Orsay
n° d'ordre : 6079

U.F.R. scientifique d'Orsay

Intégration des biais de langage à l'algorithme générer-et-tester

---

Contributions à l'apprentissage disjonctif

Thèse
présentée
pour obtenir
le titre de Docteur en Sciences
Spécialité Informatique
par

Fabien Torre

soutenue le 28 janvier 2000
devant la commission d'examen

Francesco Bergadano	rapporteur
Christine Froidevaux	président
Yves Kodratoff	directeur administratif
Pierre Marquis	rapporteur
Céline Rouveirol	directeur scientifique
Henry Soldano	examinateur

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

Préface

Le soir tombait...

Il tombait bien, d'ailleurs, et juste à pic pour remplacer le jour, dont le déclin rapide laissait à penser qu'il ne passerait pas la nuit.

À l'horizon, dans une apothéose de gloire comparable à celle de la Sécurité sociale, le soleil se couchait. C'était un vigoureux coucher de soleil, et les plus vieux du pays disaient que, de mémoire de plus vieux du pays, ils ne se rappelaient pas en avoir jamais contemplé d'aussi réussi, depuis le début de leur carrière de plus vieux du pays.

Il faisait bon ; l'air était saturé de senteurs parfumées où dominaient les odeurs de poivre et sel des barbouziers nains et des gougnafiers moléculaires. Au zénith, Vespa, l'astre bénéfique des usagers du vélomoteur, allumait ses feux de position.

Le paysage, d'une émouvante grandeur, était également grandeur nature, et l'on entendait, sous l'ormeau, battre la crème fraîche à coups de marteau.

Au détour d'un chemin, un moustique aux yeux bleus, bègue au surplus, et fainéant, de surcroît, venait se vautrer sur le faîte d'un brin d'herbe pour y attendre la suite des événements.

Dans la cité, toute proche, chacun organisait sa vie nocturne au mieux des intérêts supérieurs de la nation.

À la lueur d'un réverbère, deux ivrognes échangeaient des vœux à l'occasion de la nouvelle lune, tandis qu'un fils de famille, dévoyé, préférait s'engager dans une rue adjacente plutôt qu'à la légion étrangère.

Assise sous la lampe, une jeune femme, qui attendait un enfant, s'apprêtait à aller le chercher à l'autocar de 22 heures.

Dans sa mansarde, un étudiant, qui préparait sa licence de lettres, compulsait fébrilement les textes des grands philosophes depuis l'époque de Confucius jusqu'à mercredi en huit.

Sur sa table de salle à manger, un monteur en chauffage central, qui avait apporté du travail à finir à la maison, terminait le rivetage d'une chaudière à mazout.

Dans un fauteuil Empire, et en reps imprimé, un vieux bibliophile lisait l'*Introduction à la vie des vôtres*, de Teilhard de Chardin. C'était un beau vieillard : la moustache coupée au ras du sol, la barbe taillée à la blanquette à l'ancienne, il entrait dans sa

soixante-dix-septième année, mais admirablement conservé, il en paraissait à peine quatre-vingt-deux.

Enfin, dans une brasserie, un colonel en retraite soupirait mélancoliquement : « Avoir commandé un régiment, bougonnait-il, et se voir réduit à commander une choucroute, quelle dérision ! »

Et le temps passait. Et les braves gens, en s'endormant, les uns sur le dos, les autres sur le rôti, songeaient que, tout compte fait, tout n'était pas si mal dans ce meilleur des mondes possibles, en regrettant, toutefois, que, dans l'ensemble, ses possibilités soient tellement limitées.

En résumé, tout était en ordre, et chaque composante de ce climat nocturne ayant fait consciencieusement son devoir, plus rien ne s'oppose à ce que commence réellement le récit qui attend sagement et patiemment que le feu vert lui soit donné pour passer à l'action.

Pas d'opposition ? Pas d'objection ? Alors, place à ce qui suit.

Pierre Dac.

Introduction

L'Informatique, et plus particulièrement l'Intelligence Artificielle et l'Apprentissage Automatique présentent deux visages. D'une part, des méthodes anciennes et bien analysées répondant à des problèmes parfaitement caractérisés. Citons par exemple en Intelligence Artificielle l'algorithme A^* et l'algorithme *minmax*. D'autre part, nous trouvons des méthodes souvent amusantes et inattendues qui n'ont pour l'instant aucune application ou qui obtiennent de très bons résultats sans qu'il soit possible d'expliquer précisément pourquoi ou de démontrer de réels gains par rapport à des méthodes plus traditionnelles. Citons dans ce cas les ordinateurs quantiques et les machines à ADN.

Ces oppositions, anciennes méthodes fondées contre approches récentes moins justifiées mais tout aussi efficaces, nous les retrouvons en Apprentissage Automatique, branche de l'Intelligence Artificielle qui cherche à découvrir des règles à partir d'observations : pour un concept cible, nous disposons d'exemples et de contre-exemples et il faut apprendre des règles permettant de décider si un nouvel exemple appartient ou non au concept. Il est encore courant d'utiliser des algorithmes datant des années 70 comme l'algorithme *générer-et-tester* pour apprendre sur un faible nombre d'observations dont on suppose que leur partition en exemples et contre-exemples est juste. La fouille de données a modifié cette problématique : il s'agit maintenant d'apprendre sur des bases comprenant des millions d'exemples dont beaucoup sont bruités. Signalons aussi que certains utilisent des fourmis pour résoudre des problèmes difficiles, comme par exemple l'apprentissage non supervisé, c'est-à-dire un apprentissage où l'on ne dispose pas de l'étiquetage exemple/contre-exemple.

Cette thèse reflète cette dualité à travers les deux parties dont elle est composée.

Une première partie de cette thèse est consacrée à l'intégration des biais de langage (un biais de langage étant une propriété connue sur la forme de la solution, comme par exemple une borne sur sa taille), dans l'algorithme *générer-et-tester* que nous venons d'évoquer en tant que méthode traditionnelle. L'utilisation de ces biais de langage doit avoir deux avantages. Tout d'abord, il y a un aspect *qualité* de l'apprentissage. Plusieurs hypothèses peuvent avoir un comportement satisfaisant vis-à-vis des exemples. Les biais vont nous indