troisième compétition internationale de planification a inspiré la définition de plusieurs variantes comme $h^{FF/a}$ et h^{sa} [Keyder et Geffner, 2008].

Cependant, ces heuristiques ne sont pas admissibles, car elles surestiment le nombre d'actions nécessaires pour résoudre le problème relaxé. Une possibilité pour les rendre admissibles est de remplacer la somme dans l'équation 9.2 par le maximum du coût de chaque atome de l'ensemble G [Bonet et Geffner, 2001], mais celle-ci n'est pas très informative :

$$H_s^{max}(G) = max_{a \in G} h_s(a) \tag{9.3}$$

Ces heuristiques ont été étendues pour la planification optimale et la planification temporelle [Haslum et Geffner, 2000, 2001] tout en généralisant le calcul de mutex introduit par GRAPHPLAN à des ensembles d'actions et d'atomes de taille quelconque.

Les heuristiques dites *landmarks* [Koehler et Hoffmann, 2000 ; Hoffmann *et al.*, 2004], ont pour but d'estimer un ordre entre les atomes nécessaires (ou *landmarks*) dans toute solution possible d'un problème de planification. Un atome est dit nécessaire s'il est ajouté par au moins une action quelle que soit la solution du problème. La recherche de ce type d'atome est PSPACE-difficile en général [Hoffmann *et al.*, 2004]. L'idée est donc d'approximer cette recherche, en utilisant par exemple la méthode de chaînage arrière proposée dans [Hoffmann *et al.*, 2004], les graphes de transition de domaines [Richter *et al.*, 2008] basés sur la représentation SAS+ [Bäckström et Nebel, 1995], l'utilisation de l'heuristique h_s comme proposé dans [Vidal et Geffner, 2005] ou encore l'utilisation d'un arbre ET/OU [Keyder *et al.*, 2010]. Pour un état s, une heuristique dite *landmarks-possible* h_s^L peut être définie par $h_s^L = n - m + k$, où n est le nombre total de *landmarks* estimé pour le problème, m le nombre de *landmarks* atteints par le plan menant à s et k le nombre de *landmarks* déjà atteints mais devant être satisfaits une nouvelle fois. Plusieurs autres variantes ont été proposées dans [Karpas et Domshlak, 2009].

Certains travaux, comme [Helmert et Geffner, 2008] et [Helmert et Domshlak, 2009] proposent des généralisations et unifications d'heuristiques existantes. Une littérature très abondante sur le sujet de l'analyse structurelle et du calcul d'heuristiques existe, nous n'avons pu en donner ici qu'une vision très partielle.

9.2.4 Principaux algorithmes et planificateurs

Comme évoqué précédemment, la planification classique emprunte des méthodes de résolution à de nombreux autres domaines de recherche. Suivant l'objectif d'un planificateur – trouver un plan optimal pour un critère donné, optimiser ce critère sans prouver l'optimalité, trouver un plan rapidement sans optimisation –, certaines métho-

des s'avèrent plus adaptées que d'autres. Quelles que soient les méthodes de résolution choisies, les techniques d'analyse structurelle comme celles décrites précédemment s'avèrent indispensables pour obtenir de bonnes performances.

Les premiers types d'algorithmes parmi les plus employés sont les algorithmes de recherche heuristique tels A* [Hart *et al.*, 1968], IDA* [Korf, 1985] et leurs variantes comme weighted-A* [Pohl, 1970] ou EHC (« *Enforced Hill Climbing* ») [Hoffmann et Nebel, 2001] (cf. chapitre II.1). Ils sont utilisés pour effectuer une recherche dans les espaces d'états en chaînage avant ou arrière, ou bien dans les espaces de plans [Weld, 1994] où un nœud représente un plan partiel et un arc une opération de modification d'un plan partiel : introduction d'une action, relation de précédence entre actions, protection d'un lien causal. Ces algorithmes sont utilisés soit pour la planification non optimale, soit pour la planification optimale en nombre d'actions ou en coût total. Pour la planification non optimale, on peut citer les planificateurs HSP, HSPr [Bonet et Geffner, 1999] et HSP2 [Bonet et Geffner, 2001] qui utilisent h_s^{add} ; RePOP [Nguyen et Kambhampati, 2001] et VHPOP [Younes et Simmons, 2003] utilisent cette heuristique dans les espaces de plans partiels ; le planificateur FF [Hoffmann et Nebel, 2001] calcule à chaque état la longueur d'un plan pour le problème relaxé ; le planificateur YAHSP [Vidal, 2004] ajoutant au précédent la construction gloutonne d'un *plan anticipé* dont l'état résultant est ajouté à la liste des noeuds à développer ; Metric-FF [Hoffmann, 2002] et SAPA [Do et Kambhampati, 2003] pour la planification avec variables numériques ; SGPLAN [Chen *et al.*, 2006] qui partitionne un problème en plusieurs sous problèmes plus simples ; CRICKEY [Coles *et al.*, 2008] et COLIN [Coles *et al.*, 2009] pour la planification temporellement expressive ; Macro-FF [Botea *et al.*, 2005] et Marvin [Coles et Smith, 2007] qui calculent des macros-opérateurs ; TFD [Helmert, 2006] qui calcule une heuristique basée sur les graphes de transition de domaines ; et enfin LAMA [Richter et Westphal, 2010] qui utilise une heuristique basée sur les landmarks. Concernant la planification optimale en nombre d'actions ou en coût, on peut citer GAMER [Edelkamp et Kissmann, 2008] qui utilise aussi des techniques de model-cheking comme les BDDs, ou des variations de TFD avec des heuristiques améliorées [Helmert *et al.*, 2007 ; Cai *et al.*, 2009]. Enfin, on peut citer le planificateur TP4 [Haslum, 2006] pour la planification temporelle optimale.

Les approches à base de contraintes, comme la programmation par contraintes (PPC) (cf. chapitre II.6) ou la satisfaction de bases de clauses (SAT) (cf. II.5), sont très utilisées en planification. Les planificateurs SATPLAN [Kautz *et al.*, 1996] et BLACKBOX [Kautz et Selman, 1999] encodent un problème sous forme d'une base de clauses pour un horizon donné et utilisent ensuite un prouveur SAT pour trouver une solution (voir [Maris *et al.*, 2008] pour plus d'informations sur la planification SAT). Des améliorations des codages et planificateurs SAT ont récemment été proposées [Rintanen, 2010 ; Huang *et al.*, 2010]. Ces planificateurs calculent des plans parallèles optimaux, cas particulier de planification temporelle avec durées uniformes. Le planificateur CFDP [Grandcolas et Pain-Barre, 2007] utilise la PPC sur une structure inspirée du graphe de planification avec des règles d'élagage adaptées à la planification séquentielle optimale. GP-CSP [Do et Kambhampati, 2001] encode un problème sous forme de CSP et calcule un plan parallèle optimal. CPT [Vidal et Geffner, 2006] est un planificateur temporel optimal qui encode un problème en CSP suivant un schéma basé sur la pla-

nification dans les espaces de plans et introduit de nombreuses règles d'élagage basées notamment sur les actions qui ne font pas encore partie d'un plan partiel. On peut citer enfin le planificateur TLP-GP [Maris et Régnier, 2008] qui effectue une recherche dans une structure inspirée du graphe de planification encodant des contraintes temporelles, pour la planification temporellement expressive.

De façon ponctuelle, certaines autres techniques sont utilisées en planification classique : les réseaux de Petri pour la planification temporelle optimale [Hickmott *et al.*, 2007], les automates pondérés pour la planification factorisée optimale en coût [Fabre *et al.*, 2010], les algorithmes évolutionnaires [Bibai *et al.*, 2010] permettant d'optimiser la qualité des plans produits par un planificateur sous-jacent (YAHSP) par l'évolution d'une population d'individus représentant des découpages en états successifs à atteindre.

Pour finir, nous pouvons mentionner l'intérêt croissant de la communauté pour la prise en compte des évolutions récentes du matériel de calcul vers les architectures à base de processeurs multicœurs à mémoire partagée et les grilles de calcul à mémoire distribuée. Plusieurs approches ont été proposées ; notamment [Burns *et al.*, 2009 ; Vidal *et al.*, 2010] pour la planification non optimale sur machines multicœurs, le premier par un découpage en blocs de la liste des nœuds à développer et le second pour un accès concurrent de multiples *threads* à cette liste ; [Kishimoto *et al.*, 2009] pour la planification sequentielle optimale en coût pour architectures à mémoire distribuée, par distribution des nœuds suivant une fonction de hachage ; et enfin [Valenzano *et al.*, 2010] pour une approche basée sur un portefeuille de variations paramétriques d'un même algorithme s'exécutant en concurrence.

9.3 Planification probabiliste en représentation intensionnelle

Dans cette section, nous allons montrer comment le cadre de la planification déterministe a été étendu afin de prendre en compte des actions à effets incertains et des préférences sur les buts. Mais avant cela, nous décrivons brièvement le cadre des processus décisionnels de Markov (PDM) qui a servi de base sémantique à certaines des approches proposées pour la planification dans l'incertain. Les PDM sont une approche séquentielle de la décision dans l'incertain (cf. chapitre I.14), où un agent reçoit des récompenses en interagissant avec un environnement probabiliste.

9.3.1 Le cadre des processus décisionnels de Markov

Dans le cadre des *Processus Décisionnels de Markov (PDM)* [Puterman, 1994], l'espace des états est décrit *en extension* par un ensemble $\mathcal{X}$ et l'espace des actions par un second ensemble $\mathcal{A}$.

Le résultat de l'application d'une suite d'actions $(a_0, \ldots, a_{T-1})$ est une *trajectoire* du système, $\tau = (x_0, a_0, \ldots, x_{T-1}, a_{T-1}, x_T)$. Une autre spécificité du cadre des PDM est qu'il permet de modéliser une forme d'incertitude (probabiliste) sur l'effet des actions. Aussi, une suite d'actions fixée $(a_0, \ldots, a_{T-1})$ peut résulter en différentes trajectoires,

avec une distribution de probabilité connue $p(\tau|x_0, a_0, \ldots, a_{T-1})$. Le cadre des processus décisionnels de Markov diffère également de celui de la planification classique en ce sens qu'il permet de mesurer *l'utilité* d'une trajectoire τ non seulement par le fait qu'elle atteigne ou non un état *but*, mais par la prise en compte (additive) de degrés de satisfaction associés à toutes les transitions d'état (x_t, a_t, x_{t+1}).

Puisque les effets des actions ne sont plus déterministes, les effets d'un plan d'action (une simple suite d'actions) ne peuvent être connus à l'avance. Dans ce cas, des plans *réactifs*, qui définissent l'action courante à appliquer en fonction de l'observation de l'état du système, sont plus efficaces que des plans définis a priori. Dans le cadre des PDM, ces plans réactifs sont appelés *stratégies* ou *politiques*. Une *politique* $\delta = \{\delta_t\}_{t\in 0\ldots T-1}$ est un ensemble de fonctions associant à toute trajectoire partielle $(x_0, \ldots, x_t)$ l'action $a_t = \delta_t(x_0, \ldots, x_t) \in A$ appliquée par cette politique. Si l'on définit maintenant $p(\tau|x_0, \delta)$, la probabilité de suivre la trajectoire τ en appliquant la politique δ à partir de x_0 et $u(\tau)$ la récompense obtenue lorsque l'on suit la trajectoire τ, nous pouvons définir formellement l'utilité espérée d'une politique δ :

$$EU_\delta(x_0) = \sum_\tau p(\tau|x_0, \delta).u(\tau). \tag{9.4}$$

Ce critère permet de classer les politiques par ordre d'intérêt.

Résoudre un problème de décision séquentielle dans l'incertain consiste à déterminer une politique δ^* d'utilité espérée maximale, pour un état initial x_0, fixé $(EU_{\delta^*}(x_0) \geq EU_\delta(x_0), \forall\delta)$, ou pour un ensemble d'états initiaux. Le cadre des *processus décisionnels de Markov* (PDM) propose des algorithmes efficaces de calcul de politiques optimales, sous réserve que l'incertitude (mesurée par p) et les préférences (mesurées par u) sur les trajectoires satisfassent un certain nombre d'hypothèses.

Dans le cadre des PDM, l'hypothèse suivante est faite sur la probabilité conditionnelle $p(\tau|x_0, \delta)$:

$$p(\tau|x_0, \delta) = \prod_{t=0}^{T-1} p_t(x_{t+1}|x_t, \delta_t(x_0, \ldots, x_t) = a_t). \tag{9.5}$$

Cette hypothèse revient à dire que le processus stochastique défini à partir d'une politique fixée δ est *markovien*.

De même, la fonction d'utilité sur les conséquences est construite à partir de la somme de récompenses attachées aux transitions entre états :

$$\forall\tau = (x_0, a_0, \ldots, x_{T-1}, a_{T-1}, x_T), u(\tau) = \sum_{t=0}^{T-1} r_t(x_t, a_t, x_{t+1}). \tag{9.6}$$

Cette forme additive suppose que la trajectoire considérée est de longueur finie. Néanmoins, il est parfois plus commode de représenter et résoudre des problèmes de décision séquentielle dans l'incertain sur un horizon de temps infini, à condition que ces problèmes soient « stationnaires » (les probabilités de transition p_t et les récompenses r_t sont indépendantes du temps). Rien ne garantit alors que l'utilité $u(\tau)$ d'une trajectoire infinie, décrite par l'équation (9.6), soit finie. C'est pourquoi l'une des deux fonctions

d'utilité suivantes est utilisée en général pour les problèmes dont l'horizon est infini, plutôt que la forme (9.6) :

$$u(\tau) = \lim_{T\to\infty} \sum_{t=0}^{T} \gamma^t . r(x_t, a_t, x_{t+1}), 0 < \gamma < 1, \tag{9.7}$$

$$u(\tau) = \lim_{T\to\infty} \frac{1}{T} \Big(\sum_{t=0}^{T-1} r(x_t, a_t, x_{t+1}) \Big). \tag{9.8}$$

Si la fonction de récompense r est bornée, les deux expressions (9.7) et (9.8) sont bien définies, quelle que soit la longueur de la trajectoire τ. La fonction d'utilité (9.7) est appelée *critère gamma-pondéré* alors que la fonction (9.8) est appelée *critère moyen.*

Résoudre un problème de décision séquentielle dans l'incertain revient à *optimiser* le choix d'une politique. En d'autres termes, on cherche à calculer, pour un problème de décision séquentielle dans l'incertain $(\mathcal{X}, \mathcal{A}, p, r, H)$, une politique δ^* telle que :

$$EU_{\delta^*}(x_0) \geq EU_{\delta}(x_0), \forall \delta, \forall x_0 \in \mathcal{X}. \tag{9.9}$$

A priori, rien n'indique qu'une telle politique optimale existe si nous exigeons que *la même politique* δ^* soit optimale en tout état de départ x_0. Néanmoins, on peut montrer qu'une telle politique optimale existe, pour les trois critères évoqués [Bellman, 1957; Puterman, 1994]. Qui plus est, dans le cas où l'horizon est infini, il existe une politique optimale *stationnaire* (indépendante de t), $\delta^* : \mathcal{X} \to \mathcal{A}$. δ^* choisit l'action optimale en fonction du seul état courant.

Le problème d'optimisation défini par (9.9) est classiquement résolu par des méthodes de type *programmation dynamique stochastique* [Bellman, 1957; Bertsekas, 1987; Puterman, 1994]. L'algorithme de *recherche arrière* est utilisé pour résoudre des problèmes à horizon fini et les algorithmes *d'itération de la politique* et *d'itération de la valeur* sont les plus couramment utilisés pour le critère gamma-pondéré.

9.3.2 Modèles intensionnels de PDM

Le cadre classique des PDM repose sur une représentation explicite des états, c'est-à-dire où chaque état est énuméré dans une table [Puterman, 1994]. Cette modélisation présente deux inconvénients non négligeables pour des application réalistes : les états sont souvent décrits par un ensemble de variables à domaines discrets ou continus, et, à cause de l'explosion combinatoire due au nombre de combinaisons possibles des valeurs des variables, les états ne peuvent pas être énumérés avant la résolution du problème.

Ainsi, depuis une dizaine d'années, des modèles de PDM basés sur une représentation factorisée de l'espace d'états par variables ont été développés. Ces modèles sont dits *intensionnels*, car ni les états ni les transitions ni les récompenses ne sont construits a priori, mais sont définis par des formules logiques portant sur les variables d'état du modèle. Chaque formule peut être *instanciée* en remplaçant les variables d'état par leurs valeurs à un instant donné, ce qui permet de construire au besoin l'état correspondant à ces variables instanciées, ainsi que les transitions ou les récompenses définies sur cet état.

Nous présentons ici les deux principaux modèles intensionnels de PDM : les réseaux bayésiens dynamiques [Dean et Kanazawa, 1990 ; Boutilier *et al.*, 2000, 1998 ; Hoey *et al.*, 1999 ; St-Aubin *et al.*, 2000 ; Feng et Hansen, 2002 ; Feng *et al.*, 2003] et les modèles STRIPS probabilistes [Younes *et al.*, 2005 ; Yoon *et al.*, 2007, 2008 ; Teichteil-Königsbuch *et al.*, 2010 ; Buffet et Aberdeen, 2009 ; Kuter et Nau, 2005 ; Kolobov *et al.*, 2010a ; Joshi *et al.*, 2010].

Réseaux bayésiens dynamiques

Un réseau bayésien dynamique (cf. chapitre I.12 et [Dean et Kanazawa, 1990]) est un graphe orienté dont les nœuds sont des variables d'état rangées par niveaux temporels, et les arcs indiquent les dépendances bayésiennes entre les variables d'état (voir figure 1). Chaque niveau t contient toutes les variables d'état à l'instant t, et il ne peut y avoir d'arcs qu'entre variables de deux niveaux successifs ou d'un même niveau : par conséquent, les transitions entre deux états successifs du système — représentés par leurs variables d'état aux instants t et $t+1$ — sont nécessairement markoviennes. Si les transitions ne dépendent pas du temps, le réseau bayésien ne contient que deux niveaux t et $t+1$. Dans ce cas particulier dit *homogène*, il convient de noter les variables à l'instant t sous la forme $(X_1, \cdots, X_n)$ et les variables à l'instant $t+1$ sous la forme primée $(X'_1, \cdots, X'_n)$.

Les arcs du réseau bayésien dynamique renseignent sur la dépendance événementielle des variables d'état entre deux instants successifs, mais pas sur la probabilité de ces événements. Une table de probabilité est donc associée à tout réseau bayésien dynamique pour définir les probabilités de chaque variable X'_m à l'instant $t+1$. Formellement, si une variable m à l'instant $t+1$ dépend de variables $X_{j_1}, \cdots, X_{j_k}$ à l'instant t et de variables $X'_{i_1}, \cdots, X'_{i_l}$ à l'instant $t+1$, alors cela se traduit par des arcs des variables $X_{j_1}, \cdots, X_{j_k}$ et $X'_{i_1}, \cdots, X'_{i_l}$ vers la variable X'_m, et l'entrée $P(X'_m \mid X'_{i_1}, \cdots, X'_{i_l}, X_{j_1}, \cdots, X_{j_k})$ dans la table de probabilité.

Toutefois, la dépendance entre les variables dépend de leurs valeurs : suivant la valeur de X'_m, l'événement probabiliste associé à cette valeur ne dépend pas des mêmes variables. Ainsi, le réseau bayésien n'indique pour chaque variable primée qu'une dépendance maximale parmi toutes les valeurs de la variable primée. Il en est de même de la probabilité $P(X'_m \mid X'_{i_1}, \cdots, X'_{i_l}, X_{j_1}, \cdots, X_{j_k})$, dont les variables de dépendance pour une valeur donnée de X'_m appartiennent à un sous-ensemble de $\{X'_{i_1}, \cdots, X'_{i_l}, X_{j_1}, \cdots, X_{j_k}\}$. La structure de cette probabilité peut être assez compliquée, si bien qu'elle est elle-même représentée par un arbre de décision ou, plus généralement, par un diagramme de décision algébrique [Bahar *et al.*, 1997].

L'extension des réseaux bayésiens dynamiques aux PDM intensionnels est relativement simple. À chaque action du PDM est associé un réseau bayésien dynamique qui représente les probabilités de transition entre les variables d'état à deux instants successifs. La structure des récompenses, qui est généralement différente de celle des probabilités de transition, est définie par un autre réseau bayésien dynamique dont les arcs représentent les récompenses gagnées ou perdues par l'agent autonome pour chaque valeur des variables d'état à l'instant $t+1$ en fonction des variables d'état à l'instant t ou $t+1$. Le modèle suppose donc que la récompense gagnée ou perdue en arrivant dans un état s' est égale à la somme des récompenses gagnées ou perdues pour

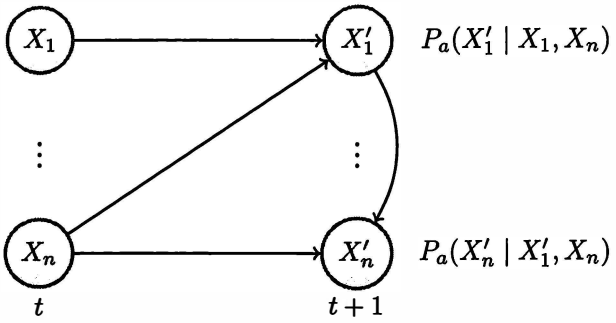


FIGURE 1 – Exemple de réseau bayésien dynamique homogène

chaque valeur des variables d'état de s'.

Modèle STRIPS probabiliste

Depuis 2004, sous l'impulsion de la première compétition internationale de planification probabiliste [Younes *et al.*, 2005], les communautés de planification classique et de processus décisionnels de Markov (PDM) se sont considérablement rapprochées. Partant du langage PDDL développé pour les compétitions de planification classique [Fox et Long, 2003], des effets probabilistes ont été rajoutés au langage afin de modéliser une sous-classe des PDM dans ce langage, nommé PPDDL (« *Probabilistic PDDL* »). Il en résulte une modélisation intensionnelle des PDM sous forme d'opérateurs STRIPS [Fikes et Nilsson, 1971] probabilistes. De nombreux planificateurs basés sur ce langage ont été développés depuis, permettant en l'espace de quelques années d'augmenter de plusieurs ordres de grandeur la taille des problèmes résolus et de diminuer tout autant les temps de calcul et la mémoire nécessaire.

Le langage PPDDL, bâti sur son prédécesseur PDDL, reprend et étend les éléments suivants :

- domaine défini en intension à l'aide de prédicats logiques ;
- définition séparée du domaine, décrivant la dynamique des actions, et du problème, définissant les objets du monde qui instancient les prédicats du domaine ;
- variables d'état représentées sous forme de prédicats logiques ;
- actions représentées sous forme de prédicats logiques ;
- domaines d'application des actions définis à l'aide d'une formule logique du premier ordre à valeur booléenne ;
- effets probabilistes des actions définis à l'aide de formules logiques du premier ordre, décrivant pour chaque effet probabiliste les variables devenant vraies ou fausses, pondérées par les probabilités d'occurrence des effets correspondants ;
- ensemble d'états initiaux possibles représentés par un ensemble de prédicats instanciés initialement vrais ;
- récompenses associées à chaque effet probabiliste, si le problème à résoudre consiste à maximiser le critère gamma-pondéré ;

— ensemble d'états buts définis sous la forme d'une formule logique du premier ordre, si le problème à résoudre consiste à atteindre le plus rapidement possible un ensemble de buts avec la plus grande probabilité ou le plus faible coût moyen.

De plus haut niveau que les réseaux bayésiens dynamiques, le langage PPDDL est moins expressif, mais il peut être exponentiellement plus compact. En effet, chaque variable d'un réseau bayésien dynamique correspond à une instanciation d'un prédicat du langage PPDDL pour un ensemble d'objets du problème. Considérons par exemple un problème consistant à empiler des cubes dans un ordre particulier. Le prédicat `on(?b1,?b2)` indique si un bloc `b1` quelconque est sur un bloc `b2` quelconque. En supposant qu'il y ait n blocs dans le problème, ce seul prédicat représente à lui seul $n(n-1)$ variables d'état du problème. Il est clair que le nombre de variables instanciées (telles que définies dans un réseau bayésien dynamique) augmente exponentiellement avec le nombre d'objets du problème. Toutefois, ce gain en compacité suppose que les variables du problème sont binaires et indépendantes sémantiquement, ce qui n'est pas le cas de nombreux problèmes : dans le « monde des grilles » par exemple, il est impossible que l'agent se trouve indépendamment dans plusieurs cases au même instant.

Le langage PPDDL suppose que les transitions sont indépendantes du temps et, contrairement aux réseaux bayésiens dynamiques, les probabilités de transition portent sur plusieurs variables d'état primées (à l'instant suivant) à la fois. Reprenons l'exemple du « monde des blocs », où il est possible qu'un bloc `b1` explose ou qu'un bloc `b2` soit détruit quand l'agent tente d'empiler `b2` sur `b1` en appliquant l'action ci-dessous :

```
(:action put-on-block
   :parameters (?b1 ?b2 - block)
   :precondition (and (holding ?b1) (clear ?b2) (no-destroyed ?b2))
   :effect (and (emptyhand) (on ?b1 ?b2) (not (holding ?b1)) (not (clear ?b2))
                (probabilistic 1/10 (when (no-detonated ?b1)
                     (and (not (no-destroyed ?b2)) (not (no-detonated ?b1)))))))
```

Le code PPDDL précédent indique que l'action `put-on-block`, quels que soient les blocs `b1` et `b2`, n'est applicable que si l'agent tient `b1` dans sa main, qu'il n'y a aucun bloc sur `b2` et que `b2` n'est pas détruit. Cette action a pour effet de libérer la main de l'agent, de mettre `b1` sur `b2` et, avec une probabilité de 0.1, et si le bloc `b1` n'avait pas explosé avant le début de l'action, à la fois de ne pas détruire le bloc `b2` et de ne pas faire exploser le bloc `b1`.

Des traducteurs automatiques de PPDDL vers des réseaux bayésiens dynamiques ont été développés [Younes *et al.*, 2005], mais ils se heurtent à une combinatoire exponentielle de la transformation dans le pire cas, ainsi qu'à un calcul difficile des probabilités de transition isolées pour chaque variable à l'instant suivant. Ce dernier point nécessite d'ailleurs, dans de nombreux cas, de rajouter au réseau bayésien dynamique des variables d'état fictives qui représentent le couplage synchrone entre les variables réelles du problème, augmentant d'autant plus la complexité de la résolution du PDM.

Avant de clore ce paragraphe, notons l'existence d'un nouveau langage de modélisation des PDM, utilisé depuis la compétition internationale de planification probabiliste en 2011. Il s'agit du langage RDDL (« *Relational Dynamic influence Diagram Language* ») [Sanner, 2010], basé sur la logique du premier ordre comme PPDDL, mais

non centré sur les actions contrairement à ce dernier. Ainsi, il est plus aisé de modéliser en RDDL des effets communs à toutes les actions, ou indépendants des actions. D'un point de vue sémantique, les expressions de RDDL se rapprochent des réseaux bayésiens dynamiques, dans la mesure où les variables d'état sont mises à jour de manière individuelle suite aux effets des actions (contrairement à PPDDL, où les variables d'état sont mises à jour par groupes).

9.3.3 Algorithmes et planificateurs

De nombreux algorithmes ont été développés pour résoudre des PDM intensionnels. Ils se rangent en deux catégories fondamentalement différentes au niveau des méthodes utilisées : les approches probabilistes [Boutilier *et al.*, 2000 ; Hoey *et al.*, 1999 ; St-Aubin *et al.*, 2000 ; Feng et Hansen, 2002 ; Feng *et al.*, 2003 ; Joshi *et al.*, 2010 ; Kuter et Nau, 2005 ; Buffet et Aberdeen, 2009], qui étendent les méthodes de résolution des PDM classiques aux PDM intensionnels, et les approches déterministes [Yoon *et al.*, 2007 ; Teichteil-Königsbuch *et al.*, 2010 ; Yoon *et al.*, 2008 ; Kolobov *et al.*, 2010a], qui réduisent la résolution du PDM intensionnel à plusieurs résolutions de problèmes de planification classique déterministe.

Approches probabilistes

Cette catégorie d'algorithmes repose essentiellement sur l'utilisation de diagrammes de décision algébriques (ADD) [Bahar *et al.*, 1997], qui sont des structures de données compactes pour représenter des fonctions de variables booléennes à valeurs réelles. Ils permettent de représenter efficacement les tables de probabilité ou de récompense des réseaux bayésiens dynamiques, ainsi que la fonction de valeur optimisée. Ils sont souvent utilisés conjointement aux diagrammes de décision binaires (OBDD) [Bryant, 1986], qui représentent des fonctions booléennes à valeurs booléennes, et qui sont donc particulièrement indiqués pour les préconditions des actions ou la politique d'action optimisée.

De nombreux algorithmes de résolution de PDM classiques ont été adaptés aux PDM intensionnels modélisés sous forme de réseaux bayésiens dynamiques, en redéfinissant les opérateurs algébriques de l'équation de Bellman sur la base des opérateurs fournis par les diagrammes de décision algébriques ou binaires. Contrairement aux PDM classiques, les PDM intensionnels basés sur les diagrammes de décision travaillent non pas explicitement sur des états individuels, mais en intension sur plusieurs états à la fois à l'aide de formules logiques portant sur les variables d'état du problème. La principale difficulté de l'adaptation des algorithmes classiques aux PDM intensionnels basés sur les diagrammes de décision réside donc dans la nécessité de repenser globalement l'équation de Bellman (sur des ensembles d'états) et non localement (sur chaque état individuel). Les principaux algorithmes développés dans ce cadre sont : SPUDD [Hoey *et al.*, 1999] (adaptation de l'itération de la valeur), APRICODD [St-Aubin *et al.*, 2000] (approximation de SPUDD), sLAO* [Feng et Hansen, 2002], sRTDP [Feng *et al.*, 2003] et FODD-PLANNER [Joshi *et al.*, 2010]. Ce dernier repose sur les diagrammes de décision du premier ordre [Wang *et al.*, 2008], qui sont une extension des diagrammes de décision pour représenter des fonctions à valeurs booléennes dont les variables sont des

formules logiques portant sur les variables d'état du problème. Cette nouvelle structure, plus informée que les diagrammes de décision sur la structure du PDM à résoudre, permet de trouver des solutions optimales en utilisant moins de ressources de calcul.

Néanmoins, les approches reposant sur les diagrammes de décision souffrent souvent de quelques inconvénients qui peuvent devenir rédhibitoires en pratique. Premièrement, ces structures de données nécessitent d'ordonner les variables dans la structure pour maintenir la structure compacte et garantir des temps d'accès convenables. Or, l'ordre des variables influe grandement sur la taille de la structure et donc sur la performance des opérations sous-jacentes ; en pratique, l'utilisateur d'un algorithme de résolution de PDM intensionnel basé sur ces structures doit fournir un ordre de variables garantissant de bonnes performances pour chaque problème, ce qui n'est pas toujours aisé. Deuxièmement, les algorithmes de résolution de PDM basés sur ces structures de données, et donc sur le modèle des réseaux bayésiens dynamiques, nécessitent de connaître les probabilités marginales, c'est-à-dire les probabilités de chaque variable isolée à l'instant suivant (cf. figure 1). Ceci présente deux inconvénients majeurs indépendants : d'abord, ce modèle introduit des cycles potentiels dans les variables de l'instant $t+1$ qu'il est nécessaire d'analyser afin de calculer correctement la probabilité de l'état suivant [Boutilier *et al.*, 1998] ; ensuite, une traduction depuis le langage PPDDL, où les probabilités de l'instant $t+1$ portent sur plusieurs variables à la fois, requiert de rajouter des variables d'état artificielles pour modéliser la dépendance entre les variables de l'instant $t+1$ [Younes *et al.*, 2005]. Troisièmement, une traduction depuis le langage PPDDL peut créer également de nombreuses variables d'état inutiles supplémentaires, dues à l'instanciation a priori de tous les prédicats du domaine pour tous les objets du problème, qu'ils soient atteignables ou non en pratique. Par exemple, dans le « monde des blocs », le prédicat `on(b1,b1)`, qui signifie que le bloc `b1` est empilé sur lui-même, ne devient jamais vrai mais il est pourtant construit et rajouté au réseau bayésien dynamique.

Par ailleurs, d'autres algorithmes de résolution des PDM intensionnels basés sur les réseaux bayésiens dynamiques définissent la politique comme une fonction paramétrée, qui représente la probabilité de choisir une action en fonction des valeurs des variables d'état. Ces algorithmes reposent sur la simulation de Monte-Carlo de la politique courante en partant d'un état initial connu, et apprennent les paramètres de la politique en fonction des observations reçues à l'aide de méthodes de descente de gradient. L'algorithme FPG [Buffet et Aberdeen, 2009], qui a remporté la compétition internationale de planification probabiliste en 2006, se base sur des réseaux de neurones pour représenter la politique et apprendre ses paramètres. Une version améliorée [Buffet et Aberdeen, 2007] utilise le planificateur déterministe FF comme une heuristique pour guider les trajectoires des simulations de Monte-Carlo vers les buts du problème à résoudre. Cette idée est au cœur des approches déterministes, que nous présentons dans le paragraphe suivant.

Enfin, récemment, certaines approches proposent de générer à la volée une partie du graphe des états atteignables, à partir d'une description intensionnelle d'un PDM. Bien que certaines méthodes reposent sur des techniques de simulation stochastique [Keller et Eyerich, 2012], la majorité de ces approches est basée sur des algorithmes de recherche heuristique [Bonet et Geffner, 2005 ; Teichteil-Königsbuch *et al.*, 2011 ;

Teichteil-Königsbuch, 2012 ; Kolobov *et al.*, 2012]. L'heuristique, qui est calculée en analysant la structure des opérateurs du langage de modélisation (PPDDL ou RDDL), permet de guider l'expansion du graphe vers les états les plus prometteurs.

Approches déterministes

Cette deuxième catégorie de méthodes comprend les approches de type « diviser pour régner » en divisant le problème probabiliste en plusieurs problèmes déterministes. Chaque action du PDM est transformée soit en une action déterministe dont l'effet est l'effet probabiliste le plus probable, soit en autant d'actions déterministes que d'effets probabilistes. Ces méthodes se sont développées depuis la première compétition internationale de planification probabiliste en 2004, car l'utilisation du langage PPDDL, basé sur PDDL, permet une interface très simple avec des planificateurs déterministes existants. Plus précisément, un domaine (probabiliste) PPDDL est transformé en plusieurs domaines (déterministes) PDDL, qui sont résolus par plusieurs appels à un même planificateur déterministe. Toutefois, les algorithmes diffèrent par la façon d'exécuter ou d'agréger les plans produits dans un contexte probabiliste.

Le premier algorithme de ce type, qui a remporté la première compétition internationale de planification probabiliste, est FF-REPLAN [Yoon *et al.*, 2007]. Cette approche est purement réactive : après avoir généré un domaine (déterministe) PDDL, l'algorithme appelle le planificateur déterministe FF [Hoffmann et Nebel, 2001] depuis l'état initial et exécute le plan solution ; si le plan échoue à cause des effets probabilistes de l'action courante, l'algorithme relance FF depuis l'état d'échec courant et exécute le nouveau plan depuis cet état. Cet algorithme, pourtant aveugle quant aux récompenses et aux états d'échec — dont les états puits ou « culs-de-sac » — a obtenu de très bons résultats sur l'ensemble des problèmes, tant en qualité qu'en temps de calcul. En fait, les problèmes de cette première compétition probabiliste avaient été simplement adaptés de la compétition déterministe, sans que les effets probabilistes n'influent sensiblement sur l'optimisation de la solution du PDM : n'importe quel plan relativement court avait une chance élevée d'atteindre le but du problème, quitte à replanifier souvent en cas d'échec. Afin de remédier à ce problème, des problèmes avec des états puits atteignables avec une probabilité élevée ont été rajoutés dans les compétitions suivantes.

Plutôt que d'appeler le planificateur déterministe à chaque échec du plan indépendamment des risques futurs, le planificateur RFF [Teichteil-Königsbuch *et al.*, 2010] agrège au sein d'une politique des plans produits par FF depuis plusieurs états qui peuvent être atteints en partant de l'état initial. Un faisceau de plans convergeant vers l'état but est ainsi construit de manière incrémentale jusqu'à ce que la probabilité d'échec à l'exécution des plans agrégés soit inférieure à un seuil donné. Cette probabilité d'échec est calculée par simulation de Monte-Carlo en partant de l'état initial et en suivant la politique agrégée courante. À chaque fois que la simulation de la politique courante atteint avec une probabilité suffisante des états pour lesquels aucune action n'est planifiée, le planificateur FF est appelé et le plan produit depuis cet état est rajouté dans la politique courante. Dans certaines configurations de l'algorithme RFF, la politique agrégée courante peut être optimisée de sorte à maximiser la probabilité d'atteindre le but en évitant les états puits. RFF a remporté la compétition internationale

de planification probabiliste en 2008, en ayant de bonnes performances tant en qualité qu'en temps de calcul pour la majorité des problèmes, y compris ceux comprenant des états puits atteignables avec une forte probabilité.

Enfin, d'autres algorithmes entremêlent une résolution classique de l'équation de Bellman avec une génération de plans déterministes à l'aide de FF. Ces approches ont l'avantage de synthétiser une politique optimale, car le planificateur déterministe est utilisé en quelque sorte comme heuristique de guidage vers la politique optimale. Le planificateur FF-HINDSIGHT [Yoon *et al.*, 2008, 2010] optimise la fonction de valeur par améliorations successives de la politique, qui est une agrégation de plans déterministes produits par FF, et qui est évaluée par simulation de Monte-Carlo en partant de l'état initial. Cet algorithme peut être considéré comme l'intégration de RFF dans un schéma d'itération de la politique. La planificateur RETRASE [Kolobov *et al.*, 2009, 2010a,b] approche la fonction de valeur par une combinaison linéaire de fonctions de base, qui sont automatiquement générées et paramétrées en utilisant des évaluations fournies par le planificateur déterministe FF. Ces deux algorithmes ont démontré de très bonnes performances en qualité sur les problèmes des dernières compétitions internationales de planification probabiliste.

9.4 Autres extensions du cadre des PDM en intelligence artificielle

En dehors de l'hypothèse de représentation des états et décisions *en extension*, d'autres hypothèses pesant sur le cadre des processus décisionnels de Markov ont dû être levées pour le rendre utilisable en planification dans l'incertain :

— L'hypothèse d'observabilité complète de l'état du monde à chaque instant.
— L'hypothèse (différente de la précédente) de connaissance parfaite du modèle (transitions, récompenses). En effet, parfois ce modèle n'est accessible qu'indirectement, par simulation ou expérimentation.
— Parfois également, seules des évaluations « qualitatives » des préférences et des connaissances sont disponibles.

Pour pallier ces différentes limitations, plusieurs pistes ont été suivies, que nous allons décrire succinctement. Pour plus de détails, le lecteur est invité à se référer aux deux ouvrages de synthèse [Buffet et Sigaud, 2008a,b]

9.4.1 Processus décisionnels de Markov partiellement observables

Dans de nombreux problèmes de décision séquentielle dans l'incertain, l'agent n'a pas une connaissance complète de l'état réel de son environnement, contrairement à ce qui est supposé dans le cadre des PDM. Ce sont ses actions qui, en plus de modifier l'état du système, lui apportent des informations permettant de préciser sa connaissance du monde. Ainsi, on peut distinguer deux types d'effets des actions, en environnement partiellement observable :

— Effet *physique* des actions : une action a_t, appliquée par l'agent à l'instant t, transforme l'état du monde x_t en un état x_{t+1} éventuellement de manière stochastique, suivant une probabilité de transition $p_t(x_{t+1}|x_t, a_t)$.
— Effet *épistémique* des actions (voir aussi le chapitre I.12) : l'état résultant x_{t+1} n'est pas observé directement, mais une *observation indirecte* o, élément d'un ensemble d'observations $\mathcal{O}$ est retournée par le système. Cette observation peut être « déterministe », dans le cas où il existe une fonction $O : \mathcal{X} \times \mathcal{A} \to \mathcal{O}$ telle que $o = O(x_{t+1}, a_t)$. Cette observation peut, plus généralement, être stochastique. Dans ce cas, $O(x_{t+1}, a_t, o) = p(o|x_{t+1}, a_t)$ est la probabilité d'observer o lorsque l'action a_t est appliquée et résulte en x_{t+1}.

La prise en compte de ces deux types d'effets conduit à étendre le cadre des processus décisionnels de Markov [Aström, 1965 ; Sondik, 1978]. La connaissance imparfaite de l'état du monde à l'instant t est modélisée par un *état de croyance* $b_t \in \mathcal{B}$, où $\mathcal{B}$ est l'ensemble des distributions de probabilité sur $\mathcal{X}$. b_0 est l'état de croyance initial et pour $t \geq 0$, $b_t(x)$ représente la probabilité associée par l'état de croyance à l'état x. Cet état de croyance évolue sous l'effet combiné des actions et observations successives.

Une première approche pour la résolution d'un PDMPO consiste à associer directement à chaque observation possible de l'état du monde une action. Mais cette méthode associe la même action à tous les états du monde conduisant à la même observation. Or, en général, de telles politiques sont sous-optimales [Littman, 1994]. Afin de déterminer une politique tirant au mieux parti des observations imparfaites, il est nécessaire de différencier des états conduisant à la même observation en prenant en compte les états rencontrés précédemment. Ceci peut être fait (dans le cas où l'horizon est fini) en définissant une politique δ_t déterminant l'action à effectuer à l'instant t, non pas simplement en fonction de la dernière observation o_{t-1}, mais en fonction de *l'historique* des actions et observations, $\{a_1, o_1, \ldots, a_{t-1}, o_{t-1}\}$.

Dans le cas où l'horizon est infini, bien qu'un PDMPO puisse être traduit en un PDM classique, la nature du problème obtenu (espace d'états continu) rend impossible sa résolution par les méthodes classiques de programmation dynamique. Cette complexité a été à la source de plusieurs voies de recherche en intelligence artificielle, et plusieurs familles d'algorithmes de résolution de PDMPO ont été proposées :

— Une première famille de méthodes met à profit la structure (linéaire par morceaux et convexe) de la fonction de valeur $V : \mathcal{B} \to \mathbb{R}$ pour proposer des algorithmes de type *itération de la valeur*. Ces algorithmes utilisent une structure d'arbre pour représenter une politique et sa fonction de valeur associée et définir une méthode itérative de calcul de la fonction de valeur (approchée) [Kaelbling *et al.*, 1998].
— Les méthodes de résolution les plus récentes sont basées sur une discrétisation de l'espace d'états, ou combinent les deux types d'approches [Pineau *et al.*, 2003], [Smith et Simmons, 2005].

9.4.2 Processus décisionnels de Markov et apprentissage

Le cadre des processus décisionnels de Markov permet de représenter et résoudre des problèmes de planification dans l'incertain. Grâce à la programmation dynamique, le surcroît de complexité de la résolution de ces problèmes lié à leur aspect séquentiel

est limité. Nous venons d'indiquer qu'il est possible d'étendre le cadre des PDM à la planification en environnement « partiellement observable ». On parle parfois dans le cadre des PDM d'observabilité partielle dans un sens différent alors que l'état du monde est parfaitement connu à chaque instant. Il s'agit du cas où le *modèle* du PDM est imparfaitement connu, i.e. lorsque les fonctions p et r du modèle $< \mathcal{X}, A, p, r >$ sont inconnues a priori mais accessibles par expérimentation, soit parce qu'on peut *simuler* la dynamique du système, soit parce qu'on peut l'expérimenter en temps réel. Les méthodes de type *apprentissage par renforcement* (voir par exemple le chapitre I.9 et le chapitre III.8) visent à résoudre de tels problèmes dans lesquels le « modèle » du PDM est appris en même temps que sa solution optimale. Pour ce faire, il existe deux types de méthodes : les méthodes *indirectes* et les méthodes *directes*.

Les *méthodes indirectes* [Kumar et Varaiya, 1986 ; Sutton, 1991 ; Peng et Williams, 1993 ; Moore et Atkeson, 1993] supposent d'apprendre dans un premier temps (par simulation ou expérimentation) le modèle (p, r) du PDM, puis de le résoudre par un algorithme de Programmation Dynamique. De manière un peu plus évoluée, on peut focaliser l'effort lié à l'apprentissage du modèle sur des zones de l'espace d'états-actions $(\mathcal{X} \times A)$ prometteuses, tout en ne négligeant pas totalement le reste de l'espace d'états-actions, afin de garantir qu'on ne passe pas à côté d'une politique optimale. Les méthodes indirectes permettent de résoudre des PDM dont on ne connaît pas le modèle, à la condition de pouvoir expérimenter ou simuler ce modèle. Ces méthodes présentent toutefois un inconvénient : elles nécessitent de stocker au moins partiellement les fonctions $\hat{p}$ et $\hat{r}$, ce qui peut nécessiter jusqu'à $O(|\mathcal{X}|^2|A|)$ d'espace de stockage.

Les *méthodes directes* [Sutton, 1988 ; Watkins, 1989 ; Watkins et Dayan, 1992] permettent de se passer de stocker le modèle (p, r) en entier et de ne garder que ce qui est nécessaire à l'évaluation de politiques ou au calcul de politiques optimales. En contrepartie, des expérimentations / simulations plus nombreuses peuvent être nécessaires. Le choix d'une méthode directe sera donc préféré lorsque les simulations ont un coût faible, et qu'un problème de taille mémoire peut se poser.

Les méthodes directes et indirectes entrelacent en général apprentissage et programmation dynamique, afin de gagner en efficacité.

9.4.3 Approches qualitatives pour les PDM

Un apport de l'IA à la théorie de la décision en général a été la proposition et l'étude de critères de décision alternatifs au critère traditionnel de l'utilité espérée. Parmi ces critères de décision alternatifs, des critères « qualitatifs », plus adaptés à certaines problématiques de l'IA (élicitation de connaissances / préférences, communication homme / machine), ont été utilisés dans le cadre de la planification dans l'incertain.

[da Costa Pereira *et al.*, 1997] ont proposé une approche de la planification dans l'incertain basée sur la théorie des possibilités. Celle-ci utilise des distributions de possibilité pour représenter les effets des actions, mais se place dans un cadre *non observable* et avec des préférences non graduelles sur des états buts. En conséquence, des plans *inconditionnels* maximisant le degré de possibilité ou de nécessité d'atteindre un état but sont calculés. Les problèmes sont modélisés dans un langage logique de type STRIPS. Par la suite, la théorie des possibilités a été utilisée afin de définir une contrepartie qualitative des processus décisionnels de Markov [Sabbadin *et al.*, 1998 ; Sabbadin,

2001]. Récemment, ces travaux ont été étendus pour offrir un langage structuré et des algorithmes de résolution adaptés pour la planification dans l'incertain [Garcia et Sabbadin, 2008]. D'autres extensions de ce cadre sont présentées dans [Mouaddib *et al.*, 2008].

9.5 Conclusion

Dans ce chapitre nous avons proposé une revue, forcément sommaire et non exhaustive, des travaux de la communauté de l'IA autour de la planification classique et de la planification dans l'incertain. Nous avons tout d'abord présenté le cadre de la planification classique STRIPS propositionnelle, puis présenté ses extensions basées sur le langage de description de problèmes PDDL devenu un standard dans la communauté. Nous avons traité brièvement la thématique de l'analyse structurelle de problèmes, à l'origine du développement de planificateurs performants, et décrit les principaux algorithmes de planification classique et planificateurs associés. Nous avons ensuite mis l'accent sur le cadre des processus décisionnels markoviens (PDM) que la communauté de l'intelligence artificielle s'est approprié afin de l'utiliser en planification sous incertitude. A cet effet des algorithmes novateurs de résolution approchée ou non de PDM ont été proposés. De plus, certains des points forts de l'IA en termes de représentation des connaissances (logique, réseaux bayésiens) ont été mis à profit afin d'améliorer le faible pouvoir d'expression du modèle PDM traditionnel.

Nous avons tenté dans ce chapitre de présenter brièvement les aspects saillants de ces diverses approches et proposé des pointeurs vers des articles les traitant de manière plus approfondie. Les lecteurs intéressés par la planification classique, les processus décisionnels de markov et leurs utilisations en IA sont invités à se pencher sur l'abondante source bibliographique (souvent anglo-saxonne) existant sur le sujet.

Nous avons dû faire des choix sur les approches décrites dans ce chapitre. Nous avons en particulier laissé de côté un pan entier, très actif, de la recherche en planification en intelligence artificielle, celui de la *planification multiacteur*. Le lecteur intéressé par ce domaine pourra par exemple s'orienter vers des références générales sur la planification distribuée [Durfee, 1999] ou sur la planification multiagent [Vlassis, 2009 ; Shoham et Leyton-Brown, 2009] et ses liens avec les MDP [Beynier *et al.*, 2010 ; Canu et Mouaddib, 2011].

Références

Akström, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10 :174–205.

Bacchus, F. (2001). The 2000 AI planning systems competition. *AI Magazine*, 22(3) : 47–56.

Bäckström, C. et Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, 11(4) :625–655.

BAHAR, R. I., FROHM, E. A., GAONA, C. M., HACHTEL, G. D., MACII, E., PARDO, A. et SOMENZI, F. (1997). Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2-3) :171–206.

BELLMAN, R. E. (1957). *Dynamic Programming*. Princeton University Press, NJ, USA.

BERTSEKAS, D. P. (1987). *Dynamic Programming : Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ, USA.

BEYNIER, A., CHARPILLET, F., SZER, D. et MOUADDIB, A. (2010). *Markov Decision Processes and Artificial Intelligence*, chapitre DEC-MDP/POMDP, pages 321–359. Wiley.

BIBAI, J., SAVÉANT, P., SCHOENAUER, M. et VIDAL, V. (2010). An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. *In Proc. ICAPS*, pages 18–25.

BLUM, A. L. et FURST, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2) :279–298.

BLYTHE, J. (1999). An overview of planning under uncertainty. *AI magazine*, 20(2) :37–54.

BONET, B. et GEFFNER, H. (1999). Planning as heuristic search : New results. *In Proc. ECP*, pages 359–371.

BONET, B. et GEFFNER, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1-2) :5–33.

BONET, B. et GEFFNER, H. (2005). mGPT : A probabilistic planner based on heuristic search. *JAIR*, 24 :933–944.

BONET, B., LOERINCS, G. et GEFFNER, H. (1997). A robust and fast action selection mechanism for planning. *In Proc. AAAI*, pages 714–719.

BOTEA, A., ENZENBERGER, M., MÜLLER, M. et SCHAEFFER, J. (2005). Macro-FF : Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24 :581–621.

BOUTILIER, C., BRAFMAN, R. I. et GEIB, C. W. (1998). Structured reachability analysis for Markov decision processes. *In Proc. UAI*, pages 24–32.

BOUTILIER, C., DEARDEN, R. et GOLDSZMIDT, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2) :49–107.

BRYANT, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8) :677–691.

BUFFET, O. et ABERDEEN, D. (2007). FF + FPG : Guiding a policy-gradient planner. *In Proc. ICAPS*, pages 42–48.

BUFFET, O. et ABERDEEN, D. (2009). The factored policy-gradient planner. *Artificial Intelligence*, 173(5-6) :722–747.

BUFFET, O. et SIGAUD, O., éditeurs (2008a). *Processus décisionnels de Markov en intelligence artificielle - vol. 1.* Traité IC2 - Informatique et systèmes d'information. Hermes - Lavoisier, Cachan, France.

BUFFET, O. et SIGAUD, O., éditeurs (2008b). *Processus décisionnels de Markov en intelligence artificielle - vol. 2.* Traité IC2 - Informatique et systèmes d'information.

Hermes - Lavoisier, Cachan, France.

BURNS, E., LEMONS, S., ZHOU, R. et RUML, W. (2009). Best-first heuristic search for multi-core machines. *In Proc. IJCAI*, pages 449–455.

BYLANDER, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2) :165–204.

CAI, D., HOFFMANN, J. et HELMERT, M. (2009). Enhancing the context-enhanced additive heuristic with precedence constraints. *In Proc. ICAPS*.

CANU, A. et MOUADDIB, A. (2011). Dynamic local interaction model : framework and algorithms. *In 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11), Workshop of Multi-Agent Sequential Decision Making in Uncertain Multi-Agent Domains (MSDM)*.

CHEN, Y., HSU, C. et WAH, B. (2006). Temporal planning using subgoal partitioning and resolution in SGPlan. *Artificial Intelligence*, 26 :323–369.

CHIEN, S., RABIDEAU, G., KNIGHT, R., SHERWOOD, R., ENGELHARDT, B., MUTZ, D., ESTLIN, T., SMITH, B., FISHER, F., BARRETT, T., STEBBINS, G. et TRAN, D. (2000). ASPEN - Automating space mission operations using automated planning and scheduling. *In Proceedings of the International Conference on Space Operations (SpaceOps)*.

COLES, A., COLES, A., FOX, M. et LONG, D. (2009). Temporal planning in domains with linear processes. *In Proc. IJCAI*, pages 1671–1676.

COLES, A., FOX, M., LONG, D. et SMITH, A. (2008). Planning with problems requiring temporal coordination. *In Proc. AAAI*, pages 892–897.

COLES, A. et SMITH, K. A. (2007). Marvin : A heuristic search planner with online macro-action learning. *Journal of Artificial Intelligence Research*, 28 :119–156.

CUSHING, W., KAMBHAMPATI, S., MAUSAM et WELD, D. S. (2007a). When is temporal planning really temporal ? *In Proc. IJCAI*, pages 1852–1859.

CUSHING, W., WELD, D. S., KAMBHAMPATI, S., MAUSAM et TALAMADUPULA, K. (2007b). Evaluating temporal planning domains. *In Proc. ICAPS*, pages 105–112.

DA COSTA PEREIRA, C., GARCIA, F., LANG, J. et MARTIN-CLOUAIRE, R. (1997). Planning with graded non deterministic actions : A possibilistic approach. *International Journal of Intelligent Systems*, 12 :935–962.

DEAN, T. et KANAZAWA, K. (1990). A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3) :142–150.

DO, M. et KAMBHAMPATI, S. (2003). Sapa : A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20 :155–194.

DO, M. B. et KAMBHAMPATI, S. (2001). Planning as constraint satisfaction : Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2).

DURFEE, E. H. (1999). Distributed problem solving and planning. *In Multiagent Systems : A Modern Approach to Distributed Artificial Intelligence*, pages 121–164. MIT Press.

EDELKAMP, S. et KISSMANN, P. (2008). GAMER : Bridging planning and general game playing with symbolic search. *In Booklet of the 6th International Planning Competition*.

FABRE, E., JEZEQUEL, L., HASLUM, P. et THIÉBAUX, S. (2010). Cost-optimal factored planning : Promises and pitfalls. *In Proc. ICAPS*, pages 65–72.

FENG, Z. et HANSEN, E. A. (2002). Symbolic heuristic search for factored Markov decision processes. *In Proc. AAAI*, pages 455–460.

FENG, Z., HANSEN, E. A. et ZILBERSTEIN, S. (2003). Symbolic generalization for on-line planning. *In Proc. UAI*, pages 209–216.

FIKES, R. et NILSSON, N. J. (1971). STRIPS : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4) :189–208.

FOX, M. et LONG, D. (2003). PDDL2.1 : An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20 :61–124.

FOX, M. et LONG, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27 :235–297.

GARCIA, L. et SABBADIN, R. (2008). Complexity results and algorithms for possibilistic influence diagrams. *Artificial Intelligence*, 172(8-9) :1018–1044.

GAZEN, B. et KNOBLOCK, C. (1997). Combining the expressiveness of UCPOP with the efficiency of Graphplan. *In Proc. ECP*, pages 221–233.

GEFFNER, H. (2000). Functional STRIPS : A more flexible language for planning and problem solving. *In* MINKER, J., éditeur : *Logic-Based Artificial Intelligence*, pages 187–209. Kluwer, Alphen aan den Rijn, Netherlands.

GEREVINI, A., HASLUM, P., LONG, D., SAETTI, A. et DIMOPOULOS, Y. (2009). Deterministic planning in the 5th international planning competition : PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6) :619–668.

GEREVINI, A. et LONG, D. (2005). Plan constraints and preferences in PDDL3. Rapport technique RT 2005-08-47, Dept. of Electronics for Automation, University of Brescia, Italy.

GHALLAB, M. et LARUELLE, H. (1994). Representation and control in IxTeT, a temporal planner. *In Proc. AIPS*, pages 61–67.

GHALLAB, M., NAU, D. et TRAVERSO, P. (2004). *Automated planning : Theory and practice*. Morgan Kaufmann, San Francisco, CA, USA.

GIVAN, R., DEAN, T. et GREIG, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1-2) :163–223.

GRANDCOLAS, S. et PAIN-BARRE, C. (2007). Filtering, decomposition and search space reduction for optimal sequential planning. *In Proc. AAAI*, pages 993–998.

GUESTRIN, C., KOLLER, D., PARR, R. et VENKATARAMAN, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19 :399–468.

HART, P., NILSSON, N. et RAPHAEL, B. (1968). A formal basis for the heuristic determination of minimum-cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 4(2) :100–107.

HASLUM, P. (2006). Improving heuristics through relaxed search - an analysis of TP4 and HSP*a in the 2004 planning competition. *Journal of Artificial Intelligence Research*, 25 :233–267.

HASLUM, P. et GEFFNER, H. (2000). Admissible heuristics for optimal planning. *In Proc. AIPS*, pages 70–82.

HASLUM, P. et GEFFNER, H. (2001). Heuristic planning with time and resources. *In Proc. ECP*, pages 121–132.

HELMERT, M. (2003). Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2) :219–262.

HELMERT, M. (2004). A planning heuristic based on causal graph analysis. *In Proc. ICAPS*, pages 161–170.

HELMERT, M. (2006). The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26 :191–246.

HELMERT, M. (2008). *Understanding Planning Tasks.* Springer Verlag, Berlin, Germany.

HELMERT, M., DO, M. B. et REFANIDIS, I. (2008a). IPC-2008, deterministic part : Changes in PDDL 3.1. http ://ipc08.icaps-conference.org/PddlExtension.

HELMERT, M., DO, M. B. et REFANIDIS, I. (2008b). IPC-2008, deterministic part : Results. http ://ipc08.icaps-conference.org/Results.

HELMERT, M. et DOMSHLAK, C. (2009). Landmarks, critical paths and abstractions : What's the difference anyway ? *In Proc. ICAPS.*

HELMERT, M. et GEFFNER, H. (2008). Unifying the causal graph and additive heuristics. *In Proc. ICAPS*, pages 140–147.

HELMERT, M., HASLUM, P. et HOFFMANN, J. (2007). Flexible abstraction heuristics for optimal sequential planning. *In Proc. ICAPS*, pages 176–183.

HICKMOTT, S. L., RINTANEN, J., THIÉBAUX, S. et WHITE, L. B. (2007). Planning via petri net unfolding. *In Proc. IJCAI*, pages 1904–1911.

HOEY, J., ST-AUBIN, R., HU, A. J. et BOUTILIER, C. (1999). SPUDD : Stochastic planning using decision diagrams. *In Proc. UAI*, pages 279–288.

HOFFMANN, J. (2002). Extending FF to numerical state variables. *In Proc. ECAI*, pages 571–575.

HOFFMANN, J. et EDELKAMP, S. (2004). PDDL2.2 : The language for the classical part of IPC-4. *In Booklet of the 4th International Planning Competition.*

HOFFMANN, J. et EDELKAMP, S. (2005). The deterministic part of IPC-4 : An overview. *Journal of Artificial Intelligence Research*, 24 :519–579.

HOFFMANN, J. et NEBEL, B. (2001). The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14 :253–302.

HOFFMANN, J., PORTEOUS, J. et SEBASTIA, L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22 :215–278.

HUANG, R., CHEN, Y. et ZHANG, W. (2010). A novel transition based encoding scheme for planning as satisfiability. *In Proc. AAAI.*

JENSEN, F. V. (2001). *Bayesian networks and decision graphs.* Springer Verlag, Berlin, Germany.

JOSHI, S., KERSTING, K. et KHARDON, R. (2010). Self-taught decision theoretic planning with first order decision diagrams. *In Proc. ICAPS*, pages 89–96.

KAELBLING, L. P., LITTMAN, M. L. et CASSANDRA, A. R. (1998). Planning and acting in partially observable domains. *Artificial Intelligence*, 101(1-2) :99–134.

KARPAS, E. et DOMSHLAK, C. (2009). Cost-optimal planning with landmarks. *In Proc. IJCAI*, pages 1728–1733.

KAUTZ, H., MCALLESTER, D. et SELMAN, B. (1996). Encoding plans in propositional logic. *In Proc. KR*, pages 374–384.

KAUTZ, H. et SELMAN, B. (1999). Unifying SAT-based and graph-based planning. *In Proc. IJCAI*, pages 318–325.

KELLER, T. et EYERICH, P. (2012). Prost : Probabilistic planning based on uct. *In ICAPS*.

KEYDER, E. et GEFFNER, H. (2008). Heuristics for planning with action costs revisited. *In Proc. ECAI*, pages 588–592.

KEYDER, E., RICHTER, S. et HELMERT, M. (2010). Sound and complete landmarks for and/or graphs. *In Proc. ECAI*, pages 335–340.

KISHIMOTO, A., FUKUNAGA, A. S. et BOTEA, A. (2009). Scalable, parallel best-first search for optimal sequential planning. *In Proc. ICAPS*, pages 10–17.

KOEHLER, J. et HOFFMANN, J. (2000). On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm. *Journal of Artificial Intelligence Research*, 12 :338–386.

KOEHLER, J., NEBEL, B., HOFFMANN, J. et DIMOPOULOS, Y. (1997). Extending planning graphs to an ADL subset. *In Proc. ECP*, pages 273–285.

KOLOBOV, A., DAI, P., MAUSAM et WELD, D. S. (2012). Reverse iterative deepening for finite-horizon MDPs with large branching factors. *In ICAPS*.

KOLOBOV, A., MAUSAM et WELD, D. S. (2009). ReTrASE : Integrating paradigms for approximate probabilistic planning. *In Proc. IJCAI*, pages 1746–1753.

KOLOBOV, A., MAUSAM et WELD, D. S. (2010a). Classical planning in MDP heuristics : With a little help from generalization. *In Proc. ICAPS*, pages 97–104.

KOLOBOV, A., MAUSAM et WELD, D. S. (2010b). SixthSense : Fast and reliable recognition of dead ends in MDPs. *In Proc. AAAI*, pages 1108–1114.

KORF, R. (1985). Depth-first iterative-deepening : an optimal admissible tree search. *Artificial Intelligence*, 27(1) :97–109.

KUMAR, P. R. et VARAIYA, P. P. (1986). *Stochastic Systems : Estimation, Identification and Adaptive Control*. Prentice Hall, Englewood Cliffs, NJ, USA.

KUTER, U. et NAU, D. S. (2005). Using domain-configurable search control for probabilistic planning. *In Proc. AAAI*, pages 1169–1174.

LABORIE, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling. *Artificial Intelligence*, 143(2) :151–188.

LABORIE, P. et GHALLAB, M. (1995). Planning with sharable resources constraints. *In Proc. IJCAI*, pages 1643–1649.

LITTMAN, M. L. (1994). Memoryless policies : Theoretical limitations and practical results. *In Proc. of the 3rd International Conference on Simulation and Adaptive Behavior*, pages 238–245.

LONG, D. et FOX, M. (1999). The efficient implementation of the plan-graph. *Journal of Artificial Intelligence Research*, 10 :85–115.

LONG, D. et FOX, M. (2003). The 3rd international planning competition : Results and analysis. *Journal of Artificial Intelligence Research*, 20 :1–59.

MARIS, F. et RÉGNIER, P. (2008). TLP-GP : Solving temporally-expressive planning problems. *In Proc. TIME*, pages 137–144.

MARIS, F., RÉGNIER, P. et VIDAL, V. (2008). Planification par satisfaction de bases de clauses. *In* SAÏS, L., éditeur : *Problème SAT : défis et challenges*, chapitre 11, pages 289–309. Hermes.

MCDERMOTT, D. (1996). A heuristic estimator for means-ends analysis in planning. *In Proc. AIPS*, pages 142–149.

MCDERMOTT, D. (2000). The 1998 AI planning systems competition. *AI Magazine*, 21(2) :35–56.

MCDERMOTT, D., GHALLAB, M., HOWE, A., KNOBLOCK, C., RAM, A., VELOSO, M., WELD, D. et WILKINS, D. (1998). PDDL – The Planning Domain Definition Language. Rapport technique CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CI, USA.

MESEGUER, P., ROSSI, F. et SCHIEX, T. (2006). Soft constraints. *In* ROSSI, F., van BEEK, P. et WALSH, T., éditeurs : *Handbook of Constraint Programming*, chapitre 9, pages 281–328. Elsevier, Amsterdam, Netherlands.

MOORE, A. W. et ATKESON, C. G. (1993). Prioritized sweeping : Reinforcement learning with less data and less real time. *Machine Learning*, 13 :103–130.

MOUADDIB, A. I., PRALET, C., SABBADIN, R. et WENG, P. (2008). Processus décisionnels de Markov et critères non classiques. *In* BUFFET, O. et SIGAUD, O., éditeurs : *Processus décisionnels de Markov en intelligence artificielle - vol 1*, chapitre 5. Hermes - Lavoisier, Cachan, France.

NEWELL, A. et SIMON, H. (1963). GPS : A program that simulates human thought. *In* FEIGENBAUM, E. et FELDMAN, J., éditeurs : *Computers and Thought*, pages 279–293. McGraw Hill, New-York, NY, USA.

NGUYEN, X. et KAMBHAMPATI, S. (2001). Reviving partial order planning. *In Proc. IJCAI*, pages 459–466.

PEARL, J. (1983). *Heuristics*. Addison Wesley, Reading, MA, USA.

PEDNAULT, E. P. D. (1989). ADL : Exploring the middle ground between STRIPS and the situation calculus. *In Proc. IJCAI*, pages 324–332.

PENBERTHY, J. et WELD, D. (1992). UCPOP : A sound, complete, partial order planner for ADL. *In Proc. KR*, pages 103–114.

PENG, J. et WILLIAMS, R. J. (1993). Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4) :437–454.

PINEAU, J., GORDON, G. et THRUN, S. (2003). Point-based value iteration : An anytime algorithm for POMDPs. *In Proc. IJCAI*, pages 1025–1032.

POHL, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3) :193–204.

PRALET, C. et VERFAILLIE, G. (2010). Réseaux de contraintes sur des chronogrammes pour la planification et l'ordonnancement. *Revue d'Intelligence Artificielle*, 24(4) : 485–504.

PUTERMAN, M. L. (1994). *Markov decision processes : Discrete stochastic dynamic programming*. John Wiley & Sons, New York, NY, USA.

RICHTER, S., HELMERT, M. et WESTPHAL, M. (2008). Landmarks revisited. *In Proc. AAAI*, pages 975–982.

RICHTER, S. et WESTPHAL, M. (2010). The LAMA planner : Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39 : 127–177.

RINTANEN, J. (2007). Complexity of concurrent temporal planning. *In Proc. ICAPS*, pages 280–287.

RINTANEN, J. (2010). Heuristics for planning with SAT. *In Proc. CP*, pages 414–428.

SABBADIN, R. (2001). Possibilistic Markov decision processes. *Engineering Applications of Artificial Intelligence*, 14 :287–300.

SABBADIN, R., FARGIER, H. et LANG, J. (1998). Towards qualitative approaches to multi-stage decision making. *International Journal of Approximate Reasoning*, 19(3-4) :441–471.

SANNER, S. (2010). Relational Dynamic Influence Diagram Language (RDDL) : Language Description. `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf`.

SHOHAM, Y. et LEYTON-BROWN, K. (2009). *Multiagent Systems : Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York.

SMITH, D. E. et WELD, D. S. (1999). Temporal planning with mutual exclusion reasoning. *In Proc. IJCAI*, pages 326–337.

SMITH, T. et SIMMONS, R. (2005). Point-based POMDP algorithms : Improved analysis and implementation. *In Proc. UAI*, pages 542–547.

SONDIK, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon : Discounted costs. *Operations Research*, 26(2) :282–304.

ST-AUBIN, R., HOEY, J. et BOUTILIER, C. (2000). APRICODD : Approximate policy construction using decision diagrams. *In Proc. NIPS*, pages 1089–1095.

SUTTON, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3(1) :9–44.

SUTTON, R. (1991). Planning by incremental dynamic programming. *In Proc. of the 8th International Workshop on Machine Learning*, pages 353–357.

TEICHTEIL-KÖNIGSBUCH, F. (2012). Stochastic safest and shortest path problems. *In AAAI*.

TEICHTEIL-KÖNIGSBUCH, F., KUTER, U. et INFANTES, G. (2010). Incremental plan aggregation for generating policies in MDPs. *In Proc. AAMAS*, pages 1231–1238.

TEICHTEIL-KÖNIGSBUCH, F., VIDAL, V. et INFANTES, G. (2011). Extending classical planning heuristics to probabilistic planning with dead-ends. *In Proc. AAAI'11*.

THIÉBAUX, S., HOFFMANN, J. et NEBEL, B. (2003). In defense of PDDL axioms. *In*

Proc. IJCAI, pages 961–968.

VALENZANO, R., STURTEVANT, N., SCHAEFFER, J., BURO, K. et KISHIMOTO, A. (2010). Simultaneously searching with multiple settings : An alternative to parameter tuning for suboptimal single-agent search algorithms. *In Proc. ICAPS*, pages 177–184.

VERFAILLIE, G., PRALET, C. et LEMAÎTRE, M. (2010). How to model planning and scheduling problems using constraint networks on timelines. *Knowledge Eng. Review*, 25(3) :319–336.

VIDAL, V. (2001). *Recherche dans les graphes de planification, satisfiabilité et stratégies de moindre engagement. Les systèmes LCGP et LCDPP.* Thèse de doctorat, IRIT, Université Paul Sabatier, Toulouse, France.

VIDAL, V. (2004). A lookahead strategy for heuristic search planning. *In Proc. ICAPS*, pages 150–160.

VIDAL, V., BORDEAUX, L. et HAMADI, Y. (2010). Adaptive K-parallel best-first search : A simple but efficient algorithm for multi-core domain-independent planning. *In Proc. 3rd Symposium on Combinatorial Search (SOCS).*

VIDAL, V. et GEFFNER, H. (2005). Solving simple planning problems with more inference and no search. *In Proc. CP*, pages 682–696.

VIDAL, V. et GEFFNER, H. (2006). Branching and pruning : An optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, 170(3) :298–335.

VLASSIS, N. (2009). Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

WANG, C., JOSHI, S. et KHARDON, R. (2008). First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31(1) :431–472.

WATKINS, C. J. (1989). *Learning from Delayed Rewards.* Thèse de doctorat, King's College, Cambridge, UK.

WATKINS, C. J. et DAYAN, P. (1992). Q-Learning. *Machine Learning*, 3(8) :279–292.

WELD, D. S. (1994). An introduction to least commitment planning. *AI Magazine*, 15(4) :27–61.

YOON, S. W., FERN, A. et GIVAN, R. (2007). FF-Replan : A baseline for probabilistic planning. *In Proc. ICAPS*, pages 352–359.

YOON, S. W., FERN, A., GIVAN, R. et KAMBHAMPATI, S. (2008). Probabilistic planning via determinization in hindsight. *In Proc. AAAI*, pages 1010–1016.

YOON, S. W., RUML, W., BENTON, J. et DO, M. B. (2010). Improving determinization in hindsight for on-line probabilistic planning. *In Proc. ICAPS*, pages 209–217.

YOUNES, H. L. S., LITTMAN, M. L., WEISSMAN, D. et ASMUTH, J. (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, 24 :851–887.

YOUNES, H. L. S. et SIMMONS, R. G. (2003). VHPOP : Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20 :405–430.

Chapitre 10

Algorithmique de l'apprentissage et de la fouille de données

L'apprentissage artificiel est essentiellement centré sur l'étude et la mise au point de méthodes permettant la recherche de régularités à partir de données correspondant à des cas particuliers. Pour ce problème général d'induction, de nombreux algorithmes ont été développés depuis les premiers travaux dans les années mille neuf cent cinquante. Ce chapitre introduit d'abord le problème de l'induction et l'importance en particulier de l'espace des hypothèses considéré, qui est lié à la représentation des connaissances jugée adéquate. C'est alors en fonction des caractéristiques générales de ces espaces d'hypothèses que sont présentées les grandes familles d'algorithmes d'apprentissage existants. La conclusion esquisse les nouvelles questions et pistes de recherche qui définissent la nouvelle frontière de la recherche en apprentissage artificiel.

10.1 Introduction

Apprendre, cela signifie faire sens du monde, y découvrir des régularités, des lois permettant de comprendre et/ou de prédire, cela fait également référence à la capacité d'adaptation, c'est-à-dire d'auto-modification afin d'améliorer sa performance. Le fait que le programme d'un ordinateur puisse, à l'instar du reste de la mémoire, être considéré par la machine comme des données modifiables a très tôt fasciné les pionniers de l'intelligence artificielle (voir l'excellent ouvrage de Nilsson [Nilsson, 2010]), ce qui a conduit à une floraison de travaux exploratoires durant les années cinquante à soixante-dix environ. Cependant, il faut reconnaître que le souci des applications, et en particulier l'exploitation des bases de données, a finalement focalisé l'intérêt du côté de l'apprentissage de motifs ou de régularités au sein de données. C'est donc principalement sur ce type d'apprentissage, le plus étudié et le mieux maîtrisé aujourd'hui,

Auteurs : Antoine Cornuéjols, Christel Vrain et Jean-Daniel Zucker.

que porte ce chapitre (voir le chapitre I.9 pour ce qui est d'autres algorithmes portant davantage sur l'adaptation et l'apprentissage par renforcement, ainsi que les chapitres III.6 et III.7 pour des techniques d'apprentissage développées pour la bioinformatique ou la vision et la robotique). Ces algorithmes d'apprentissage sont désormais au cœur des processus de fouille de données et leur compréhension est essentielle pour qui veut appréhender comment sont analysées les quantités de données [1] dont la progression est devenue exponentielle (par exemple les données post-génomiques ou bien toutes les données issues de nos téléphones mobiles).

Les premiers algorithmes d'apprentissage sont apparus en même temps que l'intelligence artificielle (IA) il y a plus de soixante ans. De même que pour l'histoire de l'IA, on peut dégager cinq grandes phases dans le développement de l'apprentissage artificiel. Très brièvement, ce sont les suivantes. La première est celle de l'enthousiasme des années 1950 et 1960 pendant lesquelles les premiers algorithmes d'apprentissage sont inspirés par la nature et la psychologie cognitive. Ainsi sont développés les prolégomènes de l'apprentissage par renforcement (celui du jeu de dames par Samuel en 1959) et de l'apprentissage neuro-mimétique (celui du perceptron par Rosenblatt 1957 [Rosenblatt, 1958]) voient le jour et semblent promettre un succès rapide pour la réalisation d'apprentissage à partir de percepts (e.g. images de rétine sous forme matricielle, positions au jeu de dames, etc.). Les années 1970 sont liées à un « âge de raison ». L'ouvrage de Minsky et Papert (1969) [Minsky et Papert, 1988] montre les limitations de l'algorithme du perceptron soulignant notamment le problème de l'apprentissage de représentations adéquates en amont d'un système adaptatif tel que le perceptron. Ce sont aussi les années d'euphorie en I.A. avec les systèmes experts et les premiers grands systèmes d'apprentissage symbolique, dont ARCH [Winston, 1970], l'apprentissage par espace des versions [Mitchell, 1979], ou encore ACT qui cherche à simuler l'apprentissage des enfants dans le domaine de l'algèbre par exemple [Anderson *et al.*, 1979 ; Anderson, 1995].

Suivent les années 1980 qui correspondent à une phase de renaissance et d'un nouvel engouement pour toutes les formes de l'apprentissage artificiel : algorithmes d'apprentissage de règles, d'arbre de décision, supervisé, non supervisé, par renforcement, par analogie, etc. De très nombreux « paradigmes » sont ainsi explorés : apprentissage par analogie, par découverte d'heuristiques, par généralisation dans un espace de versions, par analyse d'explications, par chunking, etc. (voir par exemple [Michalski *et al.*, 1986 ; Kodratoff et Michalski, 1990] et la traduction française [Michalski *et al.*, 1993]). La quatrième phase est celle de la maturité où, d'une part, les algorithmes existants sont mieux compris, mieux analysés et où, d'autre part, la théorie statistique de l'apprentissage fournit de nouveaux outils pour mieux comprendre la convergence des algorithmes. On commence alors à délaisser le modèle humain de l'apprentissage. La phase actuelle, des années 2000 à nos jours, se caractérise par un développement très rapide des techniques d'apprentissage artificiel qui irriguent tous les domaines de l'IA ainsi que de très nombreux secteurs industriels, commerciaux et financiers. D'un point de vue conceptuel, l'apprentissage est considéré comme le déroulement d'algorithmes d'optimisation heuristique ou exacte pour lesquels les chercheurs ont maintenant le recul nécessaire

1. On parle aujourd'hui de plus en plus de bases de données massives ou Big Data pour lesquels l'analyse ne peut se faire sans les algorithmes d'apprentissage et de fouille de données

pour bien en comprendre leur complexité et l'origine de leur capacité d'apprentissage.

Que ce soit en vue de prédire ou de comprendre l'environnement, l'apprentissage a pour objectif de découvrir une fonction de décision ou un modèle du monde, celui-ci pouvant prendre la forme d'une théorie ou d'une distribution de probabilités par exemple. Or l'espace des fonctions de décision ou celui des modèles est immensément vaste. Dès lors l'inférence d'une fonction ou d'un modèle particulier à partir des données disponibles pose la double question de l'exploration de cet espace et de l'estimation de la qualité des fonctions ou modèles envisageables par cette exploration. Il faut donc, d'une part, définir *un critère inductif* permettant de jauger les fonctions ou modèles en fonction des observations et de toute autre connaissance préalable, et, d'autre part, trouver *un moyen de guider une recherche dans un espace de modèles* du monde. C'est ce double défi qui est au cœur de l'apprentissage et celui qui motive les algorithmes développés.

Les problèmes d'apprentissage étudiés aujourd'hui, les sources d'informations possibles et les tâches visées sont d'une grande variété, même si on est encore loin de pouvoir rendre compte de tous les types d'apprentissages naturels, comme, par exemple, le développement de l'intelligence chez l'enfant [Woolfolk, 2001]. Nous avons essayé d'adopter dans ce chapitre un point de vue aussi générique que possible, nous concentrant sur les concepts fondamentaux et les grandes approches algorithmiques, elles-mêmes très liées aux types d'espaces de fonctions ou modèles considérés.

10.1.1 Recherche de régularités dans des données

L'apprentissage met en jeu des observations, notées $\mathbf{x}$, éventuellement associées à une étiquette y (dans ce cas on utilisera parfois aussi la notation $\mathbf{z} = (\mathbf{x}, y)$). Ces observations (et étiquettes) peuvent être de nature très diverse : données sur les clients d'une entreprise, données sur le génome de patients atteints ou non d'une certaine pathologie, descriptions de documents sur internet, etc. Elles peuvent donc être décrites soit sous forme vectorielle, ceci se fera de manière assez naturelle si les données sont stockées dans des bases de données relationnelles classiques, soit non vectorielle, comme c'est le cas pour la description de molécules ou de documents. De manière générale, on notera $\mathcal{X}$ l'espace des observations, et $\mathcal{X} \times \mathcal{Y}$ l'espace des observations et des étiquettes possibles si celles-ci sont disponibles. Les données d'apprentissage, aussi appelées *échantillon d'apprentissage*, forment un ensemble (éventuellement un multi-ensemble si il y a répétition), noté $\mathcal{S}$, pris dans $\mathcal{X}^m$ ou dans $(\mathcal{X} \times \mathcal{Y})^m$. Par exemple, un échantillon de données étiquetées prendra la forme : $\mathcal{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$ s'il y a m données d'apprentissage.

Le but de l'apprentissage est soit de permettre la prise de décision lorsqu'une nouvelle situation $\mathbf{x}$ arrive, ce qui revient souvent à savoir lui associer une étiquette, soit d'aider à la compréhension du monde par la mise à jour de régularités ou d'hypothèses permettant d'expliquer ou de rendre compte des observations connues. On notera $\mathcal{H}$ l'*espace des hypothèses* candidates.

L'apprentissage revient donc à associer à un échantillon de données $\mathcal{S}$ une ou plusieurs hypothèses les plus à même de nous rendre performant à l'avenir. Un *algorithme d'apprentissage* $\mathcal{A}$ peut ainsi être considéré comme une fonction (déterministe ou non)

$\mathcal{A} : \mathcal{S} \mapsto h$ définie de $\mathcal{X}^m$ (ou $(\mathcal{X}^m \times \mathcal{Y})^m$) $\rightarrow \mathcal{H}$ associant à un échantillon d'apprentissage $\mathcal{S}$ de taille m une hypothèse h (ou la combinaison de plusieurs hypothèses).

Ces hypothèses ou régularités sont de nature très variée en fonction du problème posé. Ainsi, on peut chercher :

- *des nouveaux descripteurs*, comme par exemple, la recherche d'axes principaux d'inertie (par analyse en composantes principales). En calculant les principaux axes d'inertie du nuage de points correspondant aux données supposées décrites dans un espace géométrisable, elle permettra en particulier de voir si les variables de description sont corrélées, ce qui peut conduire par exemple à une redescription des données selon de nouveaux axes.
- des *motifs fréquents*. Ce sont des motifs (conjonctions de couples attribut-valeur) suffisamment représentés (à l'aune d'un seuil prédéfini appelé *support*) dans la table des données. Cela pourrait être par exemple (*âge* > 60) $\wedge$ (*HDL-cholesterol* $> 1{,}65$ *mmol/L*).
- des *règles d'association* : règles de la forme $I \rightarrow J$, où I et J sont des motifs et $J \neq \varnothing$. On veut bien sûr des règles « intéressantes », ce que l'on traduit par diverses mesures.
- des *catégories* dans les données. On parle aussi de *classification non supervisée* ou de *clustering* lorsque les catégories prennent la forme de nuages de points ou de structures particulières dans l'espace de description des observations. Selon les critères d'évaluation utilisés, les catégories trouvées peuvent être très variables. On pourra éventuellement être intéressé par une hiérarchie de catégories.
- des *règles de prédiction* (*apprentissage supervisé*). Si l'une des colonnes de la table est considérée comme une étiquette (par exemple l'*activité chimique* dans la table de la figure 1), alors une tâche peut être d'apprendre à associer une étiquette à n'importe quelle observation (décrite par les autres attributs). Si les étiquettes correspondent à des exemples positifs et négatifs, on parle d'*apprentissage de concept* car on cherche alors à trouver une description caractérisant l'appartenance à ce concept. Si les étiquettes appartiennent à un ensemble fini de possibilités, on parle de *classification*. Finalement, si les étiquettes sont prises dans $\mathbb{R}$, on parle généralement de *régression*.
- un *tri des éléments de* $\mathcal{S}$, on parle alors de *ranking*. On peut ainsi vouloir fournir une liste triée de films à destination d'un internaute client d'un site de recommandation. L'apprentissage consiste alors à apprendre à trier correctement des ensembles d'éléments.

Les types de régularités recherchées sont donc variables, fonctions des buts de l'apprentissage, par exemple : prédiction ou compréhension.

L'apprentissage dit *supervisé* prend en entrée un échantillon d'observations avec leur étiquette associée : $\mathcal{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$ et cherche une dépendance entre observation et étiquette dans le but d'être capable de prédire l'étiquette d'une observation inconnue, et éventuellement d'aider l'expert à comprendre la nature de cette dépendance. Celle-ci peut prendre la forme d'une fonction ou d'une distribution de probabilité jointe $\mathbf{p}_{\mathcal{X}\mathcal{Y}}$.

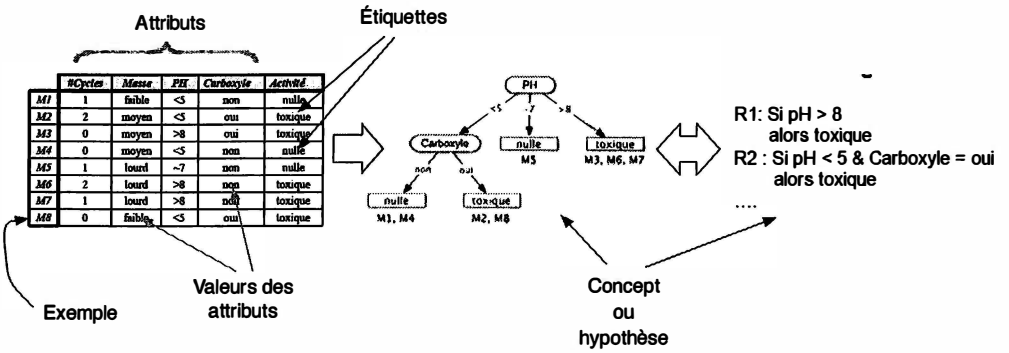


FIGURE 1 – *Extraction de régularités, ici sous la forme de règles de prédiction, à partir d'une table de données.*

De même que l'espace de description des observations prend des formes variées, depuis des espaces vectoriels jusqu'à des espaces d'objets structurés, l'*espace des étiquettes* peut être de nature diverse. Dans le cas de la *classification*, il s'agit d'un espace discret contenant un nombre fini d'éléments interprétés comme étant des classes, par exemple {molécule bioactive, molécule non bioactive}. Dans le cas de la *régression*, il s'agit le plus souvent de l'ensemble des réels $\mathbb{R}$. Mais dans le cas d'*apprentissage multivarié*, l'espace de sortie peut être le produit cartésien de plusieurs espaces. Par exemple, on pourrait vouloir prédire à la fois l'âge et le revenu d'un individu décrit par un certain nombre de caractéristiques. Certains apprentissages visent même à prédire des structures, comme celle d'une molécule ou d'un réseau de régulation par exemple. On parle alors d'apprentissage avec sortie structurée.

Étant donnée sa place centrale à ce jour dans les applications et dans l'analyse théorique de l'apprentissage, nous accordons à l'apprentissage supervisé une attention particulière dans la suite de ce chapitre. Pour les lecteurs intéressés spécifiquement par l'apprentissage non supervisé, souvent assimilé au clustering, nous renvoyons par exemple à [Xu et Wunsch, 2008].

10.1.2 Le critère inductif

L'apprentissage consiste à construire une représentation du monde en adéquation avec les observations disponibles et permettant de prendre des décisions. Cette représentation peut prendre des formes diverses : hyperplan séparateur dans l'espace des observations $\mathcal{X}$ lorsque l'on veut distinguer deux classes, distribution de probabilité, automate à états finis, description logique, etc. Pour trouver cette représentation, il faut explorer un espace de possibilités généralement énorme, l'espace des hypothèses $\mathcal{H}$. Afin d'être capable de comparer les descriptions candidates entre elles, il faut pouvoir évaluer leur mérite. C'est le rôle du critère inductif.

Ce mérite dépend de la tâche considérée. Nous prendrons ici l'exemple de **l'apprentissage supervisé**. Dans ce cas, l'objectif est de trouver une fonction de décision de l'espace des observations $\mathcal{X}$ vers l'espace des étiquettes $\mathcal{Y}$ permettant les meilleures

prédictions face aux environnements attendus. On traduit cela par une espérance de coût associée aux décisions prises dans le futur si l'on utilise l'hypothèse h. Les coûts sont définis ponctuellement pour chaque évènement $(\mathbf{x}, y)$ possible. Si l'hypothèse h est utilisée, alors la prédiction associée à l'observation $\mathbf{x}$ sera $h(\mathbf{x})$ tandis que la vraie réponse qu'il aurait fallu produire sera y. Cet écart se traduira par un coût noté $\ell(h(\mathbf{x}), y)$, calculé grâce à la *fonction de perte* $\ell(\cdot, \cdot)$. Cette fonction peut être symétrique, comme c'est le cas de la fonction qui prend la valeur 1 quand $h(\mathbf{x}) \neq y$ et 0 sinon, ou encore dans le cas d'un écart quadratique : $\ell(h(\mathbf{x}), y) = (h(\mathbf{x}) - y)^2$. Mais elle peut être non symétrique si un type d'erreur est plus important que l'autre, comme c'est, par exemple, souvent le cas en médecine. On cherche naturellement à minimiser l'espérance de perte en prenant en compte la densité de probabilité $\mathbf{p}_{\mathcal{X}\times\mathcal{Y}}$ des paires $(\mathbf{x}, y)$ qui caractérise l'environnement dans lequel évoluera le système une fois choisie l'hypothèse h.

Formellement, cela s'exprime par l'espérance de perte que l'on appelle le *risque réel* associé à la fonction de décision $h : \mathcal{X} \to \mathcal{Y}$:

$$R_{\text{Réel}}(h) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(h(\mathbf{x}), y)\, \mathbf{p}_{\mathcal{X}\times\mathcal{Y}}\, d\mathbf{x}dy$$

Malheureusement, cette densité $\mathbf{p}_{\mathcal{X}\times\mathcal{Y}}$ est inconnue *a priori*, et la seule information disponible lors de l'apprentissage est l'échantillon $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$. Une hypothèse souvent faite est que cet échantillon résulte d'un tirage aléatoire suivant la distribution $\mathbf{p}_{\mathcal{X}\times\mathcal{Y}}$. Cela revient à dire que les exemples sont tirés indépendamment les uns des autres et que l'environnement est stationnaire. En adoptant ces présupposés, il devient alors tentant de remplacer le problème de la recherche d'une hypothèse qui minimise le risque réel, inconnu, par celui de la recherche d'une hypothèse qui minimise la perte moyenne sur l'échantillon d'apprentissage, encore appelé *risque empirique* :

$$R_{\text{Emp}}(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i)$$

On remplace alors le critère du risque réel par un critère s'appuyant sur l'échantillon d'apprentissage. C'est ce que l'on appelle le *critère inductif*. Supposer que le risque empirique est indicatif du risque réel, celui qui nous importe, c'est prendre un pari sur le monde. Caractériser et quantifier ce pari a été l'objet principal des nombreuses recherches des dernières décennies dans le cadre de la théorie de l'induction et de la théorie statistique de l'apprentissage. Cette dernière prend fondamentalement appui sur des théorèmes de type centrale limite, qui nécessitent l'hypothèse de données indépendantes et identiquement distribuées.

L'analyse, par une approche statistique, de la pertinence de la minimisation du risque empirique, notablement par Vapnik et Chervonenkis [Vapnik, 1995] dans les années soixante-dix puis quatre-vingt, a ainsi montré qu'il fallait compléter le risque empirique par un terme pénalisant la sélection d'hypothèses h complexes. En effet, il est facile sinon, à l'aide d'hypothèses arbitrairement complexes, de trouver une hypothèse de risque empirique très faible, voire nul, c'est-à-dire une hypothèse se comportant très bien sur l'échantillon d'apprentissage, mais dont le risque réel est sans rapport, ce que l'on appelle « sur-apprentissage » ou sur-adaptation. Dans ce cas, l'apprentissage aura

identifié des coïncidences accidentelles dans l'échantillon d'apprentissage au dépend des régularités « vraies » du monde.

Il faut donc optimiser un compromis entre envisager un espace d'hypothèses assez riche pour pouvoir y trouver de bonnes hypothèses, et le prendre cependant assez restreint pour que le risque empirique, mesuré sur l'échantillon d'apprentissage, soit représentatif du risque réel sur les données non encore observées. L'optimisation de ce compromis peut s'opérer par plusieurs approches.

1. La *régularisation.* Elle consiste à pénaliser les hypothèses trop complexes, cette complexité étant par exemple mesurée par la régularité des fonctions de base (e.g. le degré du polynôme utilisé), et/ou par le nombre de fonctions de base impliquées (voir section 10.2.1). La méthode de *minimisation du risque structurel* proposée par Vapnik est liée à cette approche. Mais au lieu de pénaliser l'hypothèse considérée, elle pénalise la richesse de l'espace d'hypothèses dans lequel s'opère la recherche de la meilleure hypothèse. Les techniques d'élagage consistant à réduire la complexité d'une hypothèse après apprentissage entrent dans ce cadre.
2. Le *contrôle du processus d'optimisation.* L'idée est de limiter l'exploration de l'espace des hypothèses en interrompant le processus de recherche pour empêcher d'arriver à des hypothèses apparemment très bonnes mais qui au lieu de capturer seulement les régularités sous-jacentes des données, détectent aussi leurs particularités accidentelles. La *règle d'arrêt prématuré* utilisée pour les perceptrons multicouches, et qui arrête le processus d'optimisation avant sa convergence, en est un exemple.

L'étude du critère inductif dans le cas de l'apprentissage supervisé à partir de données supposées tirées indépendamment selon une distribution stationnaire est la plus avancée actuellement car elle se prête bien aux analyses de type statistique.

En revanche, l'étude formelle de l'apprentissage non supervisé ou d'autres types d'apprentissages, par exemple lorsque le monde évolue et que les données ne sont donc plus indépendantes, n'a pas la même maturité et reste encore exploratoire.

10.1.3 Le choix de l'espace des hypothèses

L'apprentissage inductif consiste à chercher une hypothèse capturant les régularités profondes du monde environnant et permettant donc des prédictions fiables. Une étape essentielle dans ce processus consiste à fixer l'espace des hypothèses $\mathcal{H}$ que l'on est prêt à considérer. C'est ce que l'on appelle la *sélection de modèles*, par référence au vocabulaire employé en statistiques. Pour cela, on s'appuie sur des connaissances préalables plus ou moins précises, des connaissances expertes, ainsi que sur des considérations liées à la facilité d'exploration de l'espace. Ainsi, par exemple, un espace d'hypothèses linéaires est très limitatif car son pouvoir expressif est réduit, mais il permet la mise en jeu de méthodes d'optimisation convexes donc efficaces. L'espace des hypothèses peut également prendre la forme d'une famille de distributions de probabilité, l'apprentissage consistant alors à estimer la valeur des paramètres fixant une distribution donnée. En *apprentissage symbolique*, l'espace des hypothèses sera défini par un langage de description (e.g. des clauses de Horn, des règles de décision, des arbres de décision, etc.) associé éventuellement à un biais de préférence pour des hypothèses « simples ».

Il existe une distinction importante entre les *modèles génératifs* et les modèles qui ne le sont pas et que l'on qualifie fréquemment de *modèles discriminatifs*. (Un ouvrage de facture très théorique focalisé sur cette distinction est [Jebara, 2003]).

1. L'apprentissage d'un modèle *génératif* vise à estimer une distribution de probabilité $\mathbf{p}_{\mathcal{X}\mathcal{Y}}$ sur l'espace conjoint des observations et des étiquettes $\mathcal{X} \times \mathcal{Y}$. Cette distribution peut être décomposée en $\mathbf{p}_{\mathcal{X}}\,\mathbf{p}_{\mathcal{Y}|\mathcal{X}}$ ou en $\mathbf{p}_{\mathcal{Y}}\,\mathbf{p}_{\mathcal{X}|\mathcal{Y}}$. Dans les deux cas, une fois les distributions apprises, il est possible de les utiliser pour générer un ensemble de points $\mathbf{z} = (\mathbf{x}, y)$ qui, si l'estimation est correcte, doit être statistiquement indistinguable de l'échantillon d'apprentissage. C'est la raison pour laquelle on parle d'apprentissage génératif. Les algorithmes utilisés ressortent des techniques statistiques d'estimation de modèles à partir de critères tels que le *maximum de vraisemblance* ou le *maximum a posteriori*. L'algorithme d'Expectation-Maximization (EM) est un exemple typique de ces algorithmes (on peut en trouver une description dans des ouvrages généraux comme [Barber, 2012 ; Bishop, 2006 ; Hastie *et al.*, 2009 ; Koller et Friedman, 2009]).
2. L'apprentissage qualifié de *discriminatif* se donne comme objectif d'associer la bonne étiquette y à toute observation $\mathbf{x}$. Pour ce faire, il n'est pas nécessaire d'estimer les distributions de probabilité sous-jacentes aux données. On peut par exemple déterminer une frontière de décision entre classes d'observations. Si cet apprentissage peut sembler moins satisfaisant puisqu'il cherche une information plus limitée sur le monde, c'est aussi ce qui fonde son avantage dans bien des situations car il demande moins de données et moins d'*a priori* sur le monde sous la forme de distributions de probabilité paramétriques qu'il faut fournir au système d'apprentissage.

Il est évident que la définition de l'espace des hypothèses contrôle l'adéquation possible au vrai modèle sous-jacent du monde, elle dicte aussi le type d'algorithmes mis en jeu. Dans la suite, nous étudions trois grandes familles d'espaces d'hypothèses correspondant à trois grandes approches algorithmiques. Deux reposent principalement sur des techniques d'optimisation dans des espaces de paramètres réels alors que le troisième met en œuvre des stratégies de recherche dans un espace structuré. Cette perspective orientée sur l'approche algorithmique nous a conduit à une organisation non classique des paradigmes d'apprentissage.

10.2 L'apprentissage selon les espaces d'hypothèses : par dictionnaire ou à partir d'exemples

Cette section regroupe deux familles d'apprentissage fondées sur des techniques d'optimisation mais qui diffèrent dans les modèles sous-jacents mis en œuvre.

1. La première famille d'apprentissage cherche à modéliser les régularités du monde à partir de *combinaisons de fonctions de base*. Dans ce cadre, une hypothèse s'exprime par une telle combinaison, et ne fait pas intervenir dans son expression les exemples d'apprentissage.

2. La seconde famille, par contraste, exprime les hypothèses en *se référant explicitement aux exemples d'apprentissage*, ou à un sous-ensemble de ceux-ci, et utilise des fonctions de similarité avec pondération éventuelle pour calculer la réponse associée à une observation.

10.2.1 Modèles définis à l'aide d'un dictionnaire de régularités

La première famille d'espaces d'hypothèses (ou famille de modèles dans le cadre statistique) concerne les hypothèses décrites comme combinaisons d'un ensemble de fonctions de base. Typiquement, il s'agit de familles de distributions de probabilité, par exemple un mélange de n gaussiennes, ou bien de combinaisons linéaires de fonctions de base :

$$h(\mathbf{x}) = \sum_{i=1}^{n} w_i \, g_i(\mathbf{x}) + w_0$$

où les $g_i(\cdot)$ sont les fonctions de base et les w_i les coefficients ou paramètres.

Parce que ces méthodes utilisent un ensemble de fonctions de base souvent *données a priori*, à l'instar des séries de Fourier, on les appelle fréquemment des méthodes à base de dictionnaire. Le nombre de fonctions de base utilisées peut servir à contrôler la capacité de l'espace d'hypothèses et donc à régulariser le risque empirique.

Ces fonctions de base $g_i(\cdot)$ permettent en un sens la redescription des entrées, et les paramètres w_i servent alors à sélectionner ou à pondérer l'importance des éléments de cette redescription. Contrairement à l'analyse de Fourier et à l'analyse fonctionnelle en général, les fonctions de base utilisées dans ces méthodes ne sont pas nécessairement orthogonales. Cela peut alors poser des problèmes d'unicité des solutions et de stabilité, les coefficients w_i pouvant varier fortement lorsque l'échantillon d'apprentissage est légèrement modifié.

Une grande partie des méthodes d'apprentissage concernent cette famille d'espaces d'hypothèses qui combinent un ensemble de fonctions de base prédéfinies. L'enjeu est alors d'apprendre les valeurs des paramètres de combinaison w_i, voire, parfois, des paramètres θ_j des fonctions de base elles-mêmes si elles sont variables : $g_j(\mathbf{x}, \theta_j)$. Sans être exhaustif, on peut citer les espaces d'hypothèses suivant :

— Les **modèles additifs** correspondent aux combinaisons les plus simples de fonctions de base : des sommes pondérées de celles-ci. Dans la *régression linéaire*, les fonctions de base sont elles-mêmes juste les coordonnées des entrées. Les *classifieurs linéaires*, tels le perceptron, sont de ce type avec une fonction signe pour décider si l'observation est positive ou non : $h(\mathbf{x}) = \text{signe}(\sum_{i=1}^{n} w_i \, g_i(\mathbf{x}) + w_0)$. Les fonctions $g_i(\mathbf{x})$ correspondent alors à l'extraction de descripteurs ou de formes dans les entrées. Les fonctions de base peuvent aussi prendre la forme de fonctions prises dans des familles plus complexes, par exemple les *splines*. De fait, si l'on dispose d'un dictionnaire de fonctions de base adapté, il est possible d'approcher n'importe quel signal ou n'importe quelle fonction cible [Hastie *et al.*, 2009], la question étant évidemment de trouver ce dictionnaire.

L'apprentissage par *boosting*, qui, dans sa version de base, apprend une combinaison linéaire de « classifieurs faibles » (faibles au sens où chaque classifieur $h_t(\cdot)$ de la combinaison peut n'être qu'à peine meilleur qu'un classifieur opérant par tirage aléatoire de la classe à prédire) : $h_T(\mathbf{x}) = \text{signe}\{\sum_{t=1}^{T} w_t\, h_t(\mathbf{x})\}$ est un bel exemple de la puissance de telles méthodes. Ici, chaque fonction de base est alors adaptative et apprise en même temps que les coefficients w_t de la combinaison [Schapire, 2003 ; Shapire et Freund, 2012 ; Zou, 2012].

— Les **modèles de mélanges**, qui font partie des approches génératives de l'apprentissage, servent à estimer la densité de probabilité $\mathbf{p}_{\mathcal{X}}$ par une formule du type : $\mathbf{p}(\mathbf{x}) = \sum_{i=1}^{n} \pi_i\, \mathbf{p}_i(\mathbf{x}|\theta_i)$. La densité de probabilité générale sur $\mathcal{X}$ est ainsi décomposée en une somme pondérée de fonctions de densité. Chacune des fonctions de base $\mathbf{p}_i(\mathbf{x}|\theta_i)$ consiste typiquement en une fonction paramétrique simple (de paramètre θ_i) comme une densité normale par exemple. Le coefficient π_i représente la probabilité qu'un point tiré au hasard ait été engendré à partir de la densité i, que l'on interprète parfois comme une classe de formes « expliquant » les observations.

— Les **perceptrons multicouche** combinent des fonctions de base (typiquement des sigmoïdes), réalisées par les « neurones », dans une structure hiérarchique fixée par l'utilisateur entre une couche d'entrée et la couche de sortie via des couches cachées (voir figure 2). L'apprentissage consiste dans ce cadre à apprendre les poids des connexions entre les neurones de couches consécutives, soit encore les coefficients v_k pris en compte dans la fonction d'activation (souvent une sigmoïde $s(a) = \frac{1}{1+\exp(-a)}$) :

$$g(\mathbf{x}) = s\left(\sum_{j=1}^{d} v_k x_k + v_0\right) = s(\mathbf{v}\cdot\mathbf{x})$$

où s est la fonction d'activation (e.g. une sigmoïde) (une référence assez complète sur ces méthodes neuronales est [Dreyfus *et al.*, 2008]).

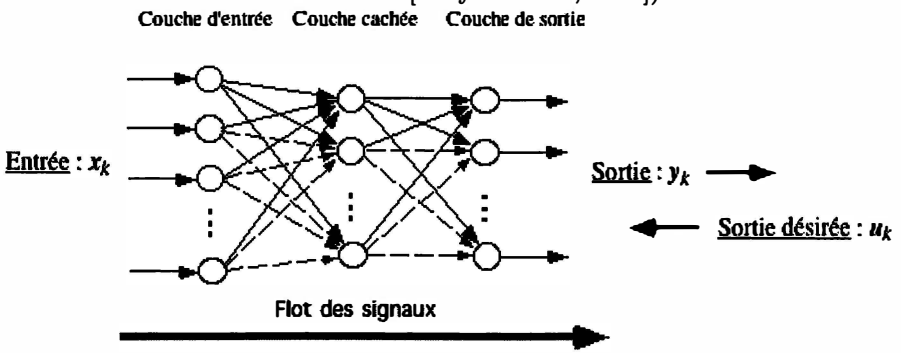


FIGURE 2 – *Schéma d'un perceptron multicouche.*

— Les **modèles graphiques à structure fixée**, dont les réseaux bayésiens ou les modèles de Markov font partie, sont des représentations de dépendances

conditionnelles entre variables. Lorsque la structure de ces dépendances est fixée (ce qui peut faire l'objet d'un apprentissage, voir section 10.3), l'apprentissage consiste à apprendre les probabilités conditionnelles en chaque nœud du réseau, ce qui revient à apprendre les paramètres des fonctions de base. (Voir [Koller et Friedman, 2009], et [Pearl, 1988] pour l'ouvrage pionnier dans ce domaine).

Le premier problème à résoudre dans l'emploi de ces méthodes concerne donc le choix du « dictionnaire » de fonctions de base. Le deuxième problème est celui de l'estimation des paramètres w_i. Plusieurs approches existent pour le résoudre. Elles dépendent du type de fonction de perte utilisée dans la définition du risque empirique et de l'ensemble des fonctions de base. Trois grandes familles d'approches peuvent être distinguées :

1. *Méthodes à base de gradient* [Snyman, 2005]. Ces méthodes partent d'une estimation initiale du vecteur de paramètres $\mathbf{w} = (w_0, w_1, \ldots, w_n)^\top$ puis l'adaptent par une méthode de gradient. Lorsque celui-ci est calculé sur le risque empirique (régularisé), on parle de gradient total. Lorsqu'il est calculé après chaque présentation d'une donnée $\mathbf{z}_t$, on parle de gradient stochastique. De nombreuses variantes existent. Le célèbre algorithme de rétro-propagation de gradient est ainsi une procédure permettant de calculer le gradient de l'erreur en chaque neurone et chaque connexion d'un perceptron multicouche.
2. *Méthodes itératives* [Kelley, 1987]. Les méthodes itératives cherchent également à diminuer la mesure de risque de manière itérative. Cependant, elles ne s'appuient pas sur une estimation de gradient, mais utilisent des fonctions d'approximation ou des fonctions de perte spéciales permettant d'assurer qu'un procédé itératif diminue finalement le risque. Les méthodes du type expectation-maximization (EM), utilisées dans le cas de données non supervisées, sont un exemple d'une telle approche.
3. *Optimisation gloutonne* [Cormen *et al.*, 2001]. Les méthodes d'optimisation gloutonnes décomposent le problème d'optimisation en sous-problèmes plus simples. Dans le cas des hypothèses définies par combinaison de fonctions de base, ce genre de méthode consiste à optimiser l'un des termes de la combinaison (e.g. l'un des termes de la combinaison linéaire), puis à le fixer et optimiser un autre terme jusqu'à ce que tous les termes soient optimisés. Un bon optimum n'étant pas nécessairement atteint après une seule passe, on peut éventuellement répéter plusieurs fois ce cycle d'optimisation. Le *boosting* qui opère par une sorte de descente de gradient conjugué est un exemple de cette famille d'approches.

En dehors de la détermination de la méthode d'optimisation la plus adaptée au problème, le principal problème de ces méthodes à partir de dictionnaire est de déterminer le bon dictionnaire de fonctions de base, c'est-à-dire, *in fine*, le bon espace $\mathcal{H}$ de fonctions hypothèses. C'est encore une fois le problème de la sélection de modèles dont l'objectif est en particulier de contrôler le risque de sur-apprentissage c'est-à-dire de sur-adaptation aux données au détriment de l'estimation des « vraies » régularités du monde (voir la fin de la section 10.1.2).

Il faut noter la *place particulière qu'occupent les modèles linéaires* dans les modèles à partir de dictionnaire de fonctions de base. Si ils peuvent paraître limités dans leur

pouvoir expressif, ils présentent cependant de nombreux avantages qui les font toujours apprécier à la fois des théoriciens et des praticiens.

D'abord, leur pouvoir expressif dépend en fait du pouvoir expressif des fonctions de base. Si celles-ci sont adaptées au problème, ces modèles peuvent approcher n'importe quelle régularité cible. Surtout, si les fonctions de base sont suffisamment orthogonales entre elles, les coefficients w_i attachés à chaque fonction de base permettent d'en évaluer l'importance dans le signal étudié. Ces modèles se prêtent ainsi plus aisément à une interprétation par l'utilisateur ou l'expert que des modèles plus complexes, non linéaires par exemple. Finalement, il est possible à la fois d'améliorer encore cette interprétabilité et de contrôler la richesse de l'espace des hypothèses, si cruciale pour contrôler le risque de sur-apprentissage, en cherchant à diminuer le nombre de coefficients w_i qui sont non nuls. L'idée est alors de pénaliser les hypothèses en fonction du nombre de fonctions de base mises en jeu. Ces approches sont actuellement très étudiées (voir, par exemple, les méthodes LASSO [Hastie *et al.*, 2009]).

Cependant, contraindre l'expression du modèle du monde à être linéaire peut aussi conduire à un modèle artificiel qui ne rend pas compte de la structure des dépendances entre propriétés du monde. L'engouement actuel pour ces modèles est donc à tempérer comme le font certaines études critiques (voir [Bengio et Cun, 2007]).

10.2.2 Modèles définis à partir des exemples d'apprentissage

Si apprendre c'est souvent généraliser pour mieux oublier les exemples et les représenter sous la forme d'un modèle, on peut aussi construire des modèles qui conservent explicitement les exemples d'apprentissage. C'est le cas de la méthode élémentaire qui consiste à étiqueter une nouvelle observation en fonction de l'étiquette de ses plus proches voisins. C'est plus généralement le principe mis en œuvre dans le raisonnement à partir de cas (cf. chapitre I.7) où les cas sont mémorisés puis réutilisés, et éventuellement adaptés, selon les besoins. Il est bien connu que ces approches ne sont efficaces qu'à proportion qu'un échantillon suffisant et suffisamment représentatif de cas soit connu et que la fonction de distance, ou plus généralement de similarité, soit adéquate. C'est justement l'objet de l'apprentissage d'essayer d'obtenir ces conditions à partir d'un échantillon d'exemples pour apprendre une bonne fonction de décision.

Si l'on se place dans le cadre des modèles linéaires, les méthodes à base d'exemples par voisinage utilisent une représentation des fonctions de décision de la forme :

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \, \kappa(\mathbf{x}, \mathbf{x}_i) \, y_i$$

La différence avec les méthodes à base de dictionnaires est évidente. On voit en effet que la fonction hypothèse est définie ici directement à l'aide des points d'apprentissage $\mathbf{x}_i$, d'une part, par l'intervention de la fonction de similarité $\kappa(\mathbf{x}, \mathbf{x}_i)$, et, d'autre part, par les étiquettes des points d'apprentissage y_i. *L'hypothèse est maintenant une combinaison linéaire des étiquettes des points d'apprentissage*, pondérées par les similarités, là où elle était précédemment une combinaison linéaire de fonctions de base.

L'apprentissage consiste donc désormais à sélectionner les bons exemples de référence $\mathbf{x}_i$. Mais avant cela, il faut se préoccuper du choix de la fonction de similarité $\kappa(\mathbf{x}, \mathbf{x}')$.

Mesures de similarité et exemples de référence

Un problème essentiel des méthodes à base d'exemples est celui du choix de la mesure de similarité, voire de distance, appropriée. Ainsi, tout joueur d'échec sait très bien que deux situations de jeu exactement identiques à l'exception de la position d'un pion, peuvent en fait être totalement différentes pour l'issue du jeu. Ce qui est vrai pour le jeu d'échec l'est également lorsque l'on compare des objets variés comme des molécules, des textes, des images, etc. À chaque fois, le choix de la bonne mesure de similarité est crucial pour qu'une méthode basée sur des similarités à des exemples connus soit performante.

En dehors même de la bonne adéquation de la fonction de similarité considérée à la sémantique du domaine se pose le problème de sa définition formelle. Si les mathématiques nous fournissent de nombreuses mesures adaptées aux données vectorielles numériques, par exemple sous la forme de distances, le problème est bien plus ouvert lorsque les données font intervenir des descripteurs symboliques et/ou sont définies dans des espaces non vectoriels, par exemple des textes. C'est pourquoi une partie importante des contributions actuelles en apprentissage artificiel concerne la définition et le test de nouvelles mesures de similarité appropriées pour des types de données particuliers : séquences, fichiers en format XML, données structurées, etc.

Une *distance* est une fonction symétrique de deux arguments, nulle seulement lorsque les deux arguments sont égaux et obéissant à l'inégalité triangulaire. Lorsque l'une des propriétés n'est pas vérifiée, on parle souvent plus librement de *dissimilarité*. Un exemple de distance est celui de la norme euclidienne entre deux vecteurs $||\mathbf{x} - \mathbf{y}|| = \sqrt{\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle}$ dans lequel $\langle \mathbf{x}, \mathbf{x}' \rangle$ est le produit scalaire entre les vecteurs $\mathbf{x}$ et $\mathbf{x}'$. Ce produit scalaire peut servir de base à des mesures de similarité comme le cosinus entre deux vecteurs. Plus les vecteurs sont « alignés », plus ils sont proches selon cette similarité.

Il se trouve que si l'on cherche un séparateur linéaire entre deux nuages de points qui maximise la marge entre ces nuages (en supposant qu'un tel séparateur existe), on peut exprimer ce séparateur comme une fonction de points particuliers $\mathbf{x}_i$ dans l'échantillon d'apprentissage appelés par Vapnik *vecteurs de support* :

$$h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle y_i$$

Il s'agit là d'une découverte importante car elle signifie que la solution qui optimise le risque empirique régularisé (au sens où on cherche une marge maximale et non n'importe quel séparateur linéaire) prend la forme d'une combinaison linéaire définie à partir de certains exemples d'apprentissage.

Cependant, cette observation, si intéressante soit-elle, n'aurait pas eu un grand impact si il n'avait pas été réalisé, encore une fois par Vapnik et ses collègues, que

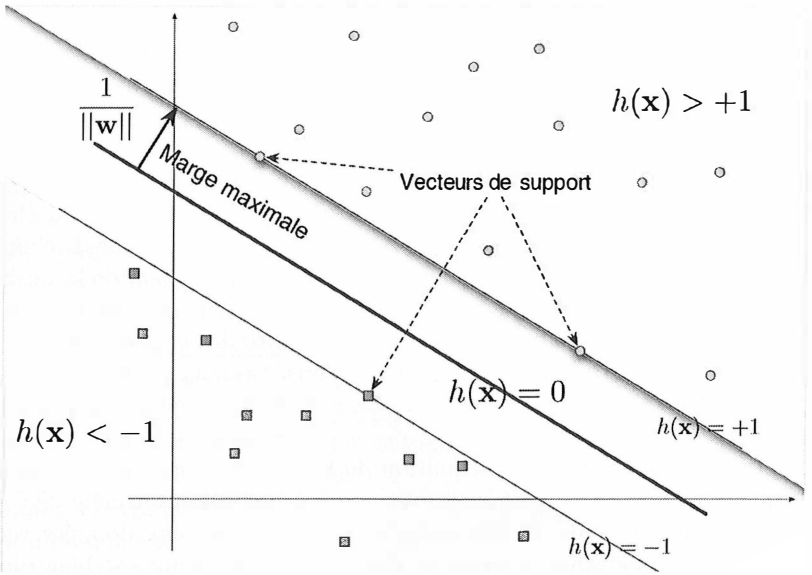


FIGURE 3 – *L'hyperplan optimal maximise la distance aux points d'apprentissage des deux classes. Cette distance est entièrement déterminée par les vecteurs de support.*

l'on pouvait généraliser ce type de solution à des problèmes de séparation non linéaire en remplaçant le produit scalaire $\langle \mathbf{x}, \mathbf{x}' \rangle$ par une fonction vérifiant certaines propriétés particulières que l'on appelle fonction noyau : $\kappa(\mathbf{x}, \mathbf{x}')$.

On peut alors montrer qu'un problème de séparation *non linéaire* entre deux classes d'objets dans l'espace d'entrée $\mathcal{X}$ peut s'exprimer sous la f*orme d'une combinaison linéaire* : $h(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \, \kappa(\mathbf{x}, \mathbf{x}_i) \, y_i$ où $\kappa(\mathbf{x}, \mathbf{x}')$ est une fonction noyau et les $\mathbf{x}_i$ sont des exemples d'apprentissage particuliers permettant de définir la séparatrice entre les nuages de points.

L'algorithme des *Séparateurs à Vastes Marges* (SVM ou « *Support Vector Machines* » en anglais) trouve les points supports et les coefficients α_i optimaux. Ils sont solution d'un problème d'optimisation quadratique, c'est-à-dire d'un problème convexe admettant un optimum unique, ce qui est un énorme avantage par rapport à d'autres méthodes, comme par exemple les perceptrons multicouches. Sans entrer dans les détails, la complexité de cet algorithme est dominée par le nombre d'exemples d'apprentissage, et non par la dimension de l'espace considéré, comme c'est généralement le cas pour les autres approches. Afin de pouvoir traiter des données en grand nombre et des données complexes, on cherche à rendre plus efficace le processus d'optimisation, par exemple en remplaçant le problème d'optimisation par un autre, plus facile à calculer, mais de même solution.

Cependant, la qualité de la fonction apprise repose fondamentalement sur le choix de la fonction noyau utilisée. Sans surprise, on peut montrer ici encore que ce choix est profondément lié au problème de la sélection de modèles et donc au contrôle du sur-apprentissage. De nombreux travaux de recherche portent sur la sélection automatique

de fonctions noyau. Cependant, celle-ci est encore largement du ressort de l'expert, particulièrement lorsque les données à comparer sont complexes, comme des molécules, des documents ou des objets structurés en général.

Il est intéressant de noter que le développement des SVM, initié pour le problème de la classification en deux classes, a conduit à la réalisation que de nombreuses autres méthodes linéaires, telles l'analyse en composantes principales ou les filtres de Kalman, pouvaient aisément se généraliser au cas non linéaire. En fait, à chaque fois qu'une méthode conduit à des solutions mettant en jeu des produits scalaires $\langle \mathbf{x}, \mathbf{x}' \rangle$, elle peut se généraliser au cas non linéaire en remplaçant ce produit scalaire par une fonction noyau $\kappa(\mathbf{x}, \mathbf{x}')$. Cette observation a stimulé un vaste mouvement de réutilisation et de développement de nombreuses méthodes linéaires dans le cadre non linéaire grâce à ce que l'on appelle les *méthodes à noyaux* (voir [Schölkhopf et Smola, 2002 ; Shawe-Taylor et Cristianini, 2004]).

Points de référence

Nous venons de voir que, dans le cas de la classification, et plus précisément de la recherche d'une frontière entre classes, les points supports jouaient un rôle capital. Il se trouve que ces points supports, calculés dans les algorithmes de type SVM, sont les plus proches de la frontière cherchée. Ce n'est pas surprenant car, en un certain sens, ce sont eux qui contraignent cette frontière et permettent de déterminer la « marge » entre les nuages de points. Il serait cependant possible d'imaginer que d'autres points jouent un rôle essentiel. Ainsi, dans le cas de l'apprentissage non supervisé, et spécialement du clustering, les points de référence sont généralement au contraire des points qui représentent le « centre de gravité » des classes. Ces points, correspondant à des points d'apprentissage ou bien calculés, jouent alors le rôle de prototypes.

10.3 Espace des hypothèses : modèles à structure variable

Dans le cas des méthodes à base de dictionnaire, le choix du bon espace d'hypothèses, c'est-à-dire du bon dictionnaire se traduit par le problème dit de la sélection de modèles. Cette sélection peut se faire à la main, par l'expert, ou par des méthodes plus systématiques, mais l'espace des hypothèses $\mathcal{H}$ est donné *a priori* : celui de toutes les fonctions de base potentielles, de la même manière que l'on peut chercher la meilleure base de fonctions dans une analyse en ondelettes. Les méthodes à base d'exemples introduisent une souplesse supplémentaire dans la mesure où l'expression des hypothèses candidates dépend désormais directement des exemples d'apprentissage. L'adaptation au problème se fait ainsi plus naturellement, au prix cependant du bon choix de la fonction de similarité ou de la fonction noyau permettant la comparaison entre les exemples. Dans tous les cas, l'apprentissage consiste essentiellement à explorer un espace d'hypothèses que l'on peut qualifier de paramétré, d'où des techniques spécifiques de recherche. De plus, ces deux familles de méthodes sont souvent utilisées dans le cadre de modèles linéaires, ce qui autorise des algorithmes efficaces mais limite aussi la possibilité de rendre compte de régularités complexes, du moins de manière intelligible.

Il est heureusement possible d'aller plus loin dans l'adaptation de l'espace des hypothèses au problème d'apprentissage étudié. C'est ce que permettent les modèles à structure variable. Dans ces modèles, la forme des hypothèses n'est plus fixée *a priori* comme, par exemple, une combinaison linéaire de fonctions ou de sorties pondérées par des ressemblances aux points d'apprentissage, mais elle est variable. Il peut exister des *hiérarchies de descripteurs* ou de *concepts intermédiaires* ou encore des expressions relationnelles arbitrairement complexes. C'est le cas des réseaux bayésiens ou modèles graphiques, des modèles par arbre de décisions, ou encore celui des modèles à base de règles.

Dans ces approches, l'apprentissage comporte naturellement deux aspects. D'une part, il doit permettre de *découvrir une bonne structure* pour représenter les hypothèses, d'autre part, il doit *estimer les valeurs optimales des paramètres* éventuels.

La recherche dans ce cas de la meilleure approximation des régularités cibles inconnues est encore plus difficile et peut paraître désespérée. Le nombre de structures possibles est en effet généralement gigantesque, chacune possédant ses propres paramètres. Comme on ne dispose plus dans cet espace de propriétés de continuité ou d'analycité comme dans l'espace des modèles paramétrés, une exploration bien guidée paraît impossible. De surcroît, toujours afin de contrôler le risque de sur-apprentissage, il faut pouvoir estimer la qualité des espaces d'hypothèses considérés dans le processus constructif et cela à chaque pas. Les difficultés sont donc magnifiées par rapport aux apprentissages décrits précédemment.

Il n'existe pas actuellement de méthodes génériques s'appliquant à tous les espaces de structures. Chaque famille d'espaces a ses spécificités et des approches particulières qui font encore souvent l'objet de recherches. Ainsi, dans les réseaux bayésiens, il existe des approches fondées sur les contraintes tirant profit du test d'indépendance entre variables. D'autres approches sont fondées sur des fonctions de score prenant en compte à la fois l'adéquation aux données, mais aussi une mesure de complexité du modèle candidat et des techniques d'exploration de graphe très variées. Nous allons voir cependant que dans certains cas, essentiellement l'apprentissage de concept utilisant un langage d'expression symbolique des hypothèses, il est possible d'exploiter certaines structures dans l'espace des hypothèses qui rendent l'apprentissage efficace.

Avant d'aller plus loin et devant la diversité des modèles, précisons quelques notions de base sur les différents langages de représentation utilisés dans la suite de cette section. Considérons d'abord l'apprentissage de règles.

Une règle est constituée d'une prémisse spécifiant les conditions d'application de la règle et d'une conclusion spécifiant ce qui est vrai lorsque la prémisse est réalisée. Plusieurs langages peuvent être utilisés pour exprimer les règles : la logique des propositions, composée uniquement de symboles de proposition, comme par exemple dans la règle `rouge` $\wedge$ `bruyant` $\wedge$ `cher` $\rightarrow$ `voiture_bling_bling`, la représentation attribut-valeur comme dans la règle `température` $\geq$ `37,5` $\rightarrow$ `fièvre` ou encore la logique du 1er ordre plus riche et permettant d'exprimer des relations plus complexes entre les objets, comme par exemple `père(X,Y), père(Y,Z)` $\rightarrow$ `grand_père(X,Z)`. Parfois, lorsque le concept à apprendre est implicite (par exemple, apprentissage du concept `voiture_bling_bling`), on omet la conclusion de la règle et on s'intéresse uniquement à la conjonction de propriétés définissant le concept. Notons enfin que des expressions

plus complexes peuvent aussi être recherchées, comme des clauses générales permettant de spécifier des alternatives dans la conclusion ou d'introduire de la négation. L'apprentissage de connaissances exprimées en logique du 1er ordre est le domaine d'étude de la *Programmation Logique Inductive* [Raedt, 2008 ; Dzeroski et Lavrac, 2001], qui a connu un grand essor dans les années 80. Une autre famille concerne les *modèles graphiques* [Koller et Friedman, 2009] : ce sont des graphes dont les nœuds sont les variables et permettant de représenter des dépendances entre variables. On distingue principalement les réseaux bayésiens, graphes orientés associant à chaque nœud la probabilité conditionnelle de ce nœud, étant donnés ses parents et les modèles de Markov, graphes non orientés pour lesquelles une variable est indépendante des autres, étant donnés ses voisins. Un courant actuel, dénommé *Apprentissage Relationnel Statistique* [Raedt *et al.*, 2008 ; Getoor et Taskar, 2007] tend à coupler la Programmation Logique Inductive et les approches probabilistes. Enfin, l'*inférence grammaticale* ([de la Higuera, 2010], [Cornuéjols et Miclet, 2010] ch.7, ou encore [Miclet, 1990]) s'intéresse à l'apprentissage de grammaires ou de langages à partir de données ; la famille de modèles considérée est souvent celle des automates, éventuellement probabilistes. Nous supposons connues ici les notions d'automates et de grammaires.

Notons que souvent on ne cherche pas une hypothèse mais un ensemble, c'est-à-dire une disjonction d'hypothèses. Considérons ainsi l'apprentissage d'un concept à partir d'exemples positifs et négatifs. Il peut être irréaliste de chercher une unique règle couvrant les exemples positifs et rejetant les négatifs, puisque cela supposerait que tous les exemples positifs suivent le même modèle : on recherche alors plutôt un ensemble de règles. Cependant, étendre le modèle à un ensemble d'hypothèses introduit à nouveau le compromis nécessaire au sein du critère inductif touchant à la complexité de l'espace des hypothèses. En effet, la disjonction des exemples positifs, $\mathbf{x}_1 \vee \ldots \vee \mathbf{x}_m$, représente un ensemble d'hypothèses qui couvre tous les exemples positifs et rejette tous les négatifs, dès lors que ces derniers diffèrent des positifs. On a alors une hypothèse arbitrairement complexe, variant au gré des exemples d'apprentissage, et d'erreur empirique nulle. Néanmoins, cet apprentissage par cœur ne correspond à aucune généralisation. Pour pallier ce problème, il faut introduire ici aussi des contraintes sur les hypothèses, c'est-à-dire faire de la régularisation comme évoqué en section 10.1.2.

10.3.1 Espace des hypothèses - relation de généralité et opérateurs

Comment, lorsque l'on ne considère plus un espace paramétré, effectuer une exploration informée de l'espace des hypothèses ? Une possibilité très intéressante existe dans le cas de l'apprentissage de concept, c'est-à-dire de la description d'une classe d'observations (contre toutes les autres). Dans ce cas en effet, on recherche une hypothèse correspondant à une partie de l'espace des observations $\mathcal{X}$ qui contient les exemples positifs (et exclut les exemples négatifs si ceux-là sont disponibles). Deux hypothèses peuvent alors être comparées en fonction des parties de l'espace $\mathcal{X}$ qu'elles décrivent respectivement, ou encore en fonction des exemples qu'elles *couvrent*. On définit la *couverture d'une hypothèse* comme l'ensemble des observations de $\mathcal{X}$ que cette hypothèse couvre. Une hypothèse est plus générale qu'une autre si sa couverture contient

la couverture de la seconde.

La relation d'inclusion définie sur $\mathcal{X}$ induit ainsi une relation de généralité sur $\mathcal{H}$ qui est un préordre partiel. La figure 4 illustre cette notion. On peut noter que ce préordre peut être transformé en une relation d'ordre en considérant que deux hypothèses sont équivalentes si et seulement si l'une est plus générale que l'autre et vice versa et en considérant l'ensemble quotient de $\mathcal{H}$ par rapport à cette relation d'équivalence. Cela revient à ne considérer qu'un représentant parmi les hypothèses équivalentes. Cette relation est partielle et non totale, ce qui signifie que deux éléments quelconques dans l'espace considéré peuvent ne pas être liés par cette relation.

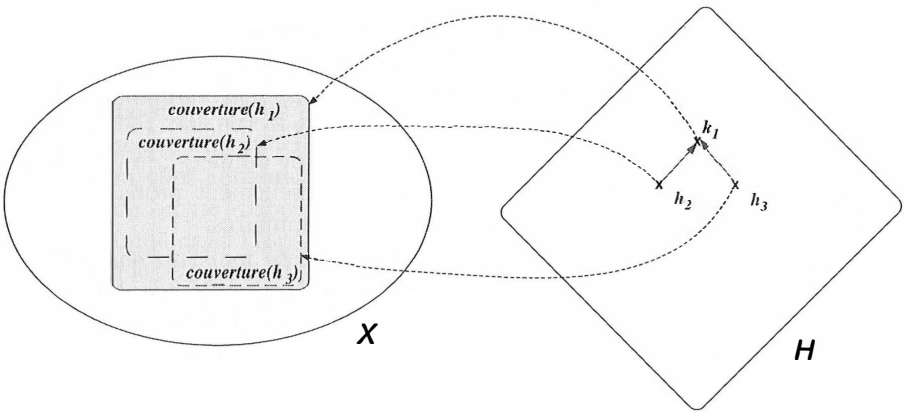


FIGURE 4 – *La relation d'inclusion dans $\mathcal{X}$ induit une relation de généralisation dans $\mathcal{H}$. Il s'agit d'une relation de préordre partiel : ici, les hypothèses h_2 et h_3 sont incomparables entre elles, mais elles sont toutes les deux plus spécifiques que h_1.*

La relation de couverture est fondamentale pour le problème de l'induction. En effet, une hypothèse incorrecte (donc couvrant indûment des exemples négatifs) devra être spécialisée pour que sa couverture exclut ces exemples, alors qu'une hypothèse incomplète (ne couvrant pas tous les exemples positifs connus) devra être généralisée pour que ces exemples deviennent éléments de sa couverture. Il est donc naturel que le processus d'induction soit guidé par la couverture des hypothèses et la relation d'inclusion éventuelle entre ces couvertures.

Nous avons parlé de la relation d'inclusion dans l'espace des observations $\mathcal{X}$, alors que l'apprentissage s'effectue par une exploration de l'espace des hypothèses $\mathcal{H}$. Il faut donc, d'une part, définir une relation de généralité permettant de structurer l'espace des hypothèses et modélisant la relation d'inclusion dans $\mathcal{X}$ et, d'autre part, définir des opérateurs de changement d'hypothèses qui respectent la relation de généralité définie dans $\mathcal{H}$ ou la relation d'inclusion dans $\mathcal{X}$.

Relations de généralité

La définition de la relation de généralité et le test de couverture sont étroitement liés. Si la relation de généralité définie sur l'espace des hypothèses respecte la rela-

tion d'inclusion sur $\mathcal{X}$, alors la recherche peut être guidée par le test de couverture, et des critères quantitatifs, comme le nombre d'exemples positifs ou négatifs couverts, satisfaisant des propriétés de monotonie, peuvent être définis afin de guider et d'élaguer la recherche. Cependant, définir une relation de généralité dépend du langage de représentation des hypothèses et des exemples. Si une telle relation est évidente en logique des propositions, elle devient plus problématique en logique des prédicats. En effet, en logique du 1er ordre, une définition naturelle serait l'implication logique entre deux formules, mais ce problème est non décidable et c'est la raison pour laquelle a été introduite par Plotkin [Plotkin, 1970] la notion de θ-subsomption entre clauses : soient deux clauses A et B, A est plus générale que B s'il existe une susbtitution θ telle que $A.\theta \subseteq B$ [2]. Par exemple, la clause `père(X,Y), père(Y,Z) → grand_père(X,Z)`est plus générale que la clause `père(jean, paul), père(paul,marie), mère(anne,marie) → grand_père(jean,marie)`. En effet ces deux clauses sont respectivement réécrites en `¬ père(X,Y) ∨ ¬ père(Y,Z) ∨ grand_père(X,Z)`et `¬ père(jean,paul) ∨ ¬ père-(paul,marie) ∨ ¬ mère(anne,marie) ∨ grand_père(jean,marie)`. Si l'on considère ces clauses comme des ensembles de littéraux et si l'on instancie X par *jean*, Y par *paul* et Z par *marie*, la première ainsi instanciée est bien incluse dans la seconde. Cette relation de généralité permet de définir des opérateurs de généralisation permettant de transformer une clause en une clause plus générale, comme supprimer un littéral, transformer une constante en variable ou transformer deux occurrences d'une même variable en des variables différentes. Cette définition ne tient pas compte des connaissances que l'on peut avoir sur le domaine, comme par exemple qu'un père ou une mère sont des parents. Mais même cette définion est problématique car elle implique des comparaisons éventuellement coûteuses entre hypothèses. Il est parfois possible d'inclure l'espace des exemples dans celui des hypothèses, en faisant d'un exemple une sorte d'hypothèse atomique. On parle alors de *single representation trick* ou d'astuce de la représentation unique. Cela simplifie en général le test de couverture. Par exemple, une hypothèse pourra être une clause et un exemple pourra être une clause complètement instanciée, comme dans l'illustration donnée ci-dessus. Il est important de réaliser que, même dans ce cas, la comparaison entre une hypothèse et un exemple, afin de savoir si l'hypothèse couvre l'exemple, peut également être source de complexité algorithmique. La complexité du *test de couverture* est souvent un élément déterminant dans le choix du codage des données et du langage des hypothèses. Par exemple, dans le cas de l'apprentissage d'un ensemble de clauses de Horn (équivalent à un programme Prolog), le test de couverture peut impliquer un démonstrateur de théorèmes ou, à tout le moins, un programme de satisfaction de contraintes.

Il arrive que les données résultent de mesures imprécises ou approximatives, et, en général, il est possible que la précision sur les valeurs des attributs de description soit illusoire. Dans ce cas, il peut être intéressant de changer de représentation pour rendre compte d'une précision plus réduite et permettre éventuellement d'obtenir des descriptions de concept plus compréhensibles. Le formalisme des *rough sets* [Suraj, 2004] offre un outil pour le raisonnement approximatif applicable en particulier pour la sélection d'attributs informatifs, la catégorisation, et la recherche de règles de classification ou de

2. Une clause est une disjonction de littéraux, assimilée dans cette définition à un ensemble de littéraux.

décision. Sans entrer dans les détails, le formalisme des rough sets permet de chercher une redescription de l'espace des exemples tenant compte de relations d'équivalence induites par les descripteurs sur les exemples disponibles (voir chapitre I.3). Puis, étant donnée cette nouvelle granularité de description, les concepts peuvent être décrits en termes d'approximation inférieure et supérieure (voir figure 5). Cela conduit à de nouvelles définitions des notions de couverture d'un exemple par un concept et de relation de généralité entre concepts.

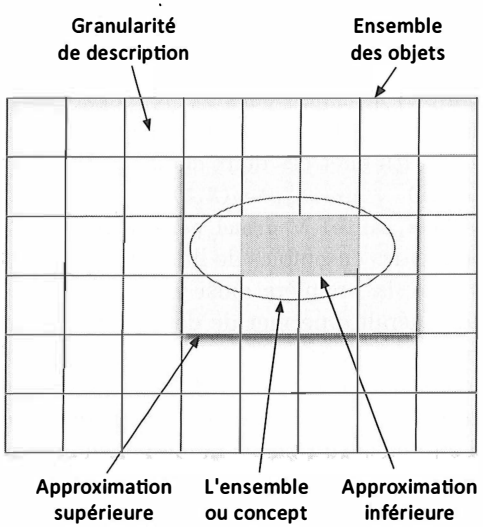


FIGURE 5 – *Les rough sets : changement de granularité de description et redéfinition d'un concept par une approximation inférieure et une approximation supérieure.*

On voit donc l'*importance du langage de représentation des hypothèses* dans la détermination d'une relation de préordre permettant l'exploration efficace de l'espace des hypothèses et dans la définition des opérateurs de changement d'hypothèses (ou de motifs).

Parmi les relations d'ordre possibles, on privilégie celles qui induisent une structure de *treillis* sur $\mathcal{H}$. Cela signifie que pour tout couple d'hypothèses h_i et h_j, il existe au moins une hypothèse qui soit plus générale que chacune d'entre elles et qu'il n'est pas possible de la spécifier sans perdre cette propriété. L'ensemble de ces hypothèses est appelé le *généralisé maximalement spécifique* de h_i et h_j et noté $gms(h_i, h_j)$. De même, il existe un ensemble d'hypothèses plus spécifiques que h_i et h_j qu'il n'est pas possible de généraliser sans perdre cette propriété. On appelle cet ensemble le *spécialisé maximalement général* et on le note $smg(h_i, h_j)$.

Par une extension facile au cas de plus de deux hypothèses, on peut définir de même un ensemble $gms(h_i, h_j, h_k, \ldots)$ et un ensemble $smg(h_i, h_j, h_k, \ldots)$.

Finalement, nous supposons[3] qu'il existe dans $\mathcal{H}$ une hypothèse plus générale que toutes les autres (ou élément maximal) notée $\top$ et une hypothèse plus spécifique que

3. C'est en général le cas en pratique.

toutes les autres (ou élément minimal) notée $\perp$ (voir la figure 6).

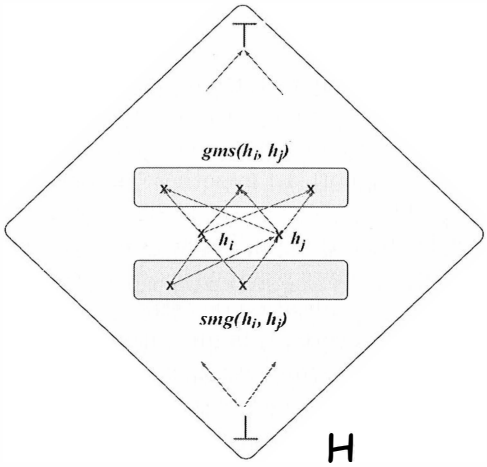


FIGURE 6 – *Une vision schématique et partielle du treillis de généralisation sur $\mathcal{H}$ induit par la relation d'inclusion dans $\mathcal{X}$. Chaque flèche indique la relation de généralité (notée $\preceq$ dans le texte). L'élément le plus spécifique du treillis est $\perp$ et le plus général est $\top$.*

Un domaine dans lequel l'apprentissage repose sur la relation de généralité dans l'espace des hypothèses est celui de l'induction de langages ou de grammaires à partir d'exemples, et éventuellement de contre-exemples, de séquences. Dans le cas de ce que l'on appelle l'*inférence grammaticale*, la description des hypothèses prend généralement la forme d'automates. La plupart des travaux ont porté sur l'induction de langages réguliers qui correspondent aux automates à états finis. Il se trouve que l'utilisation de cette représentation conduit naturellement à une opération associée à une relation de généralité entre automates. Sans entrer dans les détails formels, si l'on considère un automate à états finis et que l'on fusionne de manière appropriée deux états de cet automate, on obtient un nouvel automate qui accepte au moins toutes les séquences acceptées par le premier automate. Cet automate dérivé est donc plus général. Utilisant cette opération de généralisation, la plupart des méthodes d'inférence grammaticale partent ainsi d'un automate particulier, acceptant exactement les séquences positives, et le généralisent, par une succession de fusions d'états, en s'arrêtant soit quand une séquence négative est couverte soit quand un critère d'arrêt sur l'automate courant est vérifié. L'ouvrage de Colin de la Higuera [de la Higuera, 2010] décrit très complètement toutes les techniques de l'inférence grammaticale.

Opérateurs de généralisation/spécialisation

Une fois définie une relation de généralité sur l'espace des hypothèses, il faut définir des opérateurs qui permettent de parcourir cet espace des hypothèses. Ainsi, par exemple, supposons que l'espace des hypothèses soit celui de conjonctions en logique

des propositions. Une description du concept de `voiture bling-bling` pourra ainsi être : `rouge` $\wedge$ `bruyant` $\wedge$ `cher`. Par ailleurs, on supposera que les opérateurs de changement d'hypothèse sont l'*ajout* ou le *retrait* d'un terme de la conjonction. On pourrait ainsi produire la description `rouge` $\wedge$ `bruyant` à partir de la description précédente. Il est évident que les véhicules vérifiant cette description incluent ceux qui vérifient la première description, elle est donc plus générale. On peut montrer que l'opérateur de retrait d'un terme de la conjonction dans ce langage est associé à un gain en généralité de l'hypothèse produite par rapport à l'hypothèse de départ. Cette association entre opérateur de changement d'hypothèse et degré de généralité peut alors être utilisée pour guider efficacement l'exploration de l'espace des hypothèses. En effet, si une hypothèse candidate courante ne couvre pas tous les exemples positifs, il faudra envisager les opérateurs associés à une généralisation de cette hypothèse, tandis, qu'inversement, si des exemples négatifs sont couverts, il faudra utiliser des opérateurs associés à une spécialisation. Cela conduit à des *stratégies* d'exploration de l'espace des hypothèses, *descendante* (par spécialisation), *ascendante* (par généralisation) ou mixte.

Les opérateurs de généralisation/spécialisation dépendent du langage de description des hypothèses. Si c'est assez simple en logique des propositions (e.g. pour spécialiser, ajouter une condition à la règle), cela devient plus compliqué si l'on suppose que l'on dispose de connaissances sur le domaine permettant par exemple de spécialiser une condition, ou dans des langages plus complexes comme la logique du 1er ordre. Un opérateur pour être intéressant doit vérifier des propriétés comme être localement fini, i.e. engendrer un nombre fini et calculable de raffinements, ou encore être complet, i.e. si deux hypothèses sont comparables du point de vue de leur généralité, on peut passer de l'une à l'autre par l'application d'un nombre fini de raffinements, etc.

Exploration de l'espace des hypothèses

Pour apprendre un concept à partir d'exemples positifs et négatifs, la méthode la plus courante consiste à apprendre une hypothèse couvrant des exemples positifs puis itérer le processus sur les exemples restants. D'autres méthodes sont envisageables comme séparer les exemples positifs en classes (classification non supervisée) et apprendre une règle pour chaque classe.

Pour construire une règle couvrant des exemples positifs, plusieurs stratégies sont utilisées : une approche gloutonne, qui construit la règle itérativement ajoutant des conditions dans le corps de la règle de manière heuristique (comme par exemple dans le système Foil) ou une approche dirigée par les données qui à partir d'un exemple positif construit une règle couvrant cet exemple (comme dans le système Progol).

Des approches récentes proposent de rechercher de manière exhaustive toutes les règles avec un support et une confiance suffisants (on retrouve alors la problématique de recherche de règles d'association) et de construire un classifieur à partir de ces règles : par exemple, chaque règle applicable vote pour la classe correspondante avec un poids fonction de sa confiance.

Analyse formelle de concepts

Comme nous l'avons vu, il existe une dualité entre l'espace des observations et l'espace des hypothèses. Cette dualité est particulièrement mise en évidence par la notion de connexion de Galois mise en œuvre dans l'Analyse Formelle de Concepts [Ganter *et al.*, 1998, 2005]. Etant donnés un ensemble d'objets $\mathcal{O}$, un ensemble de descripteurs propositionnels $\mathcal{A}$ et une relation exprimant que tel objet satisfait tel descripteur, on construit deux opérateurs. Le premier, noté f, correspond à la notion de couverture : étant donné un ensemble de descripteurs A, $f(A)$ retourne l'extension de A, i.e. l'ensemble des objets satisfaisant A. Le second, noté g, correspond à la notion de généralisation : étant donné un ensemble d'objets O, $g(O)$ retourne l'intention de O, i.e., l'ensemble des descripteurs vérifiés par tous les objets de O. Considérons à nouveau le concept de 'voiture'. Une voiture peut être décrite par les descripteurs `bruyante`, `silencieuse`, `chère`, `bon_marché`, `couleur_vive`, `couleur_claire`. Supposons que l'on ait trois exemples de voitures définies par : `bruyante` $\wedge$ `chère` $\wedge$ `couleur_vive` pour la première, `bruyante` $\wedge$ `chère` $\wedge$ `couleur_claire` pour la deuxième et `bruyante` $\wedge$ `bon_ marché` $\wedge$ `couleur_claire` pour la troisième. Le descripteur `bruyante` a pour extension les trois voitures alors que l'ensemble formé par les deux descripteurs `bruyante` et `chère` n'a pour extension que les deux premières voitures. L'intension des deux premières voitures, i.e., les descripteurs qu'elles ont en commun est composée de `bruyante` et `chère`. Ce couple d'opérateurs forme une correspondance de Galois : f et g sont anti-monotones (si $A_1 \subseteq A_2$ alors $f(A_2) \subseteq f(A_1)$) et si $O_1 \subseteq O_2$ alors $g(O_2) \subseteq g(O_1)$), tout ensemble de descripteurs est inclus dans l'intension de son extension (pour tout A de $\mathcal{A}$, on a $A \subseteq g(f(A))$) et tout ensemble d'objets est inclus dans l'extension de son intension (pour tout O de $\mathcal{O}$, $O \subseteq f(g(0))$). Enfin, on a les propriétés suivantes : $f(A) = f(g(f(A)))$ et $g(O) = g(f(g(O)))$. L'opérateur gof a reçu une attention particulière dans le domaine de la recherche de motifs fréquents et est appelée *fermeture* d'un ensemble de descripteurs.

Un *concept* est alors défini comme un couple d'objets et de descripteurs, (O, A), tel que O est l'extension de A et A est l'intention de O. Dans l'exemple précédent, le couple formé des deux premières voitures et des descripteurs `bruyante` et `chère` forme un concept. Cette notion est particulièrement utilisée lors de la recherche d'itemsets fréquents puisque si (O, A) est un concept alors A est un ensemble fermé ($A = g(f(A))$) et l'ensemble des itemsets fermés muni de $\subseteq$ est un treillis.

Pour conclure, notons que toutes les stratégies de recherche sont fondées sur le lien entre relation de généralité dans l'espace des hypothèses et couverture dans l'espace des observations. Dans le cas de l'apprentissage à partir d'exemples positifs et négatifs, on va généraliser (resp. spécialiser) une hypothèse pour couvrir plus d'exemples positifs (resp. rejeter plus d'exemples négatifs). Dans celui de la recherche de motifs fréquents (couvrant un pourcentage donné d'observations) la propriété de monotonie de la couverture des motifs (tout sous-ensemble d'un motif fréquent, qui peut aussi être vu comme une conjonction de conditions, est nécessairement également fréquent) est fondamentale ; elle permet d'organiser efficacement la recherche en rendant possible l'élagage de directions de recherche. L'algorithme Apriori, le plus connu en fouille de données, exploite astucieusement cette propriété (voir [Han et Kamber, 2006]).

10.3.2 Quatre illustrations

Induction d'arbre de décision.

L'induction d'arbres de décision est certainement le cas le plus connu d'apprentissage de modèles à structure variable. Notons cependant que contrairement aux approches décrites précédemment, l'algorithme d'apprentissage ne repose pas sur l'exploitation d'une relation de généralité entre modèles, mais il utilise une stratégie gloutonne construisant un arbre de plus en plus complexe correspondant à un découpage de plus en plus fin de l'espace $\mathcal{X}$ des observations.

En apprentissage artificiel, un arbre de décision comporte des nœuds et des arcs entre ces nœuds. Chaque nœud interne correspond à un test sur l'un des attributs de description des données, par exemple `taille > 1.70m`, alors que les feuilles correspondent à l'une des étiquettes de classe. Pour classer une observation, on effectue le test à la racine de l'arbre et en fonction de la réponse, on suit la branche correspondante vers l'un des sous-arbres, et ce jusqu'à arriver à une feuille qui est alors l'étiquette prédite. Il est notable qu'un arbre de décision peut s'exprimer sous la forme d'une conjonction de règles de prédiction, chacune d'entre elles ayant pour condition une conjonction de tests depuis la racine de l'arbre jusqu'à une feuille et pour conclusion l'étiquette de classe associée (voir figure 7).

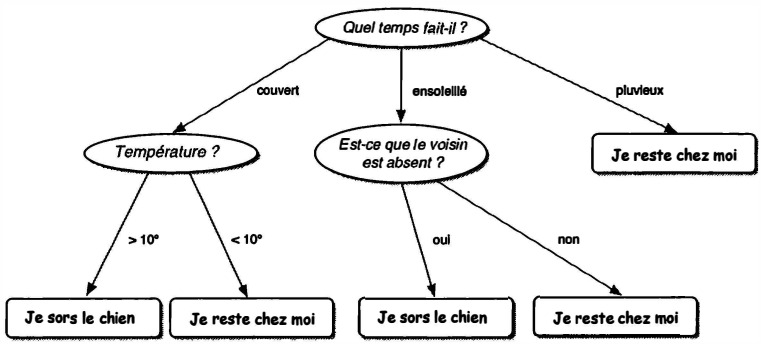


FIGURE 7 – *Un arbre de décision concernant la promenade du chien.*

Un arbre de décision est ainsi l'expression symbolique d'une partition de l'espace des entrées. L'apprentissage consiste à trouver une telle partition en cherchant à optimiser un critère inductif. Les algorithmes existants opèrent par partitions successives en sous-espaces, chaque partition correspondant à un test sur l'un des attributs. Le critère inductif global est remplacé par un critère local qui optimise l'homogénéité, en termes d'étiquette, des classes produites dans la partition. Les critères les plus utilisés sont certainement le gain d'information utilisé dans C5.0 [Quinlan, 1993 ; Kotsiantis, 2007] et fondé sur la notion d'entropie ou l'indice de Gini, utilisé dans le système Cart [Breiman *et al.*, 1984]. On ne peut donc obtenir que des partitions *parallèles aux axes* dans la mesure où les tests sur des variables numériques sont en général de la forme $X \geq \textit{seuil}$. Des techniques d'élagage de l'arbre construit sont ensuite appliquées afin d'éviter la sur-adaptation aux données.

Cet algorithme est de complexité réduite, en $\mathcal{O}(m \cdot d \cdot \log(m))$ où m est la taille

de l'échantillon d'apprentissage et d le nombre d'attributs de description. De plus, il permet d'obtenir des hypothèses généralement faciles à interpréter. C'est l'exemple type d'un algorithme de type *diviser pour règner* avec exploration gloutonne.

Espace des versions et algorithme d'élimination des candidats

Tom Mitchell a montré à la fin des années soixante-dix [Mitchell, 1982, 1997] comment on pouvait exploiter de manière très intéressante une relation d'ordre partiel de généralité pour apprendre l'*espace des versions* c'est-à-dire l'ensemble de toutes les hypothèses qui sont cohérentes avec les données connues à un instant donné, c'est-à-dire qui sont complètes (couvrent tous les exemples positifs) et correctes (ne couvrent aucun exemple négatif). Ces hypothèses sont donc de risque empirique nul. Sous certaines conditions en effet, on peut montrer que l'ensemble des hypothèses cohérentes avec les données d'apprentissage sont bornées par deux ensembles frontière dans le treillis de généralisation défini sur $\mathcal{H}$. L'un, dénommé le *S-set* est l'ensemble des hypothèses les plus spécifiques couvrant les exemples positifs et excluant les exemples négatifs. L'autre, le *G-set*, est l'ensemble des hypothèses maximalement générales également cohérentes avec les données d'apprentissage.

L'algorithme d'élimination des candidats conçu par Tom Mitchell permet de mettre à jour ces deux ensembles frontière en considérant séquentiellement les exemples d'apprentissage. Il correspond à une recherche bidirectionnelle en largeur d'abord qui maintient incrémentalement, après chaque nouvel exemple, le *S-set* et le *G-set*. Il élimine les candidats, c'est-à-dire les hypothèses, car chaque nouvel exemple ajoute des contraintes sur les hypothèses possibles, celles de risque empirique nul. En supposant que le langage de description des hypothèses soit parfaitement choisi, et que les données soient suffisantes et non bruitées, l'algorithme peut en principe converger vers une hypothèse unique qui est le concept cible. Une excellente description de cet algorithme se trouve dans le chapitre 2 du livre de Tom Mitchell [Mitchell, 1997].

Alors que cette technique, qui est à la base de nombreux algorithmes d'apprentissage d'expressions en logique ainsi que d'automates à états finis, connaissait une baisse d'intérêt avec l'avénement des méthodes plus numériques, elle est redevenue d'actualité grâce à l'émergence du domaine de la fouille de données et aux travaux de chercheurs provenant davantage de la communauté des bases de données.

Programmation Logique Inductive.

La Programmation Logique Inductive (PLI) a fait l'objet d'une attention particulière depuis les années 80. Initialement étudiée pour la synthèse de programmes logiques à partir d'exemples, elle s'est progressivement orientée vers l'apprentissage de connaissances dans des formalismes du 1er ordre à partir de données relationnelles, dépassant ainsi le cadre classique de données décrites dans des espaces vectoriels et permettant de prendre en compte les relations entre les données. Lors de la synthèse de programmes logiques, un problème clef concernait l'apprentissage de concepts récursifs ou mutuellement récursifs. Un nouvel intérêt pour ce problème s'est fait jour avec les récents développements en ASP (Answer Set Programming) (voir le chapitre II.4) L'évolution de la PLI l'a conduite à s'attaquer à d'autres problèmes comme le traitement des

données numériques et le traitement des données bruitées.

L'un des intérêts de la PLI est la possibilité de décrire des connaissances du domaine, permettant d'obtenir des définitions de concepts plus intéressantes. Par exemple si l'on souhaite apprendre le concept de 'grand-père' à partir d'exemples de personnes liées par la relation père ou mère, introduire le concept de 'parent' permettra d'obtenir une définition plus concise du concept. La définition de θ-subsomption définie en Section 10.3.1 doit alors être modifiée en conséquence.

L'une des principales difficultés à laquelle doit s'attaquer la PLI est la complexité due à la taille de l'espace des hypothèses et la complexité du test de couverture. Pour pallier ces problèmes, on définit des biais syntaxiques permettant de réduire la taille de l'espace de recherche ou des biais sémantiques. On a aussi cherché à profiter pleinement des travaux menés en apprentissage propositionnel. Dans ce cadre, une technique assez couramment utilisée, appelée propositionnalisation, consiste à transformer le problème d'apprentissage en 1er ordre en un problème propositionnel ou attribut-valeur (voir un exemple dans [Zelezný et Lavrac, 2006]). Toute la difficulté réside alors dans la construction de caractéristiques pertinentes reflétant le caractère relationnel des données et minimisant la perte d'information.

Les systèmes les plus connus de PLI sont certainement les systèmes FOIL [Quinlan, 1996] et PROGOL [Muggleton, 1995]. FOIL adopte une stratégie descendante : la clause la plus générale (sans condition dans le corps de la règle) est successivement raffinée pour rejeter tous les exemples négatifs. Il repose sur une stratégie gloutonne : une heuristique, proche du gain d'information et fondée sur le nombre d'exemples positifs couverts et le nombre d'exemples négatifs rejetés mesure la qualité des spécialisations, la meilleure est choisie sans retour arrière. Cette stratégie souffre d'un problème connu : il peut être nécessaire d'ajouter des raffinements, correspondant à des dépendances fonctionnelles, qui ne permettent pas de rejeter des exemples négatifs mais indispensables pour construire une clause cohérente. PROGOL quant à lui choisit un exemple positif, construit sa clause saturée (la plus spécifique couvrant cet exemple) et effectue sa recherche dans l'espace des généralisations de cette clause saturée. Ses résultats dépendent de l'ordre dans lequel les exemples positifs sont traités. Intéressant en terme de stratégies de recherche, le système ALEPH (cf. `http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html`) peut être considéré comme une plate-forme permettant d'implanter différentes stratégies.

Durant les années quatre-vingt dix, certains travaux en informatique ont montré qu'une grande classe de problèmes de satisfaction de contraintes présentait un phénomène de *transition de phase*, à savoir une variation brutale de la probabilité de trouver une solution quand on varie les paramètres du problème. Peu de temps après, il a été noté que l'apprentissage de programmes logiques peut être réduit à un problème de satisfaction de contraintes (trouver une hypothèse couvrant les exemples positifs, mais non les négatifs). Il a alors été montré empiriquement qu'un phénomène de transition de phase apparaît effectivement dans la programmation logique inductive dès que le problème n'est plus trivial. Cette découverte a été étendue à certains types de problèmes en inférence grammaticale. L'impact de ce phénomène de transition de phase et les moyens d'y faire face font l'objet de débats [Saitta *et al.*, 2011].

Notons enfin que si la classification supervisée a été longtemps la principale tâche

étudiée en ILP, il existe actuellement des travaux sur la recherche de motifs fréquents dans les bases de données relationnelles, ou la découverte de sous-groupes.

Parmi les références importantes, citons [Lavrac et Dzeroski, 1994; Dzeroski et Lavrac, 2001; Raedt, 2008; Fürnkranz *et al.*, 2012].

La recherche de motifs fréquents et de règles d'association : l'algorithme Apriori.

On appelle *motif* (ou *itemset*) toute conjonction de propositions ou de relations attribut-valeur, par exemple $(\hat{a}ge > 60) \wedge (HDL\text{-}cholesterol > 1{,}65\ mmol/L)$, suivant le langage de description choisi et on appelle *motif fréquent* tout motif dont la fréquence d'apparition dans une base de données dépasse un certain seuil appelé *support*. Un problème essentiel en fouille de données est de rechercher les motifs fréquents. Entre autres, cela permet ensuite d'identifier des règles d'association que l'on espère intéressantes, une règle d'association étant de la forme $I \rightarrow J$ où I et J sont des motifs disjoints.

Comme dans la technique de calcul de l'espace des versions, il est naturel de considérer la relation de généralité entre motifs. Un motif est plus général qu'un autre si les exemples dans lesquels il apparaît contient les exemples dans lesquels apparaît le second. L'opérateur de généralisation (resp. spécialisation) est ici la suppression (resp. ajout) d'un terme du motif. On peut alors définir un treillis modélisant la relation de généralité dans l'espace des motifs. A ce treillis est associée une relation d'anti-monotonie : tout motif non fréquent ne peut être spécialisé en un motif fréquent, c'est-à-dire que si un motif n'est pas fréquent, il est inutile de lui ajouter des termes, il restera non fréquent.

Cette observation est à la base de l'algorithme Apriori [Agrawal et Srikant, 1994], certainement le plus connu des algorithmes de recherche de motifs fréquents et de règles d'association. L'algorithme se décompose en deux étapes : la recherche des motifs fréquents puis la construction à partir de ces motifs des règles d'association. La première étape nécessite de confronter les hypothèses à la base de données, elle pose des problèmes de complexité importants et c'est donc celle qui a reçu le plus d'attention.

Dans le cadre le plus simple, Apriori effectue une recherche en largeur dans le treillis, considérant d'abord les motifs de taille 1, puis ceux de taille 2 et ainsi de suite. La base de données n'est parcourue qu'une seule fois par niveaux pour calculer le support des motifs de ce niveau. Pour élaguer la recherche, la contrainte d'anti-monotonie est utilisée. La complexité dépend de la taille de la base de données (parcourue pour calculer les supports) et du nombre de parcours de la bases de données, que l'on peut estimer comme de l'ordre de la taille des plus grands motifs fréquents.

La complexité d'Apriori restant trop élevée, de nombreux algorithmes ont été développés, soit proposant de nouvelles stratégies de recherche (par partitionnement de la base de données ou par échantillonnage), soit reposant sur des représentations différentes des bases de données : association à chaque itemset des identifiants de la base de données (APriori-Tid [Agrawal et Srikant, 1994]), représentation sous forme d'un arbre (FP-Growth, [Han *et al.*, 2004]), ...

Enfin, notons qu'une autre source de complexité est le nombre de motifs fréquents engendrés. On s'est donc intéressé à définir des représentations condensées de ces motifs. La plupart des travaux reposent sur la notion de connexion de Galois et de fermeture

définie comme suit : un motif est fermé si les observations qui le contiennent n'ont pas d'autres éléments en commun. On retrouve la notion de concept sous-jacente à l'Analyse Formelle de Concepts, décrite en section 10.3.1. On peut remarquer que le support d'un motif peut alors se définir comme le cardinal de son extension (le nombre d'éléments de la base de données qui le vérifient) et on a alors des propriétés exprimant par exemple que deux motifs qui ont la même fermeture ont le même support, ou encore que si un motif est inclus dans un autre et a même support alors les deux motifs ont même fermeture. Ces propriétés permettent de montrer que l'ensemble des motifs fermés avec leur support forme une représentation exacte de l'ensemble des motifs et qu'il suffit de ne stocker en mémoire que les motifs fermés (voir par exemple [Zaki, 2000] et [Bastide *et al.*, 2000] pour un travail sur la recherche de fermés).

10.4 Méta-apprentissages

Un célèbre théorème, le *no-free lunch theorem*, affirme qu'aucune méthode d'apprentissage n'est uniformément supérieure à n'importe quelle autre méthode sur tous les problèmes d'induction possibles. La raison profonde en est qu'*a priori* il n'existe aucun lien entre les données d'apprentissage et les données test. Si l'apprentissage revient à jouer contre un adversaire sans contrainte. il ne peut gagner en moyenne. Il est donc nécessaire de s'appuyer sur l'hypothèse qu'il existe une « continuité » entre le passé et l'avenir autorisant l'extrapolation des régularités détectées dans le passé au futur. Concrètement, la conséquence de ce théorème est que chaque méthode est adaptée à une classe de problèmes, mais est mauvaise en dehors de cette classe, c'est-à-dire éventuellement pire qu'une méthode de prédiction opérant au hasard. Et il est tout à fait illusoire de vouloir démontrer la supériorité d'un algorithme d'apprentissage dans l'absolu (en dehors de considération d'efficacité). Il est à noter que ce théorème n'est pas propre à l'apprentissage, mais a des contreparties dans plusieurs domaines, en optimisation en particulier, ainsi que pour le problème de la détermination de la meilleure distance dans des problèmes de clustering (voir par exemple : [Wolpert, 1996b,a]).

Puisqu'il est illusoire de chercher un algorithme idéal, il devient intéressant de déterminer automatiquement l'algorithme adapté au problème courant, voire de combiner des algorithmes. Parce que l'on suppose donnée une famille de méthodes d'apprentissage et qu'il s'agit de trouver comment les appliquer à meilleur escient, on parle de « méta-apprentissage ». Un exemple minimal de méta-apprentissage aurait comme argument deux méthodes d'apprentissage de concept $\mathcal{A}_1$ et $\mathcal{A}_2$ (par exemple un SVM et une méthode d'induction d'arbres de décision) et produirait une hypothèse $H(\mathbf{x}) = \text{signe}\{\alpha\, h_1(\mathbf{x}) + (1-\alpha)\, h_2(\mathbf{x})\}$ à partir des deux hypothèses h_1 et h_2 produites par $\mathcal{A}_1$ et $\mathcal{A}_2$ à partir d'un échantillon d'apprentissage, le problème étant ici de déterminer le meilleur coefficient $\alpha \in [0,1]$.

Il existe de multiples méthodes de méta-apprentissage. La plus large famille concerne des méthodes d'apprentissage de combinaisons d'hypothèses, à l'instar de l'exemple cité plus haut. Elles incluent les méthodes de *boosting* et de *bagging*. Une autre famille consiste à combiner des apprentissages s'appuyant sur des descriptions des données différentes, comme par exemple l'en-tête des mails, d'une part, et leur texte, d'autre part. On parle alors de *co-apprentissage* [Blum et Mitchell, 1998].

Un autre type de méta-apprentissage consiste à régler automatiquement les méta-paramètres de l'algorithme utilisé, ceux qui contrôlent l'espace d'hypothèses considéré (encore appelé *biais de représentation*), ou bien la stratégie d'exploration de cet espace (appelé *biais de recherche*).

10.4.1 Méta-apprentissage par comités d'experts

Le *boosting.* Peut-être le plus célèbre des algorithmes de méta-apprentissage, le *boosting* est dû à Bob Shapire et à Yoav Freund dans les années quatre-vingt-dix [Shapire et Freund, 2012]. Stimulé par une question de son directeur de thèse, Michael Kearns, Shapire a d'abord montré qu'il était possible de prendre un algorithme d'apprentissage de concept à peine meilleur que la prédiction au hasard et d'en tirer une règle de prédiction supérieure en découpant astucieusement l'échantillon d'apprentissage initial en trois sous-échantillons à partir desquels sont appris trois hypothèses h_1, h_2 et h_3 qui sont alors combinées de manière linéaire : $H(\mathbf{x}) =$ vote majoritaire $\{h_1(\mathbf{x}), h_2(\mathbf{x}), h_3(\mathbf{x})\}$. L'idée essentielle est d'apprendre trois hypothèses qui se complètent au sens où la deuxième est bonne là où la première n'était pas meilleure que le hasard, et la troisième est apprise pour départager les deux premières là où elles sont en désaccord. Après ce premier résultat d'un intérêt surtout théorique, Schapire et Freund ont généralisé cette méthode dans un algorithme, Adaboost [Freund et Schapire, 1996], qui apprend itérativement T hypothèses en focalisant à chaque étape t, l'hypothèse courante sur les régions de l'espace des entrées où les hypothèses précédentes étaient mauvaises.

1. Apprentissage d'une hypothèse h_t sur l'échantillon d'apprentissage courant $\mathcal{S}_t$
2. Modification de l'échantillon courant pour produire $\mathcal{S}_{t+1}$ en surpondérant les exemples sur lesquels h_t s'est trompée et en sous-pondérant les autres.

Ces étapes sont répétées T fois pour obtenir finalement l'hypothèse combinée :

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \, h_t(\mathbf{x})$$

Les coefficients α_t sont calculés en fonction de la performance de l'hypothèse h_t sur l'échantillon courant $\mathcal{S}_t$. Meilleure est la performance, plus grand est ce coefficient, ce qui revient naturellement à donner plus de poids aux « experts » qui se sont révélés les meilleurs. Nous renvoyons à [Cornuéjols et Miclet, 2010 ; Shapire et Freund, 2012 ; Zou, 2012] par exemple pour plus de détails.

Cet algorithme, très simple à implanter, donne souvent des résultats excellents, dans la mesure où les hypothèses sont suffisamment variables quand l'échantillon d'apprentissage varie (par pondération des exemples) et où les données sont peu bruitées. De plus, il est remarquablement robuste au sur-apprentissage, ce que des études théoriques ont expliqué par un argument de type « recherche d'une marge maximale », à l'instar des méthodes à noyaux.

Le *bagging.* Dû à Breimann [Breiman, 1996], le *bagging* est une autre méthode très usitée de combinaison d'hypothèses dans laquelle la manipulation de l'échantillon d'ap-

prentissage à chaque étape se fait par un tirage aléatoire avec remise dans l'échantillon initial et où l'hypothèse finale est simplement la moyenne des hypothèses apprises.

Les arbres de décision. Il est remarquable que les méthodes décrites précédemment combinent des « experts » (hypothèses) locaux, au sens où on a focalisé leur apprentissage sur des régions particulières de l'espace, par un vote applicable en n'importe quel point de l'espace d'entrée $\mathcal{X}$. Intuitivement, il semblerait plus adapté de ne consulter les experts que sur leur domaine d'expertise et pas ailleurs. C'est exactement ce qui est réalisé dans l'apprentissage par arbre de décision. Chaque nœud de l'arbre engendre une subdivision de l'espace $\mathcal{X}$. Ces subdivisions sont poursuivies jusqu'à ce qu'un expert local soit capable de prédire l'étiquette ou la valeur pour toute forme $\mathbf{x}$ de la sous-région correspondante. Il s'agit d'un certain côté d'une forme duale de méta-apprentissage par combinaison d'experts.

10.4.2 Apprendre à paramétrer les algorithmes

Chaque algorithme d'apprentissage est associé à un principe inductif (e.g. minimisation du risque empirique) et à une famille de méthodes (e.g. perceptron multicouche) et met en jeu des méta-paramètres (e.g. nombre de couches et de neurones cachés). Ces paramètres sont susceptibles de gouverner l'espace des hypothèses exploré par l'apprenant, la stratégie exploratoire, la taille de la mémoire, etc. Ils sont fréquemment réglés par l'utilisateur, à partir de son expérience et en fonction des caractéristiques de la tâche étudiée afin de maximiser la performance en généralisation de l'algorithme utilisé. Pourquoi cependant ne pas étendre l'apprentissage au contrôle de ces méta-paramètres ?

Tel qu'il est conçu actuellement, l'apprentissage est associé à un problème d'optimisation lié au principe inductif mis en jeu. On cherche un modèle du monde (ou hypothèse) optimisant une certaine fonction définie sur l'échantillon d'apprentissage disponible avec l'espoir que le modèle appris sera performant dans de nouvelles situations, ce que l'on appelle la performance en généralisation. L'apprentissage des méta-paramètres consiste à jouer sur le problème d'optimisation lui-même en le soumettant à un méta-problème d'optimisation dans lequel l'optimisation porte sur la performance en généralisation (voir algorithme 10.1).

Algorithme 10.1 : ALGORITHME DE MÉTA-APPRENTISSAGE

début
  répéter
    Réglage des méta-paramètres
    répéter
      Apprentissage de l'hypothèse la meilleure sur $\mathcal{S}_n$: $\hat{h}(\mathcal{S}_n)$
    jusqu'à *Optimisation du critère inductif sur l'échantillon* $\mathcal{S}_n$
  jusqu'à *Optimisation de la performance en généralisation*

Ainsi, une approche générale mise en avant par Vapnik, appelée *Structural Risk Minimization*, consiste à considérer des espaces d'hypothèses de capacité d'approximation (ou d'expressivité) croissante et à chercher dans chacun de ces espaces la meilleure hy-

pothèse $\hat{h}$. Normalement, la performance en généralisation de la séquence d'hypothèses correspondante présente un optimum général. On choisit alors l'espace d'hypothèses associé comme espace à explorer pour des apprentissages ultérieurs dans les mêmes conditions.

Une question cruciale devient alors celle de l'évaluation de la performance en généralisation d'une hypothèse.

Estimation de la performance en généralisation. Le cœur du problème est dû à ce que par définition, lors de l'apprentissage, la performance en généralisation ne peut être estimée qu'en utilisant les données disponibles, c'est-à-dire l'échantillon d'apprentissage $\mathcal{S}_m$. Pour qu'il y ait bien deux problèmes d'optimisation emboîtés, il est indispensable que le problème d'optimisation lié à l'apprentissage de l'hypothèse $\hat{h}$ (bouche interne) n'utilise pas les mêmes données que le problème d'optimisation des méta-paramètres (boucle externe). Sans entrer dans les détails, on risque sinon d'obtenir des résultats optimistes et biaisés.

Lorsque les données sont abondantes, il est possible de réserver une part significative de l'échantillon pour l'estimation de la performance. On apprend alors $\hat{h}$ sur un sous-échantillon $\mathcal{S}_n \subset \mathcal{S}_m$ et on évalue la performance sur l'échantillon $\mathcal{S}_m \setminus \mathcal{S}_n$, généralement appelé échantillon de validation. Souvent, cependant, les données sont rares et il faut recourir à des procédés tels que la *validation croisée* pour à la fois apprendre l'hypothèse $\hat{h}$ et évaluer aussi précisément que possible la performance en généralisation (voir [Cornuéjols et Miclet, 2010 ; Japkowicz, 2011] pour des informations plus complètes).

La seule estimation du taux d'erreur en généralisation est parfois insuffisante pour l'objectif que l'on se fixe. Ainsi, par exemple, il peut être utile d'estimer plus précisément les taux de faux positifs et de faux négatifs, ou bien encore de précision et de rappel. Lorsque l'on fait varier certains méta-paramètres de l'algorithme, on peut alors obtenir des courbes montrant l'évolution de ces critères. La courbe ROC (de l'anglais *Receiver Operating Characteristics*, motivé par le développement des radars durant la seconde guerre mondiale) en est l'exemple le plus connu. On cherchera alors typiquement à optimiser l'aire sous la courbe ROC qui caractérise le pouvoir discriminant de la méthode de classification employée.

Il faut souligner que dans le cas de données non étiquetées, en apprentissage non supervisé, il existe des critères spécifiques d'estimation de la performance. Ils sont alors très dépendants d'hypothèses *a priori* sur le type de régularités présentes dans le monde, et peuvent facilement donner des résultats très trompeurs si ces hypothèses ne sont pas vérifiées.

Exemples de méta-paramètres susceptibles d'être optimisés automatiquement. Les méta-paramètres en jeu dans l'apprentissage concernent essentiellement d'une manière ou d'une autre l'espace des hypothèses considéré et la manière de l'explorer. Voici une liste non exhaustive d'exemples.

— *Réglage direct de l'espace des hypothèses.* On citera par exemple les techniques permettant de contrôler automatiquement l'architecture des réseaux connexionnistes, soit en ajoutant des neurones formels et leurs connexions (e.g. *cascade*

correlation), soit au contraire en retirant des connexions ou des neurones (e.g. *optimal brain damage*).

— *Réglage indirect de l'espace des hypothèses.* Les techniques de régularisation qui contrôlent un compromis entre l'adéquation aux données et la régularité des hypothèses considérées peuvent également faire l'objet d'une procédure automatique de réglage du compromis. Par ailleurs, la régularisation elle-même peut jouer sur différentes caractéristiques de l'espace des hypothèses. Souvent, on caractérise la régularité d'une hypothèse par la norme ℓ_2 appliquée à ses paramètres, l'idée étant de pénaliser les hypothèses peu « lisses » [4]. Récemment, les recherches se sont tournées vers la norme ℓ_1 qui permet de favoriser les hypothèses parcimonieuses, c'est-à-dire mettant en jeu peu de paramètres. Dans le cas de modèles linéaires, cela se traduit par la préférence pour des modèles s'exprimant à l'aide de peu de variables descriptives. Cela permet une sélection des attributs les plus pertinents ou informatifs pour la tâche considérée.
Les méthodes d'élagage d'arbres de décision peuvent être considérées comme appartenant à cette famille de méthodes, de même que les techniques dites de *weight decay* dans les réseaux connexionnistes.

— *Contrôle du processus d'exploration de l'espace des hypothèses.* Toujours dans le but de contrôler le risque de sur-apprentissage, certaines techniques limitent le processus de recherche de la meilleure hypothèse $\hat{h}$, par exemple en arrêtant l'optimisation avant que ne soit atteint un optimum. L'idée sous-jacente est qu'ainsi on empêche l'algorithme de capturer des régularités accidentelles, propres à l'échantillon de données disponibles mais non représentatives des vraies régularités du monde. On citera la *règle d'arrêt prématuré* utilisée dans les réseaux connexionnistes.

— Dans les méthodes à base de comparaison à des exemples d'apprentissage (voir section 10.2.2), ce sont les fonctions noyau ou les distances utilisées qui constituent les principaux méta-paramètres à régler. Il existe désormais des approches pour apprendre automatiquement les fonctions noyau prises dans un espace de fonctions disponibles. Ces techniques doivent alors compenser l'absence de connaissances expertes par une taille d'échantillon d'apprentissage plus importante.

Le méta-apprentissage peut également consister à sélectionner automatiquement le meilleur algorithme d'apprentissage pour un type d'application donné. Le méta-apprentissage s'attaque alors directement au « no-free lunch theorem » en cherchant à trouver pour chaque classe de problèmes les algorithmes les mieux adaptés. Inversement, on peut aussi chercher à déterminer la « carte de compétences » de chaque algorithme. Étant donné alors un problème, un algorithme maître décide quelle est la meilleure méthode pour le résoudre. Si cette approche est séduisante, il faut reconnaître que les efforts passés, parfois au niveau de programmes européens, n'ont pas sensiblement modifié l'activité des praticiens de l'apprentissage artificiel qui continuent à faire appel à leur expérience [Giroud-Carrier *et al.*, 2004].

---

4. Une norme ℓ_p appliquée à une fonction paramètrée h s'exprime par : $\ell_p(h) = \left\{\sum_{i=1}^{n} w_i^p\right\}^{1/p}$, où les w_i sont les paramètres, supposés ici en nombre fini, de h.

10.5 Conclusion

Nous avons décrit jusqu'ici le cadre classique de l'apprentissage artificiel. Il se caractérise par le fait qu'il est davantage orienté vers la capacité de prédiction que par celle de compréhension. L'accent est donc mis sur la production d'hypothèses ou de modèles du monde qui sont souvent des fonctions d'un espace d'observations à un espace d'étiquettes, et qui sont assez peu intégrées à une théorie du domaine. Par ailleurs, dans ce cadre, les données appartiennent généralement à un espace vectoriel, c'est-à-dire qu'elles sont décrites par un nombre limité et fixe d'attributs. Les sorties ou étiquettes quant à elles sont souvent unidimensionnelles : une classe ou un réel. Finalement, le cadre classique repose fondamentalement sur l'hypothèse que les données sont tirées aléatoirement à partir d'une distribution stationnaire. C'est ce qui permet en effet de donner une expression à la performance visée sous la forme du risque réel, qui est une espérance. C'est aussi ce qui permet de recourir à des théorèmes centraux limite pour déterminer les performances attendues des algorithmes mis en jeu.

Depuis une dizaine d'années, un nombre croissant de champs d'applications (génomique, analyse de textes, recherche d'information sur le web, étude de réseaux sociaux, ...) ne rentrent pas dans le cadre classique rendant nécessaire l'émergence de nouvelles directions de recherche. On peut les catégoriser en fonction des caractéristiques dépassant le cadre classique :

— *Données non indépendantes et identiquement distribuées.* C'est le cas de l'analyse de sources séquentielles de données provenant par exemple de mesures qui ont une corrélation temporelle, voire spatiale dans les nouvelles applications environnementales. De plus, ces séquences peuvent être issues de *processus non stationnaires.* On parle alors de dérive de concept. L'apprentissage ne peut alors plus se faire en une fois, à partir d'une base d'exemples constituée. Il faut avoir recours à des algorithmes d'apprentissage en ligne. L'une des questions majeures est de déterminer ce qui doit être mémorisé et ce qui peut être oublié tandis que le flux de données est traité, parfois en temps réel. La complexité des algorithmes devient alors un souci majeur. Tandis que la communauté de l'apprentissage artificiel parle d'apprentissage en ligne et de ses variantes (apprentissage incrémental, apprentissage et dérive de concept, ...), la communauté des bases de données et de la fouille de données étudie la fouille à partir de « flux de données ». (Voir [Quinonero-Candela *et al.*, 2009 ; Sugiyama et Kawanabe, 2012 ; Gama et Gaber, 2007 ; Gama, 2010])

— *Données semi-supervisées.* L'association d'une étiquette à chaque observation est souvent une opération coûteuse, requérant par exemple un expert. Un cas trivial est celui de la détection de spams ou pourriels. L'étiquetage des courriels nécessite l'attention de l'utilisateur. Sachant qu'à côté des exemples ainsi étiquetés il existe naturellement une masse de données non étiquetées (e.g. les courriels reçus mais non étiquetés), est-il possible d'en profiter pour aider ou affiner l'apprentissage supervisé ? C'est à ce problème que s'attaque l'apprentissage semi-supervisé. On peut dire grossièrement que si de bons a priori existent sur la forme de la distribution des données, la présence de données non étiquetées peut être bénéfique. En revanche, des a priori erronés peuvent conduire à une dégradation des résultats de l'apprentissage [Cornuéjols et Miclet, 2010].

— L'apprentissage peut avoir comme objectif de *calculer en sortie une structure*, comme par exemple une molécule, un arbre de dérivation grammaticale ou encore la structure typique d'un réseau. Il devient alors intéressant, voire nécessaire, de tirer parti d'une notion de distance ou de similarité dans cet espace de sortie. Dans certains cas, on utilise aussi une distance définie sur l'espace combiné des entrées et des sorties. Si les algorithmes ne sont pas forcément différents, l'apprentissage de sorties structurées introduit cependant tout un ensemble de nouveaux problèmes, mais aussi de nouvelles possibilités.
— Un nombre croissant d'applications implique de calculer en sortie non pas une valeur, par exemple l'étiquette associée à une entrée, mais de *produire un classement* (aussi appelé ordonnancement) de l'ensemble des données fournies en entrée. Ainsi, étant donné le profil d'un utilisateur on cherchera à classer les films ou les ouvrages les plus susceptibles de l'intéresser. Les notions classiques de fonctions de perte et de risque empirique ne peuvent alors plus être utilisées directement. Toute une nouvelle classe d'algorithmes est ainsi développée pour répondre à ces nouveaux objectifs.
— L'un des secteurs de recherche les plus actifs récemment concerne le *graph mining*, c'est-à-dire l'exploitation de données relatives à des graphes. Certaines applications visent à caractériser les nœuds d'un graphe, par exemple à apprendre le concept d'auteur influant ou d'amplificateur de rumeur. D'autres applications se focalisent sur la comparaison de graphes, par exemple des graphes d'interactions cellulaires. Dans tous les cas, de nouvelles techniques de comparaison de données avec les hypothèses demandent à être mises au point.
— L'une des limites de la logique classique est son incapacité à représenter l'incertitude et à permettre le raisonnement incertain. Le calcul des probabilités offre de son côté un cadre rigoureux pour le raisonnement incertain, mais il est limité aux concepts propositionnels. Depuis plusieurs années, en particulier dans le cadre de l'apprentissage relationnel statistique (*Statistical Relational Learning*), des efforts de recherche s'efforcent de marier au mieux les possibilités des deux approches, et de dépasser leurs limites. L'une des premières questions est de définir la notion de « couverture » d'un exemple par une hypothèse. Dans le cadre de la logique, cette notion dépend d'une procédure de preuve qui renvoie un booléen `vrai` ou `faux`. Dans le cadre du calcul probabiliste, la notion de couverture devient probabiliste, renvoyant une valeur de probabilité.

Finalement, il faut noter que le cadre classique de l'apprentissage artificiel a presque toujours considéré que les problèmes d'apprentissage étaient isolés. Pour chaque problème, on développait un nouveau système d'apprentissage et on réalisait un nouvel apprentissage indépendamment de ce qui avait été réalisé ailleurs. Mais pour apprendre à jouer au Go, peut-on profiter d'avoir appris à jouer aux dames, ou, au contraire, faut-il l'éviter ? Les travaux sur le raisonnement par analogie portaient en germe ce genre de questions. Le nouveau courant sur l'*apprentissage par transfert* les explore dans le cadre de l'induction. Cela va modifier les algorithmes d'apprentissage par l'intégration de connaissances passées et le problème de leur utilisation et de leur traduction pour faire face à un nouveau contexte.

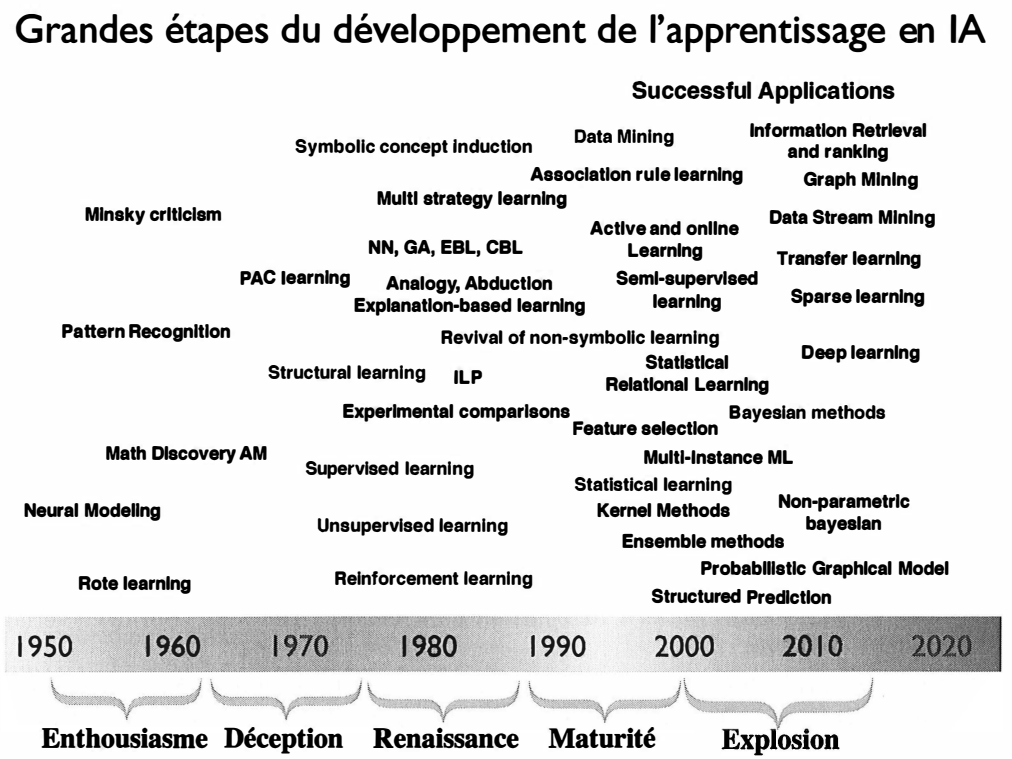


FIGURE 8 – *Esquisse d'une fresque historique présentant diverses techniques d'apprentissage artificiel avec leur date approximative d'apparition.*

Pour ceux qui souhaitent tester ou utiliser des algorithmes d'apprentissage, il existe de nombreux logiciels spécialisés, dont certains dans le domaine public. La référence [Harrington, 2012] décrit plusieurs algorithmes classiques d'apprentissage artificiel en détaillant leur code en Python. Un ouvrage d'approche similaire avec le code en R est [Conway et White, 2012]. La « boîte à outils » Weka est décrite dans [Witten *et al.*, 2011].

Ce chapitre a décrit les grandes classes d'approches algorithmiques existantes. Il ne pouvait être question d'énumérer et de présenter l'ensemble des algorithmes développés au cours des six dernières décennies. Durant cette période, l'accent s'est progressivement déplacé d'approches algorithmiques et symboliques très inspirées par l'étude de la cognition naturelle à des approches de nature bien davantage statistique lorsque la pression des applications a mis en exergue l'exploitation de bases de données décrites en attribut-valeur et supposées indépendantes et identiquement distribuées. Ne pouvant rendre compte en détail de cette évolution, nous terminons cependant par la fresque de la figure 8 évoquant un certain nombre de techniques et leurs dates d'apparition approximatives.

Références

AGRAWAL, R. et SRIKANT, R. (1994). Fast algorithms for mining association rules in large databases. *In Very Large Data Bases (VLDB-94)*, pages 487–499, Santiage, Chile.

ANDERSON, J. (1995). *The architecture of cognition*, volume 5. Lawrence Erlbaum.

ANDERSON, J., KLINE, P. et BEASLEY, C. (1979). A general learning theory and its application to schema abstraction. *Psychology of learning and motivation*, 13 :277–318.

BARBER, D. (2012). *Bayesian Reasoning and Machine Learning.* Cambridge University Press.

BASTIDE, Y., PASQUIER, N., TAOUIL, R., STUMME, G. et LAKHAL, L. (2000). Mining minimal non-redundant association rules using frequent closed itemsets. *In Computational Logic*, pages 972–986.

BENGIO, Y. et CUN, Y. L. (2007). Scaling learning algorithms towards ai. *In* BOTTOU, L., CHAPELLE, O., DECOSTE, D. et WESTON, J., éditeurs : *Large-Scale Kernel Machines*, page 416. MIT Press.

BISHOP, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer-Verlag, Secaucus, NJ, USA.

BLUM, A. et MITCHELL, T. (1998). Combining labeled and unlabeled data with co-training. *In 11th Computational Learning Theory (COLT-98)*, pages 92–100. Morgan Kaufmann.

BREIMAN, L. (1996). Bagging predictors. *Machine Learning*, 24(2) :123–140.

BREIMAN, L., FRIEDMAN, J., OLSHEN, R. et STONE, C. J. (1984). *Classification and regression trees.* Wadsworth and Brooks/Cole Advanced Books and Software.

CONWAY, D. et WHITE, J. M. (2012). *Machine Learning for Hackers.* O'Reilly.

CORMEN, T., LEISERSON, C., RIVEST, R. et STEIN, C. (2001). *Introduction to algorithms.* MIT press.

CORNUÉJOLS, A. et MICLET, L. (2010). *Apprentissage artificiel. Concepts et algorithmes. (2ème édition).* Eyrolles.

de la HIGUERA, C. (2010). *Grammatical inference : learning automata and grammars.* Cambridge University Press.

DREYFUS, G., MARTINEZ, J.-M., SAMUELIDES, M., GORDON, M., BADRAN, F. et THIRIA, S., éditeurs (2008). *Apprentissage statistique.* Eyrolles.

DZEROSKI, S. et LAVRAC, N., éditeurs (2001). *Relational data mining.* Springer.

FREUND, Y. et SCHAPIRE, R. (1996). Experiments with a new boosting algorithm. *In Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156.

FÜRNKRANZ, J., GAMBERGER, D. et LAVRAC, N. (2012). *Foundations of Rule Learning.* Springer.

GAMA, J. (2010). *Knowledge Discovery from Data Streams.* Chapman & Hall.

GAMA, J. et GABER, M. M., éditeurs (2007). *Learning from Data Streams. Processing*

Techniques in Sensor Networks. Springer.

GANTER, B., STUMME, G. et WILLE, R., éditeurs (2005). *Formal Concept Analysis : Foundations and Applications.* Springer.

GANTER, B., WILLE, R. et FRANKE, C. (1998). *Formal Concept Analysis : Mathematical Foundations.* Springer.

GETOOR, L. et TASKAR, B., éditeurs (2007). *An introduction to statistical relational learning.* MIT Press.

GIROUD-CARRIER, C., VILALTA, R. et BRAZDIL, P. (2004). Special issue on meta-learning. *Machine Learning journal*, 54.

HAN, J. et KAMBER, M. (2006). *Data mining. Concepts and techniques (2nd Ed.).* Morgan Kaufmann.

HAN, J., PEI, J., YIN, Y. et MAO, R. (2004). Mining frequent patterns without candidate generation : A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1) :53–87.

HARRINGTON, P. (2012). *Machine Learning in Action.* Manning.

HASTIE, T., TIBSHIRANI, R. et FRIEDMAN, J. (2009). *The elements of statistical learning. Data mining, inference, and prediction.* Springer.

JAPKOWICZ, N. (2011). *Evaluating Learning Algorithms : A Classification Perspective.* Cambridge University Press.

JEBARA, T. (2003). *Machine Learning : Discriminative and Generative.* Springer.

KELLEY, C. (1987). *Iterative methods for optimization*, volume 18. Society for Industrial Mathematics.

KODRATOFF, Y. et MICHALSKI, R. S. (1990). *Machine learning : an artificial intelligence approach*, volume 3. Morgan Kaufmann Publishers.

KOLLER, D. et FRIEDMAN, N. (2009). *Probabilistic Graphical Models. Principles and Techniques.* MIP Press.

KOTSIANTIS, S. B. (2007). Supervised machine learning : A review of classification techniques. *Informatica*, 31 :249–268.

LAVRAC, N. et DZEROSKI, S. (1994). *Inductive logic programming - techniques and applications.* Ellis Horwood series in artificial intelligence. Ellis Horwood.

MICHALSKI, R., CARBONELL, J. et MITCHELL, T. (1986). *Machine learning : An artificial intelligence approach (vol. 1 and 2)*, volume 1 and 2. Morgan Kaufmann.

MICHALSKI, R., CARBONELL, J., MITCHELL, T. et KODRATOFF, Y. (1993). Apprentissage symbolique, une approche de l'intelligence artificielle. *Volumes I and II, Cepadues Editions.*

MICLET, L. (1990). Grammatical inference. *In* BUNKE, H. et SANFELIU, A., éditeurs : *Syntactic and structural pattern recognition theory and applications.* World Scientific.

MINSKY, M. et PAPERT, S. (1988). *Perceptrons (2nd ed.).* MIT Press.

MITCHELL, T. (1979). *Version spaces : An approach to concept learning.* Ph.d. thesis, Stanford University.

MITCHELL, T. (1982). Generalization as search. *Artificial Intelligence journal*, 18 :203–226.

MITCHELL, T. (1997). *Machine Learning.* McGraw-Hill.

MUGGLETON, S. (1995). Inverse entailment and progol. *New Generation Comput.*, 13(3&4) :245–286.

NILSSON, N. (2010). *The quest for artificial intelligence. A history of ideas and achievements.* Cambridge University Press.

PEARL, J. (1988). *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference.* Morgan Kaufmann.

PLOTKIN, G. (1970). A note on inductive generalization. *In Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press.

QUINLAN, J. (1993). *C4.5 : Programs for Machine Learning.* Morgan Kauffman.

QUINLAN, J. R. (1996). Learning first-order definitions of functions. *CoRR*, cs.AI/9610102.

QUINONERO-CANDELA, J., SUGIYAMA, M., SCHWAIGHOFER, A. et LAWRENCE, N., éditeurs (2009). *Dataset shift in machine learning.* MIT Press.

RAEDT, L. D. (2008). *Logical and Relational Learning.* Springer.

RAEDT, L. D., FRASCONI, P., KERSTING, K. et MUGGLETON, S., éditeurs (2008). *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 de *Lecture Notes in Computer Science.* Springer.

ROSENBLATT, F. (1958). The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386.

SAITTA, L., GIORDANA, A. et CORNUÉJOLS, A. (2011). *Phase Transitions in Machine Learning.* Cambridge University Press.

SCHAPIRE, R. (2003). The boosting approach to machine learning : An overview. *In* DENISON, D. D., HANSEN, M. H., HOLMES, C., MALLICK, B. et YU, B., éditeurs : *Nonlinear Estimation and Classification.* Springer.

SCHÖLKHOPF, B. et SMOLA, A. (2002). *Learning with kernels.* MIT Press.

SHAPIRE, R. et FREUND, Y. (2012). *Boosting : Foundations and Algorithms.* MIT Press.

SHAWE-TAYLOR, J. et CRISTIANINI, N. (2004). *Kernel methods for pattern analysis.* Cambridge University Press.

SNYMAN, J. (2005). *Practical mathematical optimization : an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer.

SUGIYAMA, M. et KAWANABE, M. (2012). *Machine Learning in Non-Stationary Environments : Introduction to Covariate Shift Adaptation.* MIT Press.

SURAJ, Z. (2004). An introduction to rough sets theory and its applications : A tutorial. *In ICENCO'2004*, Cairo, Egypt.

VAPNIK, V. (1995). *The nature of statistical learning theory.* Springer.

WINSTON, P. (1970). Learning structural descriptions from examples. Rapport technique, DTIC Document.

WITTEN, I., FRANK, E. et HALL, M. (2011). *Data Mining : Practical Machine Learning Tools and Techniques.* Morgan Kaufmann.

WOLPERT, D. (1996a). The existence of a priori distinctions between learning algorithms. *Neural Computation*, 8(7) :1391–1420.

WOLPERT, D. (1996b). The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7) :1341–1390.

WOOLFOLK, A. (2001). *Educational Psychology.* Allyn and Bacon.

XU, R. et WUNSCH, D. (2008). *Clustering.* Wiley-IEEE Press.

ZAKI, M. J. (2000). Generating non-redundant association rules. *In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, KDD*, pages 34–43.

ZELEZNÝ, F. et LAVRAC, N. (2006). Propositionalization-based relational subgroup discovery with rsd. *Machine Learning*, 62(1-2) :33–63.

ZOU, Z.-H. (2012). *Ensemble Methods : Foundations and Algorithms.* Chapman & Hall.

Chapitre 11

Méta-heuristiques et intelligence artificielle

11.1 Introduction

Une méta-heuristique est une méthode générique pour la résolution de problèmes combinatoires NP-difficiles. La résolution de ces problèmes nécessite l'examen d'un très grand nombre (exponentiel) de combinaisons. Tout un chacun a déjà été confronté à ce phénomène d'explosion combinatoire qui transforme un problème apparemment très simple en un véritable casse-tête dès lors que l'on augmente la taille du problème à résoudre. C'est le cas par exemple quand on cherche à concevoir un emploi du temps. S'il y a peu de cours à planifier, le nombre de combinaisons à explorer est faible et le problème est très rapidement résolu. Cependant, l'ajout de quelques cours seulement peut augmenter considérablement le nombre de combinaisons à explorer de sorte que le temps de résolution devient excessivement long.

L'enjeu pour résoudre ces problèmes est de taille : un très grand nombre de problèmes industriels sont confrontés à ce phénomène d'explosion combinatoire comme, par exemple, les problèmes consistant à planifier une production en minimisant les pertes de temps, ou à découper des formes dans des matériaux en minimisant les chutes. Il est donc crucial de concevoir des approches « intelligentes », capables de contenir ou de contourner l'explosion combinatoire afin de résoudre ces problèmes difficiles en un temps acceptable.

Les problèmes d'optimisation combinatoires peuvent être résolus par deux principales familles d'approches. Les approches « exactes » explorent de façon systématique l'espace des combinaisons jusqu'à trouver une solution optimale. Afin de (tenter de) contenir l'explosion combinatoire, ces approches structurent l'espace des combinaisons en arbre et utilisent des techniques d'élagage, pour réduire cet espace, et des heuristiques, pour déterminer l'ordre dans lequel il est exploré. Ces approches exactes permettent de résoudre en pratique de nombreux problèmes d'optimisation combina-

Auteurs : Jin-Kao Hao et Christine Solnon.

toires. Cependant, les techniques de filtrage et les heuristiques d'ordre ne réduisent pas toujours suffisamment la combinatoire, et certaines instances de problèmes ne peuvent être résolues en un temps acceptable par ces approches exhaustives.

Une alternative consiste à utiliser des méta-heuristiques qui contournent le problème de l'explosion combinatoire en n'explorant délibérément qu'une partie de l'espace des combinaisons. Par conséquent, elles peuvent ne pas trouver la solution optimale, et encore moins prouver l'optimalité de la solution trouvée ; en contrepartie, la complexité en temps est généralement faiblement polynomiale.

Organisation du chapitre. Il existe essentiellement deux grandes familles d'approches heuristiques : les *approches perturbatives*, présentées en 11.2, construisent des combinaisons en modifiant des combinaisons existantes ; les *approches constructives*, présentées en 11.3, génèrent des combinaisons de façon incrémentale en utilisant pour cela un modèle stochastique. Ces différentes approches peuvent être hybridées, et nous présentons en 11.4 quelques-uns des schémas d'hybridation les plus connus. Nous introduisons ensuite en 11.5 les notions d'intensification (exploitation) et de diversification (exploration), communes à toutes ces approches heuristiques : l'intensification vise à diriger l'effort de recherche aux alentours des meilleures combinaisons trouvées, tandis que la diversification vise à garantir un bon échantillonnage de l'espace de recherche. Enfin, nous développerons en 11.6 deux applications de ces méta-heuristiques à la résolution de problèmes classiques de l'intelligence artificielle, à savoir la satisfiabilité de formules booléennes et la satisfaction de contraintes .

Notations. Dans la suite de ce chapitre, nous supposerons que le problème à résoudre est défini par un couple (E, f) tel que E est un ensemble de combinaisons candidates, et $f : E \to \mathbb{R}$ est une fonction objectif associant à chaque combinaison de E une valeur numérique. Résoudre un tel problème consiste à chercher la combinaison $e^* \in E$ qui optimise (maximise ou minimise, selon les cas) f.

Nous illustrerons plus particulièrement les différentes méta-heuristiques introduites dans ce chapitre sur le problème du voyageur de commerce, qui constituera notre « fil rouge » : étant donnés un ensemble V de villes et une fonction $d : V \times V \to \mathbb{R}$ donnant pour chaque paire de villes différentes $\{i, j\} \subseteq V$ la distance $d(i, j)$ séparant les villes i et j, il s'agit de trouver le plus court circuit passant par chaque ville de V une et une seule fois. Pour ce problème, l'ensemble E des combinaisons candidates est défini par l'ensemble des permutations circulaires de V et la fonction objectif f à minimiser est définie par la somme des distances entre deux villes consécutives dans la permutation.

11.2 Méta-heuristiques perturbatives

Les approches perturbatives explorent l'espace des combinaisons E en perturbant itérativement des combinaisons déja construites : partant d'une ou plusieurs combinaisons initiales (généralement prises aléatoirement dans E), l'idée est de générer à chaque étape une ou plusieurs nouvelles combinaisons en modifiant une ou plusieurs combinaisons générées précédemment. Ces approches sont dites « basées sur les instances » dans

[Zlochin *et al.*, 2004]. Les approches perturbatives les plus connues sont les algorithmes génétiques, décrits en 11.2.1, et la recherche locale, décrite en 11.2.2.

11.2.1 Algorithmes génétiques

Les algorithmes génétiques [Holland, 1975 ; Goldberg, 1989 ; Eiben et Smith, 2003] s'inspirent de la théorie de l'évolution et des règles de la génétique qui expliquent la capacité des espèces vivantes à s'adapter à leur environnement par la combinaison des mécanismes suivants :

- la *sélection naturelle* fait que les individus les mieux adaptés à l'environnement tendent à survivre plus longtemps et ont donc une plus grande probabilité de se reproduire ;
- la *reproduction par croisement* fait qu'un individu hérite ses caractéristiques de ses parents, de sorte que le croisement de deux individus bien adaptés à leur environnement aura tendance à créer un nouvel individu bien adapté à l'environnement ;
- la *mutation* fait que certaines caractéristiques peuvent apparaître ou disparaître de façon aléatoire, permettant ainsi d'introduire de nouvelles capacités d'adaptation à l'environnement, capacités qui pourront se propager grâce aux mécanismes de sélection et de croisement.

Les algorithmes génétiques reprennent ces mécanismes pour définir une méta-heuristique. L'idée est de faire évoluer une population de combinaisons, par sélection, croisement et mutation, la capacité d'adaptation d'une combinaison étant ici évaluée par la fonction objectif à optimiser. L'algorithme 11.1 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 11.1 : Algorithme génétique

```
Initialiser la population avec un ensemble de combinaisons de E
tant que critères d'arrêt non atteints faire
    Sélectionner des combinaisons de la population
    Créer de nouvelles combinaisons par recombinaison et mutation
    Mettre à jour la population
retourner la meilleure combinaison ayant appartenu à la population
```

Initialisation de la population : en général, la population initiale est générée de façon aléatoire, selon une distribution uniforme assurant une bonne diversité des combinaisons.

Sélection : cette étape consiste à choisir les combinaisons de la population qui seront ensuite candidates pour la recombinaison et la mutation. Il s'agit là de favoriser la sélection des meilleures combinaisons, tout en laissant une petite chance aux moins bonnes combinaisons. Il existe de nombreuses façons de procéder à cette étape de sélection. Par exemple, la sélection par tournoi consiste à choisir aléatoirement deux combinaisons et à sélectionner la meilleure des deux (ou bien à sélectionner une des deux selon une probabilité dépendant de la fonction objectif). La sélection peut également

être réalisée selon d'autres critères comme la diversité. Dans ce cas de figure, seuls les individus « distancés » sont autorisés à survivre et à se reproduire.

Recombinaison (croisement) : cette étape vise à créer de nouveaux individus par un mélange de combinaisons sélectionnées. L'objectif est de conduire la recherche dans une nouvelle zone de l'espace où de meilleures combinaisons peuvent être trouvées. A cette fin, la recombinaison doit être bien adaptée à la fonction à optimiser et capable de transmettre de bonnes propriétés des parents vers la combinaison créée. De plus, l'opérateur de recombinaison devrait idéalement permettre de créer des descendants diversifiés. Du point de vue de l'exploration et l'exploitation, la recombinaison est destinée à jouer un rôle de diversification stratégique avec un objectif à long terme de renforcement de l'intensification.

Mutation : cette opération consiste à modifier de façon aléatoire certains composants des combinaisons obtenues par croisement.

Exemple 25. Pour le voyageur de commerce, un opérateur de recombinaison simple consiste à recopier une sous-séquence du premier parent, et à compléter la permutation en plaçant les villes manquantes dans l'ordre où elles apparaissent dans le deuxième parent. Un opérateur de mutation classique consiste à choisir aléatoirement quelques villes et les échanger.

Mise à jour de la population : cette étape détermine quelles nouvelles combinaisons doivent devenir membre de la population et quelles anciennes combinaisons de la population doivent être remplacées. La politique de mise-à-jour est essentielle pour maintenir une diversité appropriée de la population, pour prévenir une convergence prématurée du processus de recherche, et pour permettre à l'algorithme de toujours découvrir de nouvelles zones prometteuses dans l'espace de recherche. Ainsi, les décisions sont souvent prises selon des critères liés à la fois à la qualité et à la diversité. Par exemple, une règle bien connue de mise à jour basée uniquement sur la qualité consiste à remplacer la pire des combinaisons de la population, tandis qu'une règle fondée sur la diversité consiste à substituer les anciennes combinaisons de la population par de nouvelles combinaisons similaires, conformément à une mesure de similarité donnée. D'autres critères comme l'age peuvent aussi être considérés.

Critères d'arrêt : le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'un critère d'arrêt soit atteint. Il peut s'agir d'un nombre maximum de générations, un nombre maximum d'évaluations, un nombre maximum de générations sans améliorer la meilleure solution, une qualité de solution atteinte ou encore une diversité de population inférieure à un seuil donné.

11.2.2 Recherche locale

Une recherche locale explore l'espace des combinaisons de proche en proche, en partant d'une combinaison initiale et en sélectionnant à chaque itération une combinaison

voisine de la combinaison courante, obtenue en lui appliquant une transformation élémentaire [Hoos et Stützle, 2005]. L'algorithme 11.2 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 11.2 : Recherche locale

Générer une combinaison initiale $e \in E$
tant que *critères d'arrêt non atteints* **faire**
  Choisir $e' \in v(e)$
  $e \leftarrow e'$
retourner *la meilleure combinaison construite*

Fonction de voisinage : l'algorithme de recherche locale est paramétré par une fonction de voisinage $v : E \rightarrow \mathcal{P}(E)$ définissant l'ensemble des combinaisons que l'on peut explorer à partir d'une combinaison donnée. Étant donnée une combinaison courante $e \in E$, $v(e)$ est un ensemble de combinaisons que l'on peut obtenir en appliquant une modification « élémentaire » à e. On peut généralement considérer différents opérateurs de modification comme, par exemple, changer la valeur d'une variable ou échanger les valeurs de deux variables. Chaque opérateur de modification différent induit un voisinage différent, qui peut contenir un nombre plus ou moins grand de combinaisons plus ou moins similaires à la combinaison courante. Ainsi, le choix de l'opérateur de voisinage a une influence forte sur les performances de l'algorithme. Il est généralement souhaitable que l'opérateur de voisinage permette d'atteindre la combinaison optimale à partir de n'importe quelle combinaison initiale de E, ce qui revient à imposer que le graphe orienté associant un sommet à chaque combinaison de E et un arc (e_i, e_j) à chaque couple de combinaisons telles que $e_j \in v(e_i)$, admette un chemin depuis n'importe lequel de ses sommets jusqu'à la combinaison optimale.

Exemple 26. Pour le problème du voyageur de commerce, l'opérateur *2-opt* consiste à supprimer deux arêtes et les remplacer par les deux nouvelles arêtes qui reconnectent les deux chemins créés par la suppression des arêtes. Plus généralement, l'opérateur *k-opt* consiste à supprimer k arêtes mutuellement disjointes et à réassembler les différents morceaux de chemin ainsi créés en ajoutant k nouvelles arêtes de façon à reconstituer un tour complet. Plus k est grand, plus la taille du voisinage est importante.

Génération de la combinaison initiale : la combinaison à partir de laquelle le processus d'exploration commence est souvent générée de façon aléatoire. Elle est parfois générée en suivant une heuristique constructive gloutonne (voir la section 11.3.1). Lorsque la recherche locale est hybridée avec une autre méta-heuristique, comme par exemple les algorithmes génétiques ou les algorithmes par colonies de fourmis, la combinaison initiale peut être le résultat d'un autre processus de recherche.

Choix du voisin : à chaque itération de la recherche locale, il s'agit de choisir une combinaison dans le voisinage de la combinaison courante. Ce choix d'une combinaison voisine est appelé « mouvement ». Il existe un très grand nombre de stratégies de choix. On peut par exemple sélectionner à chaque itération le meilleur voisin [Selman *et al.*, 1992], c'est-à-dire, celui qui améliore le plus la fonction objectif ou bien le premier

voisin trouvé qui améliore la fonction objectif. De telles stratégies « gloutonnes » (aussi appelées « montées de gradients ») risquent fort d'être rapidement piégées dans des *optima* locaux, c'est-à-dire, sur des combinaisons dont toutes les voisines sont moins bonnes. Pour s'échapper de ces *optima* locaux, on peut considérer différentes méta-heuristiques, par exemple, pour n'en citer que quelques-unes :

- la marche aléatoire (*random walk*) [Selman *et al.*, 1994], qui autorise avec une très petite probabilité de sélectionner un voisin de façon complètement aléatoire ;
- le recuit simulé (*simulated annealing*) [Aarts et Korst, 1989], qui autorise de sélectionner des voisins de moins bonne qualité selon une probabilité qui décroit avec le temps ;
- la recherche taboue (*tabu search*) [Glover et Laguna, 1993], qui empêche de boucler sur un petit nombre de combinaisons autour des *optima* locaux en mémorisant les derniers mouvements effectués dans une liste taboue et en interdisant les mouvements inverses à ces derniers mouvements ;
- la recherche à voisinage variable [Mladenović et Hansen, 1997], qui change d'opérateur de voisinage lorsque la combinaison courante est un optimum local par rapport au voisinage courant.

Répétition du processus de recherche locale : une recherche locale peut être répétée plusieurs fois à partir de combinaisons initiales différentes. Il peut s'agir de combinaisons initiales indépendantes, générées aléatoirement. On parle alors de recherche locale à plusieurs points de départ (*multi-start local search*). Il peut également s'agir de combinaisons initiales obtenues en perturbant une combinaison issue d'un processus de recherche locale précédent. On parle alors de recherche locale itérée (*iterated local search*) [Lourenco *et al.*, 2002]. On peut par ailleurs faire plusieurs recherches locales en parallèle, en partant de différentes combinaisons initiales, et redistribuer régulièrement les combinaisons courantes en supprimant les moins bonnes et dupliquant les meilleures selon le principe « aller avec les meilleurs » (*go with the winner*) [Aldous et Vazirani, 1994].

11.3 Méta-heuristiques constructives

Les approches constructives construisent une ou plusieurs combinaisons de façon incrémentale, c'est-à-dire, en partant d'une combinaison vide, et en ajoutant des composants de combinaison jusqu'à obtenir une combinaison complète. Ces approches sont dites « basées sur les modèles » dans [Zlochin *et al.*, 2004], dans le sens où elles utilisent un modèle, généralement stochastique, pour choisir à chaque itération le prochain composant de combinaison à ajouter à la combinaison en cours de construction.

Il existe différentes stratégies pour choisir les composants à ajouter à chaque itération, les plus connues étant les stratégies gloutonnes aléatoires, décrites en 11.3.1, les algorithmes par estimation de distribution, décrits en 11.3.2 et la méta-heuristique d'optimisation par colonies de fourmis, introduite en 11.3.3.

11.3.1 Algorithmes gloutons aléatoires

Les algorithmes gloutons (*greedy*) construisent une combinaison en partant d'une combinaison vide et en choisissant à chaque itération un composant de combinaison pour lequel une heuristique donnée est maximale. Un algorithme glouton est capable de construire une combinaison très rapidement, les choix effectués n'étant jamais remis en cause. La qualité de la combinaison construite dépend de l'heuristique.

Exemple 27. Un algorithme glouton pour le problème du voyageur de commerce peut être défini de la façon suivante : partant d'une ville initiale choisie aléatoirement, on se déplace à chaque itération sur la ville non visitée la plus proche de la dernière ville visitée, jusqu'à ce que toutes les villes aient été visitées.

Les algorithmes gloutons aléatoires (*greedy randomized*) construisent plusieurs combinaisons et adoptent également une stratégie gloutonne pour le choix des composants à ajouter aux combinaisons en cours de construction, mais ils introduisent un peu d'aléatoire afin de diversifier les combinaisons construites. On peut par exemple choisir aléatoirement le prochain composant parmi les k meilleurs ou bien parmi ceux qui sont à moins de α pour cent du meilleur composant [Feo et Resende, 1989]. Une autre possibilité consiste à choisir le prochain composant en fonction de probabilités dépendant de la qualité des différents composants [Jagota et Sanchis, 2001].

Exemple 28. Pour le problème du voyageur de commerce, si la dernière ville visitée est i, et si C contient l'ensemble des villes qui n'ont pas encore été visitées, on peut définir la probabilité de visiter la ville $j \in C$ par $p(j) = \frac{[1/d(i,j)]^\beta}{\sum_{k \in C}[1/d(i,k)]^\beta}$. β est un paramètre permettant de régler le niveau d'aléatoire : si $\beta = 0$, alors toutes les villes non visitées ont la même probabilité d'être choisies ; plus on augmente β et plus on augmente la probabilité de choisir les villes les plus proches.

11.3.2 Algorithmes par estimation de distributions

Les algorithmes par estimation de distribution (*Estimation of Distribution Algorithms* ; EDA) [Larranaga et Lozano, 2001] sont des algorithmes gloutons aléatoires itératifs : à chaque itération un ensemble de combinaisons est généré selon un principe glouton aléatoire similaire à celui décrit en 11.3.1. Cependant, les EDA exploitent les meilleures combinaisons construites lors des itérations précédentes pour construire de nouvelles combinaisons. L'algorithme 11.3 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 11.3 : Algorithmes par estimation de distributions

Générer une population de combinaisons $P \subseteq E$
tant que *critères d'arrêt non atteints* **faire**
  Construire un modèle probabiliste M en fonction de P
  Générer de nouvelles combinaisons à l'aide de M
  Mettre à jour P en fonction des nouvelles combinaisons
retourner *la meilleure combinaison ayant appartenu à la population*

Génération de la population initiale P : en général la population initiale est générée de façon aléatoire selon une distribution uniforme, et seules les meilleures combinaisons générées sont gardées.

Génération du modèle probabiliste M : différents types de modèles probabilistes, plus ou moins simples, peuvent être considérés. Le modèle le plus simple, appelé PBIL [Baluja, 1994], est basé sur la probabilité d'apparition de chaque composant sans tenir compte d'éventuelles relations de dépendances entre les composants. Dans ce cas, on calcule pour chaque composant sa fréquence d'apparition dans la population et on définit la probabilité de sélection de ce composant proportionnellement à sa fréquence. D'autres modèles plus fins, mais aussi plus coûteux, utilisent des réseaux bayésiens [Pelikan *et al.*, 1999]. Les relations de dépendance entre composants sont alors représentées par les arcs d'un graphe auxquels sont associées des distributions de probabilités conditionnelles.

Exemple 29. Pour le problème du voyageur de commerce, si la dernière ville visitée est i, et si C contient l'ensemble des villes qui n'ont pas encore été visitées, on peut définir la probabilité de visiter la ville $j \in C$ par $p(j) = \frac{freq_P(i,j)}{\sum_{k \in C} freq_P(i,k)}$, où $freq_P(i,j)$ donne le nombre de combinaisons de la population P contenant l'arête (i, j). Ainsi, la probabilité de choisir j est d'autant plus forte que la population contient de combinaisons utilisant l'arête (i, j).

Génération de nouvelles combinaisons à l'aide de M : les nouvelles combinaisons sont construites selon un principe glouton aléatoire, les probabilités de sélection des composants étant données par le modèle probabiliste M.

Mise à jour de la population : en général, seules les meilleures combinaisons, appartenant à la population courante ou aux nouvelles combinaisons générées, sont conservées dans la population de l'itération suivante. Il est possible de maintenir par ailleurs une certaine diversité des combinaisons gardées dans la population.

11.3.3 Optimisation par colonies de fourmis

Il existe un parallèle assez fort entre l'optimisation par colonies de fourmis (*Ant Colony Optimization*; ACO) et les algorithmes par estimation de distribution [Zlochin *et al.*, 2004]. Ces deux approches utilisent un modèle probabiliste glouton pour générer des combinaisons, ce modèle évoluant en fonction des combinaisons précédemment construites dans un processus itératif d'apprentissage. L'originalité et la contribution essentielle d'ACO est de s'inspirer du comportement collectif des fourmis pour faire évoluer le modèle probabiliste. Ainsi, la probabilité de choisir un composant est définie proportionnellement à une quantité de phéromone représentant l'expérience passée de la colonie concernant le choix de ce composant. Cette quantité de phéromone évolue par la conjugaison de deux mécanismes : un mécanisme de renforcement des traces de phéromone associées aux composants des meilleures combinaisons, visant à augmenter la probabilité de sélection de ces composants ; et un mécanisme d'évaporation,

visant à privilégier les expériences récentes par rapport aux expériences plus anciennes. L'algorithme 11.4 décrit ce principe général, dont les principales étapes sont détaillées ci-après.

Algorithme 11.4 : Optimisation par colonies de fourmis

Initialiser les traces de phéromone à τ_0
tant que *les conditions d'arrêt ne sont pas atteintes* **faire**
  Construction de combinaisons par les fourmis
  Mise à jour de la phéromone
retourner *la meilleure combinaison construite*

Structure phéromonale : La phéromone est utilisée pour biaiser les probabilités de choix des composants de combinaisons lors de la construction gloutonne aléatoire d'une combinaison. Un point crucial pour la performance de l'algorithme réside donc dans le choix de la structure phéromonale, c'est-à-dire, dans le choix des données sur lesquelles des traces de phéromone seront déposées. Au début de l'exécution d'un algorithme ACO, toutes les traces de phéromone sont initialisées à une valeur τ_0 donnée.

Exemple 30. Pour le problème du voyageur de commerce, une trace τ_{ij} est associée à chaque paire (i, j) de villes. Cette trace représente l'expérience passée de la colonie concernant le fait de visiter consécutivement les villes i et j.

Construction de combinaisons par les fourmis : A chaque cycle d'un algorithme ACO, chaque fourmi construit une combinaison selon un principe glouton aléatoire similaire à celui introduit en 11.3.1. Partant d'une combinaison vide, ou d'une combinaison contenant un premier composant de combinaison choisi aléatoirement ou selon une heuristique donnée, la fourmi ajoute un nouveau composant de combinaison à chaque itération, jusqu'à ce que la combinaison soit complète. A chaque itération, le prochain composant de combinaison est choisi selon une règle de transition probabiliste : étant donnés un début de combinaison S, et un ensemble C de composants de combinaison pouvant être ajoutés à S, la fourmi choisit le composant $i \in C$ selon la probabilité :

$$p_S(i) = \frac{[\tau_S(i)]^\alpha \cdot [\eta_S(i)]^\beta}{\sum_{j \in C} [\tau_S(j)]^\alpha \cdot [\eta_S(j)]^\beta} \tag{11.1}$$

où $\tau_S(i)$ est le facteur phéromonal associé au composant de combinaison i par rapport au début de combinaison S (la définition de ce facteur dépend de la structure phéromonale choisie), $\eta_S(i)$ est le facteur heuristique associé à i par rapport au début de combinaison S (la définition de ce facteur dépend du problème), et α et β sont deux paramètres permettant de moduler l'influence relative des deux facteurs dans la probabilité de transition. En particulier, si $\alpha = 0$ alors le facteur phéromonal n'intervient pas dans le choix des composants de combinaison et l'algorithme se comporte comme un algorithme glouton aléatoire pur. A l'inverse, si $\beta = 0$ alors seules les traces de phéromone sont prises en compte pour définir les probabilités de choix.

Exemple 31. Pour le problème du voyageur de commerce, le facteur phéromonal $\tau_S(i)$ est défini par la quantité de phéromone τ_{ki} déposée entre la dernière ville k ajoutée

dans S et la ville candidate i. Le facteur heuristique est inversement proportionnel à la longueur de la route joignant la dernière ville ajoutée dans S à la ville candidate i.

Mise à jour de la phéromone : Une fois que chaque fourmi a construit une combinaison, les traces de phéromone sont mises à jour. Elles sont tout d'abord diminuées en multipliant chaque trace par un facteur $(1-\rho)$, où $\rho \in [0;1]$ est le taux d'évaporation. Ensuite, certaines combinaisons sont « récompensées » par un dépôt de phéromone. Il existe différentes stratégies concernant le choix des combinaisons à récompenser. On peut récompenser toutes les combinaisons construites lors du dernier cycle, ou bien seulement les meilleures combinaisons du cycle, ou encore la meilleure combinaison trouvée depuis le début de l'exécution. Ces différentes stratégies influent sur l'intensification et la diversification de la recherche. En général, la phéromone est déposée en quantité proportionnelle à la qualité de la combinaison récompensée. Elle est déposée sur les traces de phéromone associées à la combinaison à récompenser ; ces traces dépendent de l'application et de la structure phéromonale choisie.

Exemple 32. Par exemple, pour le problème du voyageur de commerce, on dépose de la phéromone sur chaque trace τ_{ij} telle que les villes i et j ont été visitées consécutivement lors de la construction de la combinaison à récompenser.

11.4 Méta-heuristiques hybrides

Les différentes méta-heuristiques présentées en 11.2 et 11.3 peuvent être combinées pour former de nouvelles méta-heuristiques. Deux exemples classiques de telles hybridations sont décrits en 11.4.1 et 11.4.2.

11.4.1 Algorithmes mémétiques

Les algorithmes mémétiques combinent une méthode à population (algorithmes évolutionnaires) et la recherche locale [Moscato, 1999 ; Neri *et al.*, 2011]. Cette hybridation vise à tirer profit du potentiel de diversification fourni par l'approche à population et de la capacité d'intensification offerte par la recherche locale.

Un algorithme mémétique peut être considéré comme un algorithme génétique (tel que celui décrit dans l'algorithme 11.1) étendu par un processus de recherche locale. Tout comme pour un algorithme génétique, une population de combinaisons est utilisée pour échantillonner l'espace de recherche, et un opérateur de recombinaison permet de créer de nouvelles combinaisons à partir de deux ou plusieurs combinaisons de la population. Des mécanismes de sélection et de remplacement permettent de déterminer les combinaisons qui sont recombinées ainsi que celles qui sont éliminées de la population. Cependant, l'opérateur de mutation des algorithmes génétiques est remplacé par un processus de recherche locale, qui peut être considéré comme un processus guidé de macro-mutation. L'objectif de la recherche locale est d'améliorer la qualité des combinaisons générées. En comparaison avec l'opérateur de recombinaison, la recherche locale joue essentiellement le rôle de l'intensification de la recherche en exploitant les chemins

de recherche délimités par l'opérateur de voisinage considéré. Comme la recombinaison, la recherche locale est un autre élément clé de l'approche mémétique.

11.4.2 Hybridation entre approches perturbatives et approches constructives

Les approches perturbatives et constructives sont très facilement hybridables : à chaque itération, une ou plusieurs combinaisons sont construites selon un principe glouton aléatoire (qui peut être guidé par EDA ou ACO), puis certaines de ces combinaisons sont améliorées par une procédure de recherche locale. Cette hybridation est connue sous le nom de GRASP (*Greedy Randomized Adaptive Search Procedure*) [Resende et Ribeiro, 2003]. Un point important concerne le choix de l'opérateur de voisinage considéré et de la stratégie de choix des mouvements de l'approche perturbative. Il s'agit là de trouver un compromis entre le temps mis par la recherche locale pour améliorer les combinaisons, et la qualité des améliorations. Typiquement, on choisira d'effectuer une simple recherche locale gloutonne, améliorant les combinaisons construites jusqu'à arriver sur un optimum local.

Notons que les approches EDA et ACO les plus performantes comportent bien souvent une telle hybridation avec de la recherche locale.

11.5 Intensification *versus* diversification

Pour les différentes approches heuristiques présentées dans ce chapitre, un point critique dans la mise au point de l'algorithme consiste à trouver un juste compromis entre deux tendances duales :

— il s'agit d'une part d'intensifier l'effort de recherche vers les zones les plus « prometteuses » de l'espace des combinaisons, c'est-à-dire, aux alentours des meilleures combinaisons trouvées ;

— il s'agit par ailleurs de diversifier l'effort de recherche de façon à être capable de découvrir de nouvelles zones contenant (potentiellement) de meilleures combinaisons.

La façon d'intensifier/diversifier l'effort de recherche dépend de l'approche considérée et se fait en modifiant certains paramètres de l'algorithme. Pour les approches perturbatives, l'intensification de la recherche se fait en favorisant l'exploration des meilleurs voisins : pour les algorithmes génétiques, on adopte des stratégies de sélection et de remplacement élitistes qui favorisent la reproduction des meilleures combinaisons ; pour la recherche locale, on adopte des stratégies gloutonnes qui favorisent la sélection des meilleurs voisins. La diversification d'une approche perturbative se fait généralement en introduisant une part d'aléatoire : pour les algorithmes génétiques, la diversification est essentiellement assurée par la mutation ; pour la recherche locale, la diversification est assurée en autorisant avec une faible probabilité la recherche à choisir des voisins de moins bonne qualité.

Pour les approches constructives, l'intensification de la recherche se fait en favorisant, à chaque étape de la construction, le choix de composants ayant appartenu aux

meilleurs combinaisons précédemment construites. La diversification se fait en introduisant une part d'aléatoire permettant de choisir avec une faible probabilité de moins bons composants.

En général, plus on intensifie la recherche d'un algorithme en l'incitant à explorer les combinaisons proches des meilleures combinaisons trouvées, et plus il converge rapidement, trouvant de meilleures combinaisons plus tôt. Cependant, si l'on intensifie trop la recherche, l'algorithme risque fort de « stagner » autour d'*optima* locaux, concentrant tout son effort de recherche sur une petite zone autour d'une assez bonne combinaison, sans plus être capable de découvrir de nouvelles combinaisons. Notons ici que le bon équilibre entre intensification et diversification dépend clairement du temps de calcul dont on dispose pour résoudre le problème. Plus ce temps est petit et plus on a intérêt à favoriser l'intensification pour converger rapidement, quitte à converger vers des combinaisons de moins bonne qualité.

Le bon équilibre entre intensification et diversification dépend également de l'instance à résoudre, ou plus particulièrement de la topologie de son paysage de recherche. En particulier, une recherche fortement intensifiée donnera des résultats d'autant meilleurs que le paysage de recherche comporte peu d'*optima* locaux et qu'on observe une forte corrélation entre la qualité d'une combinaison et sa distance à la combinaison optimale ; on parle alors de paysages de recherche de type « massif central ». En revanche, quand le paysage de recherche comporte un très grand nombre d'*optima* locaux répartis uniformément, les meilleurs résultats seront obtenus par une recherche fortement diversifiée... pour ne pas dire une recherche complètement aléatoire !

Différentes approches ont proposé d'adapter dynamiquement le paramétrage au cours de la résolution, en fonction de l'instance à résoudre. On parle alors de recherche réactive [Battiti *et al.*, 2008]. Par exemple, l'approche taboue réactive de [Battiti et Protasi, 2001] ajuste automatiquement la longueur de la liste taboue, tandis que la recherche locale *IDwalk* de [Neveu *et al.*, 2004] adapte automatiquement le nombre de voisins considérés à chaque mouvement.

11.6 Applications en intelligence artificielle

Les méta-heuristiques ont été appliquées avec succès à un très grand nombre de problèmes classiques NP-difficiles et à des applications réelles dans des domaines extrêmement variés. Dans cette partie, nous évoquons leur application à deux problèmes centraux en intelligence artificielle : la satisfiabilité de formules booléennes (SAT) et la satisfaction des contraintes (CSP).

11.6.1 Satisfiabilité de formules booléennes

SAT est un des problèmes centraux en intelligence artificielle. Il s'agit, pour une formule booléenne, de déterminer un modèle, à savoir une affectation d'une valeur booléenne (vrai ou faux) à chaque variable telle que la valuation de la formule soit vraie. Pour des raisons pratiques, on suppose que la formule booléenne à traiter est donnée sous la forme CNF même si d'un point de vue général, la forme CNF n'est pas nécessaire pour appliquer une métaheuristique.

Remarquons que SAT n'est pas un problème d'optimisation et ne possède pas de fonction objectif à optimiser. Les méta-heuristiques étant généralement conçues pour résoudre des problèmes d'optimisation, elles considèrent un problème plus général appelé MAXSAT et dont l'objectif est de trouver la valuation maximisant le nombre de clauses satisfaites. Dans ce contexte, le résultat d'un algorithme méta-heuristique à une instance MAXSAT peut être de deux sortes : soit l'assignation retournée à la fin satisfait toutes les clauses de la formule, auquel cas une solution est trouvée à l'instance SAT, soit elle ne satisfait pas toutes les clauses, auquel cas on ne peut pas savoir si l'instance SAT est satisfiable ou non.

L'espace de recherche d'une instance SAT est défini par l'ensemble des affectations de valeurs booléennes à l'ensemble des variables. Ainsi, pour une instance ayant n variables, la taille de l'espace est 2^n. La fonction objectif à optimiser évalue le nombre de clauses satisfaites. Cette fonction introduit un ordre total sur les combinaisons de l'espace de recherche. On peut également considérer la fonction objectif duale, évaluant le nombre de clauses non satisfaites, et correspondant à une fonction de pénalité à minimiser : chaque clause non satisfaite a un poids de pénalité égal à 1, et une combinaison dont l'évaluation est 0 est une solution. Cette fonction peut être affinée par un mécanisme de pénalité dynamique ou encore une extension incluant d'autres informations que le nombre de clauses non satisfaites.

Beaucoup de travaux ont été réalisés depuis une vintaine d'années pour la résolution pratique de SAT. Les défis régulièrement organisés par la communauté scientifique autour de SAT dynamisent continuellement les activités de production, évaluation et comparaison d'algorithmes de résolution pour SAT. Il en résulte un très grand nombre de contributions qui améliorent chaque année la performance et la robustesse des algorithmes, notamment de recherche locale stochastique (RLS).

Recherche locale stochastique

Les algorithmes RLS dédiés à SAT considèrent généralement la même fonction de voisinage : deux combinaisons sont voisines si la distance de Hamming entre elles est exactement de 1. Le passage d'une combinaison à une autre est donc défini par le choix de la variable à modifier. Pour une formule à n variables, la taille du voisinage est égale à n. Ce choix peut être limité à l'ensemble des variables contenues dans au moins une clause falsifiée, donnant ainsi un voisinage plus restreint de taille variable.

Les algorithmes RLS dédiés à SAT diffèrent essentiellement dans les techniques mises en œuvre pour trouver un bon compromis entre d'une part une exploration de l'espace de recherche suffisamment large pour réduire le risque de stagner dans une région dépourvue de solution, et d'autre part l'exploitation des informations disponibles pour découvrir une solution dans la région en cours d'exploration. Pour classer ces différents algorithmes, nous pouvons considérer d'une part la manière dont les contraintes sont exploitées, d'autre part la manière dont l'historique de recherche est utilisé [Bailleux et Hao, 2008].

Exploitation de la structure de l'instance. La structure de l'instance à résoudre est définie par les clauses de la formule. On peut distinguer trois niveaux d'exploitation de cette structure.

La recherche peut être simplement *guidée par la fonction objectif*, de sorte que seul le nombre de clauses falsifiées par la combinaison courante est pris en compte. L'exemple typique est l'algorithme emblématique GSAT [Selman *et al.*, 1992] qui, à chaque itération, sélectionne aléatoirement une configuration voisine parmi celles minimisant le nombre de clauses falsifiées. Cette stratégie de descente gloutonne constitue une technique d'exploitation agressive qui peut être facilement piégée dans les optima locaux. Pour contourner ce problème, GSAT utilise la simple technique de diversification basée sur le redémarrage de la recherche à partir d'une nouvelle configuration initiale au bout d'un nombre fixe d'itérations. Plusieurs variantes de GSAT comme CSAT et TSAT [Gent et Walsh, 1993] ainsi que le recuit simulé [Spears, 1996] utilisent ce même principe de minimisation du nombre de clauses falsifiées.

La recherche peut être également *guidée par les conflits*, de sorte que les clauses falsifiées sont explicitement prises en compte. Cette approche est particulièrement utile quand le voisinage ne contient plus de combinaison améliorant la valeur d'objectif. Pour mieux orienter la recherche, on peux recourir à d'autres informations obtenues via, par exemple, une analyse des clauses falsifiées. L'archétype de ce type d'algorithme est WALKSAT [McAllester *et al.*, 1997] qui, à chaque itération, choisit aléatoirement une clause parmi celles falsifiées par la combinaison courante. Une heuristique est ensuite utilisée pour choisir une des variables de cette clause, dont la valeur sera modifiée pour obtenir la nouvelle combinaison courante. L'algorithme GWSAT (GSAT with random walk) [Selman et Kautz, 1994] utilise aussi une marche aléatoire guidée par les clauses falsifiées qui, à chaque itération, modifie une variable appartenant à une clause falsifiée avec une probabilité p (appelée bruit) et utilise la stratégie de descente de GSAT avec une probabilité $1 - p$.

Enfin, *l'exploitation déductive des contraintes* permet d'utiliser des règles de déduction pour modifier la combinaison courante. C'est le cas d'UNITWALK [Hirsch, 2004], qui utilise la résolution unitaire pour déterminer la valeur de certaines variables de la combinaison courante d'après les valeurs d'autres variables fixées par recherche locale. Un autre type d'approche déductive est utilisé dans un algorithme récent nommé NON-CNF ADAPT NOVELTY [Pham *et al.*, 2007], qui analyse la formule à traiter pour y rechercher des liens de dépendance entre variables.

Exploitation de l'historique de recherche. Pour classer les approches RLS dédiées à SAT, on peut également considérer la manière dont est exploité l'historique des actions réalisées depuis le début de la recherche et, le cas échéant, de leurs effets. On peut distinguer trois niveaux d'exploitation de cet historique.

Pour les *algorithmes Markoviens (ou sans mémoire)*, le choix de chaque nouvelle combinaison ne dépend que de la combinaison courante. Les algorithmes GSAT, GRSAT, WALKSAT/SKC, de même que l'algorithme plus général du recuit simulé sont des exemples typiques d'algorithmes RLS sans mémoire. S'ils ont l'avantage de minimiser le temps de traitement nécessaire à chaque itération, ils ont été supplantés en pratique, au cours de la dernière décennie, par les algorithmes à mémoire.

Pour les *algorithmes avec mémoire à court terme*, le choix de chaque nouvelle combinaison prend en compte un historique de tout ou partie des modifications des valeurs des variables. Les solveurs SAT inspirés de la méthode taboue, tels que WALKSAT/TABU

[McAllester *et al.*, 1997] ou TSAT [Mazure *et al.*, 1997] sont des exemples typiques. Certains algorithmes comme WALKSAT, NOVELTY et RNOVELY, et G2WSAT [Li et Huang, 2005] intègrent également l'*ancienneté* dans le critère de choix. Typiquement, ce critère est utilisé pour privilégier parmi plusieurs variables candidates celle qui a fait l'objet de la modification la plus ancienne.

Pour les *algorithmes avec mémoire à long terme*, le choix de chaque nouvelle combinaison dépend des choix réalisés depuis le début de la recherche et de leurs effets, notamment en terme de clauses satisfaites et falsifiées. L'idée est d'utiliser un mécanisme d'apprentissage pour tirer parti des échecs et accélérer la recherche. En pratique, l'historique des clauses falsifiées est exploité en associant des poids aux clauses, l'objectif étant de diversifier la recherche en l'obligeant à prendre de nouvelles directions. Un premier exemple est weighted-GSAT [Selman et Kautz, 1993] où, après chaque modification d'une variable, les poids des clauses falsifiées sont incrémentés. Le score utilisé par l'heuristique de recherche est simplement la somme des poids des clauses falsifiées. Comme le poids des clauses souvent falsifiées augmente, le processus tend à privilégier la modification des variables appartenant à ces clauses, jusqu'à ce que l'évolution des poids des autres clauses influence à nouveau la recherche. D'autres exemples comprennent notamment DLM (*discrete lagrangian method*) [Shang et Wah, 1998], DDWF (*Divide and Distribute Fixed Weight*) [Ishtaiwi *et al.*, 2005], PAWS (*Pure Additive Weighting Scheme*) [Thornton *et al.*, 2004], ESG (*Exponentiated Subgradient Algorithm*) [Schuurmans *et al.*, 2001], SAPS (*Scaling and Probabilistic Smoothing*) [Hutter *et al.*, 2002] et WV (*weighted variables*) [Prestwich, 2005].

Algorithmes à base de population

Les algorithmes génétiques ont été appliqués à plusieurs reprises à SAT [Jong et Spears, 1989 ; Young et Reel, 1990 ; Crawford et Auton, 1993 ; Hao et Dorne, 1994]. Comme les algorithmes de recherche locale, ces AG adoptent une représentation de l'espace de recherche sous forme d'affectation de valeurs 0/1 à l'ensemble de variables. Par contre, les premiers AG traitent directement des formules logiques générales, non limitées à la forme CNF. Malheureusement, les résultats obtenus par ces algorithmes sont généralement décevants quand ils sont appliqués à des benchmarks sous forme CNF.

Le premier AG traitant les formules CNF est présenté [Fleurent et Ferland, 1996]. L'originalité de cet algorithme est l'utilisation d'un croisement spécifique et original qui tente d'exploiter la sémantique des deux parents à croiser. Étant donnés deux parents p_1 et p_2, on examine les clauses qui sont satisfaites par un parent, mais falsifiées par l'autre parent. Lorsqu'une clause c est vraie chez le parent p_1 et fausse chez le parent p_2, les valeurs de vérité de toutes variables de cette clause sont directement transmises à l'enfant. L'algorithme mémétique intégrant ce croisement et une recherche taboue a donné des résultats très intéressants lors du deuxième challenge d'implantation DIMACS.

Un autre algorithme génétique plus récent est GASAT [Lardeux *et al.*, 2006]. Comme l'algorithme de [Fleurent et Ferland, 1996], GASAT accorde une importance prépondérante à la conception d'un croisement sémantique. Ainsi, une nouvelle classe de croisements est introduite visant à corriger les « erreurs » des parents et combiner leurs

bonnes caractéristiques. Par exemple, si une clause c est fausse chez deux parents de bonne qualité, on peut forcer cette clause à être vraie chez l'enfant par l'inversion d'une variable de c. L'argument intuitif qui justifie ce croisement consiste à considérer une telle clause comme difficilement satisfiable autrement et par conséquent préférer la rendre vraie par force. De même, des mécanismes de recombinaison sont développés pour exploiter la sémantique liée à une clause qui est rendue vraie simultanément par les deux parents. Muni de ce type de croisement et d'un algorithme de recherche taboue, GASAT se montre très compétitif sur certaines catégories de benchmarks.

FlipGA [Marchiori et Rossi, 1999] est un algorithme génétique qui repose sur un croisement standard (uniforme) qui génère, par un mélange équiprobable des valeurs des deux parents, un enfant auquel est appliqué un algorithme de recherche locale.

D'autres AG sont présentés dans [Eiben et van der Hauw, 1997 ; Gottlieb et Voss, 1998 ; Rossi *et al.*, 2000], mais il s'agit en réalité d'algorithmes de recherche locale puisque la population est réduite à une seule combinaison. Leur intérêt réside dans les techniques utilisées pour affiner la fonction d'évaluation de base (nombre de clauses falsifiées) par un ajustement dynamique durant le processus de recherche. On trouve dans [Gottlieb *et al.*, 2002] une comparaison expérimentale de ces quelques algorithmes génétiques dédiés à SAT. Leur intérêt pratique reste néanmoins à démontrer puisqu'ils se sont rarement confrontés directement aux algorithmes RLS de l'état de l'art.

On remarque que dans ces algorithmes à base de population, la recherche locale constitue souvent un élément indispensable dans le but de renforcer la capacité d'intensification. Le rôle du croisement peut être différent selon qu'il est entièrement aléatoire [Marchiori et Rossi, 1999] auquel cas il sert essentiellement à diversifier la recherche, ou qu'il est basé sur la sémantique du problème [Fleurent et Ferland, 1996 ; Lardeux *et al.*, 2006] auquel cas il permet à la fois de diversifier et d'intensifier la recherche.

11.6.2 Problèmes de satisfaction de contraintes

Un problème de satisfaction de contraintes (*Constraint Satisfaction Problem* ; CSP) est un problème modélisé sous la forme d'un ensemble de contraintes posées sur des variables, chacune de ces variables prenant ses valeurs dans un domaine. Résoudre un CSP consiste à affecter des valeurs aux variables, de telle sorte que toutes les contraintes soient satisfaites.

Comme pour SAT, on considère généralement le problème d'optimisation MaxCSP dont l'objectif est de trouver l'affectation complète (affectant une valeur à chaque variable) maximisant le nombre de contraintes satisfaites. L'espace de recherche est défini par l'ensemble des affectations complètes, tandis que la fonction objectif à maximiser est définie par le nombre de contraintes satisfaites.

Algorithmes génétiques

Dans sa forme la plus simple, un algorithme génétique pour la résolution de CSP utilise une population d'affectations complètes qui sont recombinées par simple croisement, la mutation consistant à changer la valeur d'une variable. [Craenen *et al.*, 2003] propose une comparaison expérimentale de onze algorithmes génétiques pour la résolution de CSP binaires quelconques. Ils ont montré que les trois meilleurs algorithmes

(*Heuristics GA version 3*, *Stepwise Adaptation of Weights* et *Glass-Box*) ont des performances équivalentes et sont significativement meilleurs que les huit autres algorithmes. Cependant, ces meilleurs algorithmes génétiques ne sont clairement pas compétitifs, ni avec les approches exactes basées sur une recherche arborescente, ni avec d'autres approches heuristiques, comme par exemple la recherche locale ou l'optimisation par colonies de fourmis [van Hemert et Solnon, 2004].

D'autres algorithmes génétiques ont été proposés pour des problèmes particuliers de satisfaction de contraintes. Ces algorithmes spécifiques exploitent des connaissances sur les contraintes du problème à résoudre pour définir des opérateurs de croisement et de mutation mieux adaptés, permettant d'obtenir de meilleurs résultats. On peut citer notamment [Zinflou *et al.*, 2007] qui obtient des résultats compétitifs pour un problème d'ordonnancement de voitures.

Recherche locale

Il existe de très nombreux travaux concernant la résolution de CSP par des techniques de recherche locale, et cette approche permet généralement d'obtenir d'excellents résultats. Les algorithmes de recherche locale pour la résolution de CSP diffèrent essentiellement par le voisinage et la stratégie de choix considérés.

Voisinage. En général, les combinaisons explorées sont des affectations totales, et un mouvement consiste à modifier la valeur d'une variable, de sorte que le voisinage d'une affectation totale est l'ensemble des affectations que l'on peut obtenir en changeant la valeur d'une variable. En fonction de la nature des contraintes, on peut considérer d'autres voisinages, comme par exemple le voisinage consistant à échanger les valeurs de deux variables lorsque ces variables sont impliquées dans une contrainte de permutation imposant qu'une suite de variables prenne pour valeur une permutation d'une suite donnée de valeurs.

Certains travaux considèrent des voisinages non pas uniquement entre affectations totales, mais également entre affectations partielles. En particulier, l'approche *decision repair* proposée dans [Jussien et Lhomme, 2002] combine des techniques de filtrage telles que celles décrites au chapitre II.6 avec une recherche locale sur l'espace des affectations partielles. Etant donnée une affectation partielle courante A, si le filtrage détecte une incohérence alors le voisinage de A est l'ensemble des affectations résultant de la suppression d'un couple variable/valeur de A, sinon le voisinage de A est l'ensemble des affectations résultant de l'ajout d'un couple variable/valeur à A.

Stratégies de choix. Un grand nombre de stratégies différentes pour le choix du prochain mouvement ont été proposées. La stratégie *min-conflict* [Minton *et al.*, 1992] consiste à choisir aléatoirement une variable impliquée dans au moins une violation de contrainte et à choisir pour cette variable la valeur qui minimise le nombre de violations. Cette stratégie gloutonne, célèbre pour avoir permis la résolution du problème des reines pour un million de reines, a cependant tendance à rester piégée dans des *optima* locaux. Une façon simple et classique pour permettre à la stratégie *min-conflict* de sortir de ces minima locaux consiste à la combiner avec la stratégie marche aléatoire [Wallace,

1996]. D'autres strategies pour le choix du couple variable/valeur sont étudiées dans [Hao et Dorne, 1996].

La recherche locale utilisant une stratégie taboue (TabuCSP) [Galinier et Hao, 1997] obtient d'excellents résultats pour la résolution de CSPs binaires. Partant d'une affectation initiale, l'idée est de choisir à chaque itération le mouvement non tabou qui augmente le plus le nombre de contraintes satisfaites. Un mouvement consiste à changer la valeur d'une *variable en conflit*, et il est dit tabou si le couple variable/valeur a déjà été choisi lors des k derniers mouvements (k étant la longueur de la liste taboue). Un critère d'aspiration est ajouté, permettant à un mouvement tabou d'être sélectionné s'il amène à une affectation satisfaisant plus de contraintes que la meilleure affectation trouvée depuis le début.

Le langage COMET [Hentenryck et Michel, 2005] permet de concevoir rapidement des algorithmes de recherche locale pour la résolution de CSP. Il introduit notamment pour cela le concept de variable incrémentale permettant une évaluation incrémentale des voisinages. D'autres systèmes génériques de résolutions de CSP fondés sur la recherche locale sont présentés dans [Davenport *et al.*, 1994 ; Nonobe et Ibaraki, 1998 ; Codognet et Diaz, 2001 ; Galinier et Hao, 2004].

Algorithmes de construction gloutonne

On peut construire une affectation totale pour un CSP selon le principe glouton suivant : partant d'une affectation vide, on sélectionne à chaque itération une variable non affectée et une valeur à affecter à cette variable, jusqu'à ce que toutes les variables soient affectées. Pour choisir à chaque étape une variable et une valeur, on peut utiliser les heuristiques d'ordre généralement utilisées par les approches exactes décrites au chapitre II.6. Un exemple bien connu d'instanciation de ce principe est l'algorithme DSATUR [Brélaz, 1979] pour résoudre le problème de coloriage de graphes. Cet algorithme construit une combinaison de façon gloutonne en choisissant à chaque itération le sommet non colorié ayant le plus grand nombre de voisins coloriés avec des couleurs différentes (le voisin le plus saturé). Les *ex aequo* sont départagés en choisissant le sommet de plus fort degré. Le sommet choisi est alors colorié par la plus petite couleur possible.

Optimisation par colonies de fourmis

La méta-heuristique ACO a été appliquée à la résolution de CSP dans [Solnon, 2002, 2008b]. L'idée est de construire des affectations complètes selon un principe glouton aléatoire : partant d'une affectation vide, on sélectionne à chaque itération une variable non affectée et une valeur à affecter à cette variable, jusqu'à ce que toutes les variables soient affectées. La contribution principale d'ACO est de fournir une heuristique de choix de valeur : celle-ci est choisie selon une probabilité qui dépend d'un facteur heuristique (inversement proportionnel au nombre de nouvelles violations introduites par la valeur), et d'un facteur phéromonal qui reflète l'expérience passée concernant l'utilisation de cette valeur. La structure phéromonale est générique et peut être utilisée pour résoudre n'importe quel CSP. Elle associe une trace phéromonale à chaque variable x_i et chaque valeur v_i pouvant être affectée à x_i : intuitivement, cette trace représente

l'expérience passée de la colonie concernant le fait d'affecter la variable x_i à la valeur v_i. D'autres structures phéromonales ont été proposées pour la résolution de CSP particuliers tels que, par exemple, les problèmes d'ordonnancement de voitures [Solnon, 2008a] ou de sac-à-dos multidimensionnels [Alaya *et al.*, 2007].

L'optimisation par colonies de fourmis a été intégrée dans les bibliothèques de résolution de CSP et de problèmes d'optimisation sous contraintes IBM/Ilog Solver et CP Optimizer [Khichane *et al.*, 2008, 2010]. Cette intégration permet d'utiliser des langages de haut niveau pour définir de façon déclarative le problème à résoudre, la résolution du problème étant prise en charge de façon automatique par un algorithme ACO intégré au langage.

11.7 Conclusion

Les méta-heuristiques ont été utilisées avec succès pour résoudre de nombreux problèmes d'optimisation difficiles, et ces approches obtiennent bien souvent de très bons résultats lors des compétitions, que ce soit pour la résolution de problèmes réels tels que ceux posés par la société française de Recherche Opérationnelle et d'Aide à la Décision (challenges ROADEF), ou pour la résolution de problèmes plus académiques tels que le problème SAT.

La variété des méta-heuristiques pose naturellement le problème du choix de celle qui sera la plus efficace pour résoudre un nouveau problème. Evidemment, cette question est complexe et se rapproche d'une quête fondamentale en intelligence artificielle, à savoir la résolution automatique de problèmes. Nous n'abordons dans cette discussion que quelques éléments de réponse.

En particulier, le choix d'une méta-heuristique dépend de la pertinence de ses mécanismes de base pour le problème considéré, c'est-à-dire, de leur capacité à générer de bonnes combinaisons :

- Pour les AG, l'opérateur de recombinaison doit être capable d'identifier les bons motifs qui doivent être assemblés et hérités par recombinaison. Par exemple, pour la coloration de graphes, un motif intéressant est un groupe de sommets de même couleur et partagé par de bonnes combinaisons ; ce type d'information a permis la conception de croisements très performants avec deux parents [Dorne et Hao, 1998 ; Galinier et Hao, 1999] ou avec plusieurs parents [Galinier *et al.*, 2008 ; Malaguti *et al.*, 2008 ; Lü et Hao, 2010 ; Porumbel *et al.*, 2010].
- Dans le cas de la recherche locale, le voisinage doit favoriser la construction de meilleurs combinaisons. Par exemple, pour les problèmes SAT et CSP, le voisinage centré sur les variables en conflit (voir Section 11.6.2) est pertinent car il favorise l'élimination des conflits.
- Pour ACO, la structure phéromonale doit être capable de guider la recherche vers de meilleures combinaisons. Par exemple, pour les CSP, la structure phéromonale associant une trace de phéromone à chaque couple variable/valeur est pertinente car elle permet d'apprendre progressivement quelles sont les bonnes valeurs à affecter à chaque variable.

Un autre point crucial réside dans la complexité en temps des opérateurs élémentaires de la méta-heuristique considérée (recombinaison et mutation pour les AG, mou-

vement pour la recherche locale, ajout d'un composant de combinaison pour les approches constructives, ...). Cette complexité dépend des structures de données utilisées pour représenter les combinaisons. Ainsi, le choix d'une méta-heuristique dépend de l'existence de structures de données qui permettent une évaluation incrémentale de la fonction d'évaluation après chaque application des opérateurs élémentaires de cette méta-heuristique.

D'autres éléments déterminants pour les performances sont partagés par toutes les méta-heuristiques. En particulier, la fonction d'évaluation (ne pas confondre avec la fonction objectif du problème) est un point clé car elle mesure la pertinence d'une combinaison. Par exemple, pour les CSP, la fonction d'évaluation peut simplement compter le nombre de contraintes non satisfaites mais, dans ce cas, l'importance relative des contraintes n'est pas distinguée. Un affinement intéressant consiste à introduire dans la fonction d'évaluation une pénalité pour quantifier le degré de violation d'une contrainte [Galinier et Hao, 2004]. Cette fonction peut être encore affinée, comme dans le cas du problème SAT (voir la section 11.6.1), par des pondérations qui s'ajustent dynamiquement en fonction de l'historique des violations de chaque contrainte.

Enfin, les méta-heuristiques ont bien souvent de nombreux paramètres qui ont une influence prépondérante sur leur efficacité. Ainsi, on peut voir le problème de la mise au point de toute méta-heuristique comme un problème de configuration : il s'agit de choisir les bonnes briques à combiner (opérateurs de recombinaison ou de mutation, voisinages, stratégies de choix des mouvements, facteurs heuristiques, ...) ainsi que le bon paramétrage (taux de mutation, d'évaporation, bruit, longueur de la liste taboue, poids des facteurs phéromonaux et heuristiques, ...). Une approche prometteuse pour résoudre ce problème de configuration consiste à utiliser des algorithmes de configuration automatiques pour concevoir l'algorithme le mieux adapté à un ensemble d'instances données [Xu *et al.*, 2010].

Références

AARTS, E. H. et KORST, J. H. (1989). *Simulated annealing and Boltzmann machines : a stochastic approach to combinatorial optimization and neural computing.* John Wiley & Sons, Chichester, U.K.

ALAYA, I., SOLNON, C. et GHEDIRA, K. (2007). Optimisation par colonies de fourmis pour le problème du sac-à-dos multi-dimensionnel. *Techniques et Sciences Informatiques (TSI)*, 26(3-4) :271–390.

ALDOUS, D. et VAZIRANI, U. V. (1994). "go with the winners" algorithms. *In 35th Annual Symposium on Foundations of Computer Science*, pages 492–501.

BAILLEUX, O. et HAO, J.-K. (2008). Algorithmes de recherche stochastiques. *In* SAÏS, L., éditeur : *Problème SAT : procès et défis*, chapitre 5. Hermès - Lavoisier.

BALUJA, S. (1994). Population-based incremental learning : A method for integrating genetic search based function optimization and competitive learning. Rapport technique, Carnegie Mellon University, Pittsburgh, PA, USA.

BATTITI, R., BRUNATO, M. et MASCIA, F. (2008). *Reactive Search and Intelligent Optimization.* Operations research/Computer Science Interfaces. Springer Verlag.

BATTITI, R. et PROTASI, M. (2001). Reactive local search for the maximum clique problem. *Algorithmica*, 29(4) :610–637.

BRÉLAZ, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22(4) :251–256.

CODOGNET, P. et DIAZ, D. (2001). Yet another local search method for constraint solving. *In International Symposium on Stochastic Algorithms : Foundations and Applications (SAGA)*, volume 2264 de *LNCS*, pages 70–93. Springer.

CRAENEN, B. G., EIBEN, A. et VAN HEMERT, J. I. (2003). Comparing evolutionary algorithms on binary constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 7(5) :424–444.

CRAWFORD, J. M. et AUTON, L. (1993). Experimental results on the cross-over point in satisfiability problems. *In Proceedings of the National Conference on Artificial Intelligence*, pages 22–28.

DAVENPORT, A., TSANG, E., WANG, C. J. et ZHU, K. (1994). Genet : A connectionist architecture for solving constraint satisfaction problems by iterative improvement. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 325–330. AAAI.

DORNE, R. et HAO, J.-K. (1998). A new genetic local search algorithm for graph coloring. *In 5th International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 1498 de *LNCS*, pages 745–754. Springer.

EIBEN, A. et van der HAUW, J. (1997). Solving 3-sat with adaptive genetic algorithms. *In Proceedings of the Fourth IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE Press.

EIBEN, A. E. et SMITH, J. E. (2003). *Introduction to Evolutionary Computing*. Springer.

FEO, T. A. et RESENDE, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letter*, 8 :67–71.

FLEURENT, C. et FERLAND, J. A. (1996). Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26 :619–652.

GALINIER, P. et HAO, J.-K. (1997). Tabu search for maximal constraint satisfaction problems. *In International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1330 de *LNCS*, pages 196–208. Springer.

GALINIER, P. et HAO, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4) :379–397.

GALINIER, P. et HAO, J.-K. (2004). A general approach for constraint solving by local search. *Journal of Mathematical Modelling and Algorithms*, 3(1) :73–88.

GALINIER, P., HERTZ, A. et ZUFFEREY, N. (2008). An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2) :267–279.

GENT, I. P. et WALSH, T. (1993). Towards an understanding of hill-climbing procedures for sat. *In Proceedings of AAAI-93*.

GLOVER, F. et LAGUNA, M. (1993). Tabu search. *In* REEVES, C., éditeur : *Modern Heuristics Techniques for Combinatorial Problems*, pages 70–141. Blackwell Scientific Publishing, Oxford, UK.

GOLDBERG, D. E. (1989). *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley.

GOTTLIEB, J., MARCHIORI, E. et ROSSI, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10 :35–50.

GOTTLIEB, J. et VOSS, N. (1998). Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. *In* IN EIBEN, A. e. a., éditeur : *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 de *LNCS*, pages 755–764. Springer.

HAO, J.-K. et DORNE, R. (1994). An empirical comparison of two evolutionary methods for satisfiability problems. *In Proc. of IEEE Intl. Conf. on Evolutionary Computation*, pages 450–455. IEEE Press.

HAO, J.-K. et DORNE, R. (1996). Empirical studies of heuristic local search for constraint solving. *In International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1118 de *LNCS*, pages 194–208. Springer.

HENTENRYCK, P. V. et MICHEL, L. (2005). *Constraint-based local search.* MIT Press.

HIRSCH, E. A. (2004). Unitwalk : A new sat solver that uses local search guided by unit clause elimination. *In Proceedings of SAT 2002.*

HOLLAND, J. H. (1975). *Adaptation and artificial systems.* University of Michigan Press.

HOOS, H. H. et STÜTZLE, T. (2005). *Stochastic local search, foundations and applications.* Morgan Kaufmann.

HUTTER, F., TOMPKINS, D. A. D. et HOOS, H. H. (2002). Scaling and probabilistic smoothing : Efficient dynamic local search for sat. *In Proceedings of CP 2002, Principles and Practice of Constraints Programming*, volume 2470 de *LNCS*, pages 233–248. Springer Verlag.

ISHTAIWI, A., THORNTON, J., SATTAR, A. et PHAM, D. N. (2005). Neighbourhood clause weight redistribution in local search for sat. *In Proceedings of CP 2005*, pages 772–776.

JAGOTA, A. et SANCHIS, L. A. (2001). Adaptive, restart, randomized greedy heuristics for maximum clique. *Journal of Heuristics*, 7(6) :565–585.

JONG, K. D. et SPEARS, W. (1989). Using genetic algorithms to solve np-complete problems. *In Intl. Conf. on Genetic Algorithms (ICGA'89)*, pages 124–132, Fairfax, Virginia.

JUSSIEN, N. et LHOMME, O. (2002). Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139 :21–49.

KHICHANE, M., ALBERT, P. et SOLNON, C. (2008). Integration of ACO in a constraint programming language. *In 6th International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS'08)*, volume 5217 de *LNCS*, pages 84–95. Springer.

KHICHANE, M., ALBERT, P. et SOLNON, C. (2010). Strong integration of ant colony optimization with constraint programming optimization. *In 7th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR)*, volume 6140 de *LNCS*, pages 232–245. Springer.

LARDEUX, F., SAUBION, F. et HAO, J. (2006). Gasat : a genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2) :223–253.

LARRANAGA, P. et LOZANO, J. A. (2001). *Estimation of Distribution Algorithms. A new tool for Evolutionary Computation.* Kluwer Academic Publishers.

LI, C. et HUANG, W. (2005). Diversification and determinism in local search for satisfiability. *In Proceedings of SAT 2005*, volume 3569 de *LNCS*, pages 158–172. Springer.

LOURENCO, H. R., MARTIN, O. et STUETZLE, T. (2002). *Handbook of Metaheuristics*, chapitre Iterated Local Search, pages 321–353. Kluwer Academic Publishers.

LÜ, Z. et HAO, J.-K. (2010). A memetic algorithm for graph coloring. *European Journal of Operational Research*, 200(1) :235–244.

MALAGUTI, E., MONACI, M. et TOTH, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2) :302–316.

MARCHIORI, E. et ROSSI, C. (1999). A flipping genetic algorithm for hard 3-sat problems. *In Proc. of the Genetic and Evolutionary Computation Conference*, volume 1, pages 393–400.

MAZURE, B., SAÏS, L. et GRÉGOIRE, E. (1997). Tabu search for SAT. *In Proc. of the AAAI-97.*

MCALLESTER, D., SELMAN, B. et KAUTZ, H. (1997). Evidence for invariants in local search. *In Proc. of the AAAI 97.*

MINTON, S., JOHNSTON, M. D., PHILIPS, A. B. et LAIRD, P. (1992). Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58 :161–205.

MLADENOVIĆ, N. et HANSEN, P. (1997). Variable neighborhood search. *Comps. in Oerations Research*, 24 :1097–1100.

MOSCATO, P. (1999). Memetic algorithms : a short introduction. *In* CORNE, D., DORIGO, M. et GLOVER, F., éditeurs : *New Ideas in Optimization*, pages 219–234. McGraw-Hill Ltd., Maidenhead, UK.

NERI, F., COTTA, C. et MOSCATO, P. (2011). Handbook of Memetic Algorithms. Springer.

NEVEU, B., TROMBETTONI, G. et GLOVER, F. (2004). Id walk : A candidate list strategy with a simple diversification device. *In International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3258 de *LNCS*, pages 423–437. Springer Verlag.

NONOBE, K. et IBARAKI, T. (1998). A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, 106 :599–623.

PELIKAN, M., GOLDBERG, D. E. et CANTÚ-PAZ, E. (1999). BOA : The Bayesian optimization algorithm. *In Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532. Morgan Kaufmann Publishers, San Fransisco, CA.

PHAM, D. N., THORNTON, J. et SATTAR, A. (2007). Building structure into local search for sat. *In Proceedings of IJCAI*, pages 2359–2364.

PORUMBEL, D. C., HAO, J.-K. et KUNTZ, P. (2010). An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10) :1822–1832.

PRESTWICH, S. (2005). Random walk with continuously smoothed variable weights. *In Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*.

RESENDE, M. G. et RIBEIRO, C. C. (2003). *Handbook of Metaheuristics*, chapitre Greedy randomized adaptive search procedures, pages 219–249. Kluwer Academic Publishers.

ROSSI, C., MARCHIORI, E. et KOK, J. (2000). An adaptive evolutionary algorithm for the satisfiability problem. *In* CARROLL, J. e. a., éditeur : *Proceedings of ACM Symposium on Applied Computing*, pages 463–469. ACM, New York.

SCHUURMANS, D., SOUTHEY, F. et HOLTE, R. (2001). The exponential subgradient algorithm for heuristic boolean programming. *In Proceedings of AAAI 2001*, pages 334–341.

SELMAN, B. et KAUTZ, H. (1993). Domain-independent extensions to gsat : Solving large structured satisfiability problems. *In Proceedings of IJCAI-93*, pages 290–295.

SELMAN, B. et KAUTZ, H. (1994). Noise strategies for improving local search. *In Proc. of the AAAI 94*, pages 337–343.

SELMAN, B., KAUTZ, H. A. et COHEN, B. (1994). Noise strategies for improving local search. *In Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343. AAAI Press / The MIT Press, Menlo Park, CA, USA.

SELMAN, B., LEVESQUE, H. et MITCHELL, D. (1992). A new method for solving hard satisfiability problems. *In 10th National Conference on Artificial Intelligence (AAAI)*, pages 440–446.

SHANG, Y. et WAH, B. W. (1998). A discrete lagragian based global search method for solving satisfiability problems. *Journal of Global Optimization*, 12 :61–100.

SOLNON, C. (2002). Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4) :347–357.

SOLNON, C. (2008a). Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. *European Journal of Operational Research (EJOR)*, 191 :1043–1055.

SOLNON, C. (2008b). *Optimisation par colonies de fourmis - Collection Programmation par contraintes*. Hermès-Lavoisier.

SPEARS, W. M. (1996). Simulated annealing for hard satisfiability problems. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26 :533–558.

THORNTON, J., PHAM, D. N., BAIN, S. et FERREIRA, V. J. (2004). Additive versus multiplicative clause weighting for sat. *In Proceeding of AAAI 2004*, pages 191–196.

van HEMERT, J. I. et SOLNON, C. (2004). A study into ant colony optimization, evolutionary computation and constraint programming on binary constraint satisfaction problems. *In Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004)*, volume 3004 de *LNCS*, pages 114–123. Springer-Verlag.

WALLACE, R. J. (1996). Analysis of heuristics methods for partial constraint satisfac-

tion problems. *In International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1118 de *LNCS*, pages 308–322. Springer.

XU, L., HOOS, H. H. et LEYTON-BROWN, K. (2010). Hydra : Automatically configuring algorithms for portfolio-based selection. *In 24th AAAI Conference on Artificial Intelligence*, pages 210–216.

YOUNG, R. et REEL, A. (1990). A hybrid genetic algorithm for a logic problem. *In Proc. of the 9th European Conf. on Artificial Intelligence*, pages 744–746, Stockholm, Sweden.

ZINFLOU, A., GAGNÉ, C. et GRAVEL, M. (2007). Crossover operators for the car sequencing problem. *In 7th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)*, volume 4446 de *LNCS*, pages 229–239.

ZLOCHIN, M., BIRATTARI, M., MEULEAU, N. et DORIGO, M. (2004). Model-based search for combinatorial optimization : A critical survey. *Annals of Operations Research*, 131 :373–395.

Postface

Intelligence artificielle et recherche opérationnelle

Recherche opérationnelle

Hormis les travaux de quelques précurseurs célèbres comme Blaise Pascal pour ses *Réflexions sur quelques problèmes de décision dans l'incertain*, Gaspard Monge pour la *Résolution du problème des déblais et remblais* ou Harris pour sa *formule du lot économique* en gestion de stock, la recherche opérationnelle (RO) « moderne » commence en 1940, avec la seconde guerre mondiale, où Patrick Blackett est chargé par l'état major anglais de former la première équipe de recherche opérationnelle pour résoudre un problème de *localisation optimale* de radars de surveillance et des problèmes liés à la *gestion des approvisionnements.* L'emploi du terme « opérationnelle » est donc lié aux opérations militaires, qui ont constitué le premier champ d'application de la recherche opérationnelle. Depuis, et notamment en raison de l'explosion des capacités de calcul des ordinateurs, les techniques de résolution se sont considérablement développées et les champs d'applications se sont multipliés : systèmes administratifs (emplois du temps,...), ateliers de production (planification, ordonnancement,...), systèmes physiques (verres de spin,...), systèmes de transport (tournées de distribution,...), systèmes informatiques (localisation de fichiers, optimisation de code,...), systèmes biologiques (alignement de séquences,...), télécommunications (dimensionnement de réseaux,...), énergie (gestion des barrages hydrauliques, arrêt des centrales nucléaires, distribution,...).

Ces quelques exemples, quoique très limités, sont cependant suffisants pour dégager la nature et l'approche des problèmes auxquels cette discipline est confrontée. La problématique générale peut se résumer comme suit : à partir d'un problème applicatif, il s'agit d'élaborer un modèle mathématique de problème d'optimisation (souvent le résultat d'un consensus entre le demandeur et le chercheur), de développer une méthode de résolution (algorithme) exacte ou approchée, et d'évaluer la qualité des solutions produites dans l'environnement réel du problème.

On peut, à grands traits, catégoriser les modèles de la recherche opérationnelle en 3 classes : les *modèles combinatoires* (optimisation dans les graphes, ordonnancement, logistique, transport, localisation, ...), les *modèles stochastiques* (systèmes d'attente, stratégies dans l'incertain, ,...) et les *modèles d'aide à la décision* (jeux concurrentiels,

Auteur : PHILIPPE CHRÉTIENNE.

modèles de préférence,...).
Bien entendu, cette définition de la recherche opérationnelle peut se décliner selon plusieurs variantes car certains chercheurs opérationnels travaillent plutôt sur l'amélioration des techniques et d'autres sont plus proches des applications. En tout état de cause, la recherche opérationnelle est une discipline *naturellement transversale*, qui utilise des outils mathématique et informatique pour résoudre des problèmes d'optimisation à partir de modèles issus du terrain applicatif.

Intelligence artificielle

L'intelligence artificielle (IA) est une discipline, enfantée par l'informatique et donc plus récente que la recherche opérationnelle. Sa vocation générale est, selon ses créateurs, la construction de programmes informatiques capables d'exécuter des tâches accomplies jusqu'ici de manière plus satisfaisante par des êtres humains car demandant des processus mentaux de haut niveau (apprentissage, organisation, raisonnement,...). Alors que le terme « intelligence » est relatif à l'*imitation du comportement humain* (raisonnement, jeux mathématiques, compréhension, perceptions visuelle et auditive,...), le terme « artificielle » résulte de l'usage des ordinateurs. Notons cependant que cette définition est considérée comme restrictive par les tenants d'une intelligence artificielle « forte », qui envisagent la possibilité de créer des machines capables non seulement de produire des comportements intelligents mais aussi douées d'une « conscience intelligente » permettant la compréhension de ses propres raisonnements. On constate en tout état de cause que l'ambition de l'intelligence artificielle est beaucoup plus large que celle de la recherche opérationnelle. Mais il est également clair qu'à travers la résolution de problèmes d'optimisation difficiles (par exemple de nature combinatoire ou d'aide à la décision) issus de modèles suffisamment génériques, les deux disciplines peuvent se *renforcer l'une l'autre* comme j'espère le montrer sur les exemples développés dans le paragraphe suivant.

RO et IA

Mon premier exemple concerne le *raisonnement par contraintes* qui, côté intelligence artificielle, est vu comme un modèle *générique* de résolution de problèmes. Le caractère générique de l'approche, qui en constitue bien sûr un atout indéniable, peut aussi être un facteur d'inefficacité car les *spécificités* d'un problème donné, par exemple les propriétés structurelles fines de ses solutions optimales, qui (par définition) ne sont pas prises en compte dans un logiciel générique de programmation par contraintes, sont le plus souvent indispensables à la résolution (exacte ou approchée) des instances réelles. Dans beaucoup de cas, seule l'analyse approfondie des propriétés spécifiques du problème permettra de casser suffisamment la combinatoire de l'espace des solutions.
Le développement de *techniques de propagation adaptées* est cependant une réponse qui a été la source de progrès importants dans la résolution de certains problèmes combinatoires difficiles.
Considérons par exemple, dans le cadre des problèmes d'ordonnancement non préemp-

tifs, la *contrainte monomachine* où chaque tâche i de durée p_i, de date de disponibilité r_i et de deadline d_i, requiert pour son exécution une machine M disponible en seul exemplaire. Toutes les données sont des nombres entiers et nous notons respectivement δ_i la valeur $r_i + p_i$ qui minore la date de fin au plus tôt de la tâche i et ρ_i la valeur $d_i - p_i$ qui majore sa date de début au plus tard. La date de début (respectivement fin) de la tâche i est la variable S_i (respectivement C_i).
Nous présentons alors deux règles simples : « emploi du temps » et « disjonction » et deux règles plus sophistiquées : « *edge-finding* » et « *not-first, not last* » qui, introduites dans un logiciel de programmation par contraintes, ont été à la base de progrès très importants dans la résolution de problèmes d'ateliers, en particulier de type *job shop*.
La règle « emploi du temps » exploite simplement l'utilisation ou non de la machine pendant une unité de temps t, information capturée par la variable $a_i(t)$ qui vaut 1 si la tâche i utilise M pendant l'unité de temps t et 0 sinon. La propagation de cette contrainte réalise alors les ajustements suivants :

$(a_i(t) = 0) \wedge (\delta_i \geq t)$	$\Rightarrow$	$S_i \geq t$
$(a_i(t) = 0) \wedge (\rho_i < t)$	$\Rightarrow$	$C_i < t$
$a_i(t) = 1$	$\Rightarrow$	$(S_i \geq t - p_i) \wedge (C_i < t + p_i)$

La règle « disjonction » exploite la non-simultanéité de l'exécution de toute paire $\{i, j\}$ de tâches. Sa propagation réalise l'ajustement suivant :

$$\delta_i > \rho_j \quad \Rightarrow \quad S_i \geq S_j + p_j$$

Le « *edge-finding* » est un processus déductif puissant qui permet d'obtenir des relations sur l'ordre de passage des tâches sur la machine. Il peut être utilisé soit pour guider les branchements en vue de sélectionner une tâche i dans un ensemble Ω, soit pour supprimer des nœuds dans l'arbre de recherche en déduisant qu'une tâche $i \notin \Omega$ doit être exécutée avant, après ou ni avant ni après celles de Ω. Si nous notons respectivement r_Ω et d_Ω la plus petite date de début au plus tôt et la plus grande date de fin au plus tard des tâches de Ω, les ajustements du « *edge-finding* » sont les suivants :

$(i \notin \Omega) \wedge (r_\Omega + p_\Omega + p_i > d_\Omega)$	$\Rightarrow$	$i \prec \Omega$
$(i \notin \Omega) \wedge (r_{\Omega \cup \{i\}} + p_\Omega + p_i > d_\Omega)$	$\Rightarrow$	$\Omega \prec i$
$(i \notin \Omega) \wedge (i \prec \Omega)$	$\Rightarrow$	$C_i \leq \min_{\emptyset \neq \Omega' \subset \Omega}(d_{\Omega'} - p_{\Omega'})$
$(i \notin \Omega) \wedge (\Omega \prec i)$	$\Rightarrow$	$S_i \leq \max_{\emptyset \neq \Omega' \subset \Omega}(r_{\Omega'} + p_{\Omega'})$

Au contraire du « *edge-finding* » qui permet de déduire qu'une tâche i doit être exécutée avant ou après celles d'un ensemble Ω, la règle « *not-first, not last* » indique que la tâche i ne peut être exécutée ni avant ni après les tâche de Ω. Les ajustements associés se résument comme suit :

$(i \notin \Omega) \wedge (r_i + p_\Omega + p_i > d_\Omega)$	$\Rightarrow$	$non(i \prec \Omega)$
$(i \notin \Omega) \wedge (r_\Omega + p_\Omega + p_i > d_i)$	$\Rightarrow$	$non(\Omega \prec i)$
$(i \notin \Omega) \wedge non(i \prec \Omega)$	$\Rightarrow$	$S_i \geq \min_\Omega(r_j - p_j)$
$(i \notin \Omega) \wedge non(\Omega \prec i)$	$\Rightarrow$	$C_i \leq \max_\Omega(d_j - p_j)$

Ces règles se sont révélées très efficaces dans la résolution des problèmes d'ordonnancement car, tant pour le « *edge-finding* » que pour le « *not-first, not last* », les ajustements associés aux $\Omega(n2^n)$ paires $\{i, \Omega\}$ peuvent être calculés en temps $O(n^2)$. Ces résultats importants [Baptiste et Le Pape, 2000] dont la source réside dans des travaux de recherche sur la résolution exacte du problème de *job shop* [Carlier et Pinson, 1989], montrent bien, à mon sens, l'interaction positive entre recherche opérationnelle et intelligence artificielle.

Mon second exemple concerne le domaine des *métaheuristiques* et plus précisément celui de la *recherche locale*, qui constitue souvent l'approche la plus efficace pour la résolution de problèmes combinatoires difficiles. Rappelons qu'un algorithme de recherche locale est fondé sur la définition d'une *fonction voisinage* qui associe à chaque solution x l'ensemble $N(x)$ de ses voisins. A partir d'une solution initiale x_0, l'itération i consiste à chercher dans $N(x_{i-1})$ une solution x_i strictement meilleure que x_{i-1} (éventuellement en cherchant la meilleure solution de $N(x_{i-1})$). Si une telle solution n'existe pas, x_{i-1} est un optimum local vis-à-vis de N et l'algorithme se termine. L'efficacité de ce type d'algorithme est très liée au choix de N. En particulier la *taille* de $N(x)$ est un paramètre très sensible. En effet, une taille plus grande conduit généralement à moins d'itérations et à un optimum local de meilleure qualité au prix cependant d'une augmentation du temps de calcul de chaque itération. Il faut donc trouver un voisinage de grande taille (éventuellement exponentielle) duquel on puisse extraire rapidement (par exemple en temps polynomial faible) une solution de bonne qualité. C'est pourquoi s'est développée assez récemment en recherche opérationnelle une recherche active dans le domaine des « *grands voisinages* ».
Nous allons illustrer l'intérêt des « grands voisinages » sur un problème assez général de *partitionnement* dont la problématique est la suivante : étant donné un ensemble $A = \{a_1, \cdots, a_n\}$, il faut déterminer une partition $S = (S_1, \cdots, S_K)$ de A en classes (qui peuvent posséder une structure particulière) de telle sorte que la somme $\sum_{k=1}^{K} c_k(S_k)$ soit minimum. Notons que la valeur $c_k(S_k)$, qui est le coût de regroupement des éléments de S_k dans une classe, peut éventuellement être difficile à calculer. Par exemple si l'on a choisi les sommets d'une tournée, le calcul du coût de cette tournée peut s'apparenter à un problème de voyageur de commerce. Ce problème de partitionnement recouvre en particulier les problèmes de *tournées de véhicules*, le problème de l'*arbre couvrant minimum avec capacité*, le problème d'*affectation généralisée*, les problèmes de *multicoupes*, d'*agrégation* et de *partitionnement de graphes*, des problèmes de *localisation* et certains problèmes d'*ordonnancement sur machines parallèles*.
Les premiers voisinages étudiés ont été les *2-échanges* [Croes, 1958] où les $\Omega(n^2)$ voisins d'une partition S résultent de l'échange de deux éléments de A appartenant à deux classes distinctes de S. Plus récemment, Thompson et Orlin [Thomson et Orlin, 1989] ont proposé l'*échange cyclique*, généralisation du 2-échange, qui consiste à transférer cycliquement au plus K éléments pris dans des classes distinctes de S (Figure 1).

La taille $\Omega(n^K)$ du voisinage associé à l'échange cyclique est très supérieure à celle du 2-échange et peut être exponentielle en n si K dépend de n. Il est donc rédhibitoire

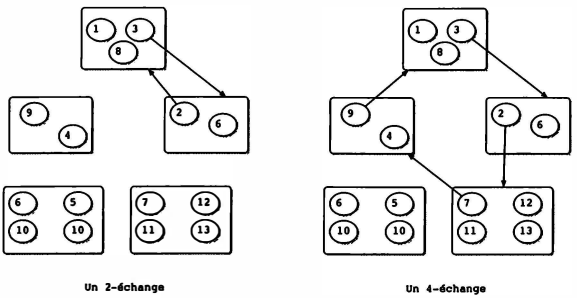


FIGURE 1 – Un 2-échange et un 4-échange

d'envisager d'énumérer tous les voisins d'une partition pour en trouver une meilleure. Un *algorithme d'optimisation* est donc nécessaire. Celui conçu par [Ahuja *et al.*, 2000] s'appuie sur un graphe valué noté $G(S)$, appelé *graphe des améliorations*, dont les sommets sont les n éléments de A et pour lequel il existe un arc (i, j) si a_i et a_j appartiennent à deux classes distinctes et si la classe obtenue à partir de celle contenant a_j en remplaçant a_j par a_i est réalisable. Le coût de l'arc (i, j) est alors la différence entre le coût de la nouvelle classe contenant a_i et celui de l'ancienne classe contenant a_j. Un circuit de $G(S)$ est dit *classes-disjoint* si les sommets empruntés appartiennent à des classes disjointes de S. Comme les échanges cycliques et les circuits classes-disjoints se correspondent dans une bijection conservant les coûts, il faut identifier dans $G(S)$ les circuits classes-disjoints négatifs, appelés *circuits valides.*
Tester l'existence d'un circuit valide étant un problème NP-complet [Thompson et Psaraftis, 1993], plusieurs heuristiques et des algorithmes d'optimisation non polynomiaux ont été développés pour trouver un circuit valide. Une variante simple d'un algorithme de recherche de plus court chemin avec correction d'étiquette [Ahuja *et al.*, 1993] s'est avérée très efficace au point que le temps de traitement d'un voisinage d'échange cyclique est quasiment équivalent à celui du 2-échange.
Un tel algorithme de recherche locale, basé les voisinages des échanges cycliques, a été appliqué [Ahuja *et al.*, 2000] en particulier au problème de l'arbre couvrant de coût minimum avec capacité. Dans ce problème, il faut, à partir d'un nœud processeur, acheminer à moindre coût des flux vers les nœuds demandeurs d'un réseau en respectant une capacité maximale K pour les arcs du réseau global (nœuds demandeurs et nœud processeur). Le graphe des liaisons possibles est supposé complet et le coût de chaque liaison est connu. Une solution de ce problème est une *partition des nœuds demandeurs* en K classes telles que la somme des demandes de chaque classe est au plus K. Le coût d'une classe est celui d'un arbre couvrant de coût minimum du sous-réseau associé à cette classe. La figure 2 montre la structure d'une solution comprenant trois arbres couvrants, des demandes unitaires et une capacité $K = 6$.

L'algorithme précédent a permis d'améliorer très sensiblement (jusqu'à 20%) les meilleures solutions connues (obtenues par des méthodes tabou) pour des instances « *benchmark* » et de résoudre des problèmes de 500 noeuds et des demandes non

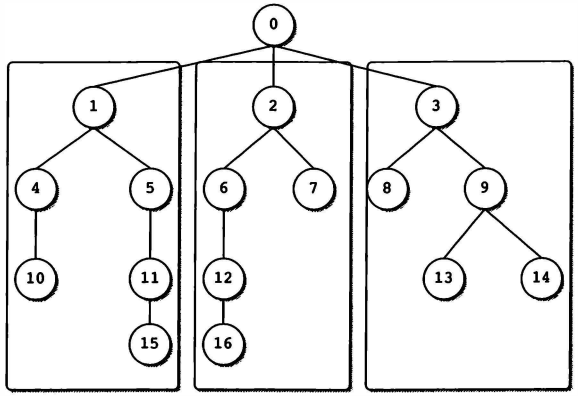


FIGURE 2 – Structure d'une solution pour l'arbre couvrant minimum avec capacité unitaires.

Conclusion

Après une présentation assez schématique des problématiques des la recherche opérationnelle et de l'intelligence artificielle, nous avons souhaité montrer, à travers deux exemples empruntés aux domaines des contraintes et des métaheuristiques, comment ces deux disciplines peuvent se renforcer l'une l'autre pour mieux résoudre les nombreux problèmes d'optimisation issus du terrain applicatif et qui restent encore souvent hors de portée malgré l'amélioration des techniques et l'accroissement de la puissance de calcul. Il est certain que la programmation par contraintes a apporté, par sa vision unificatrice des problèmes et ses techniques génériques de résolution, une amélioration très sensible du confort de programmation et se sont avérées suffisamment efficaces pour résoudre un grand nombre de problèmes d'origine applicative. La généricité de ces méthodes leur confère également une souplesse d'adaptation et une robustesse face à d'éventuelles variations dans les spécifications du problème. Il n'en reste pas moins que des problèmes hautement combinatoires sont encore mal résolus. L'apport de la recherche opérationnelle est alors d'extraire autant que possible le noyau dur de la complexité du problème à partir d'une analyse approfondie des propriétés de ses solutions optimales et d'utiliser au mieux les résultats (dominances, bornes, ...) pour les intégrer dans des méthodes énumératives intelligemment guidées. En conclusion, nous pouvons donc affirmer que les approches IA et RO se complètent pour converger vers l'objectif commun de progresser dans la résolution des problèmes difficiles.

Références

AHUJA, R., MAGNANTI, T. et ORLIN, J. (1993). *Network Flows : Theory, Algorithms, and Applications.* Prentice Hall (NJ).

AHUJA, R., ORLIN, J. et SHARMA, D. (2000). New neighborhood search structures for the capacitated minimum spanning tree problem. Rapport technique, Sloan School of management, MIT, Cambridge (MA).

BAPTISTE, P. et LE PAPE, C. (2000). Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5 :119–139.

CARLIER, J. et PINSON, E. (1989). An algorithm for solving the job shop problem. *Management Science*, 35(2) :164–176.

CROES, G. (1958). A method for solving traveling salesman problems. *Operations Research*, 6 :791–812.

THOMPSON, P. et PSARAFSTIS, H. (1993). Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41 :935–946.

THOMSON, P. et ORLIN, J. (1989). The theory of cyclic transfers. Rapport technique OR-200-89, Operations Research Center, MIT, Cambridge (MA).

Index

Table des matières

Si cet ouvrage fait l'objet d'un additif ou d'un erratum, l'information sera alors disponible sur le site internet **www.cepadues.com**, à la fiche de l'ouvrage. L'accès direct se fait en tapant la référence de l'ouvrage (1042) dans le champ recherche.

Vous pouvez faire part de vos remarques,
critiques, suggestions
aux auteurs à cette adresse :

auteurs@cepadues.com

PANORAMA DE L'INTELLIGENCE ARTIFICIELLE

Cet ouvrage, organisé en trois volumes, est issu de la communauté française des chercheurs en intelligence artificielle (IA). Il a pour objectif de dresser un panorama des recherches effectuées en IA allant de travaux fondamentaux aux applications et aux frontières, en mettant l'accent tout autant sur les résultats obtenus que sur les problématiques actuelles.

Il s'adresse à un public d'étudiants de master et de doctorat, mais aussi de chercheurs et d'ingénieurs intéressés par ce domaine.

L'ouvrage est organisé en trois volumes :

* le premier volume regroupe vingt chapitres traitant des fondements de la représentation des connaissances et de la formalisation des raisonnements (**Volume 1. Représentation des connaissances et formalisation des raisonnements**) ;

* le deuxième volume offre une vue de l'IA, en onze chapitres, sous l'angle des algorithmes (**Volume 2. Algorithmes pour l'intelligence artificielle**) ;

* le troisième volume, en onze chapitres également, décrit les principales frontières et applications de l'IA (**Volume 3. L'intelligence artificielle : frontières et applications**).

Si chaque chapitre peut être lu indépendamment des autres, les références croisées entre chapitres sont nombreuses et un index global de l'ouvrage permet d'aborder celui-ci de façon non linéaire.

Volume 2. Algorithmes pour l'intelligence artificielle

Ce deuxième volume présente les principales familles d'algorithmes développés ou utilisés en IA pour apprendre, inférer, décider. Des approches génériques pour la résolution de problèmes y sont présentées : la recherche heuristique ordonnée, particulièrement utile pour aborder certains jeux, ainsi que les métaheuristiques. Les problèmes de satisfaction de contraintes, éventuellement flexibles, complètent l'éventail de ces méthodes. Le traitement des représentations liées à la logique requiert des algorithmes spécialisés, qu'il s'agisse de déduction automatique, de satisfaisabilité d'ensembles de propositions, ou encore de programmation logique. Ils sont aussi présentés dans ce volume. L'algorithmique des modèles graphiques de représentation, en particulier celle des réseaux de type bayésien, ainsi que le développement d'algorithmes pour la planification ou l'apprentissage automatique sont également abordés. La postface dresse un parallèle entre les problématiques algorithmiques en recherche opérationnelle et en intelligence artificielle.

www.cepadues.com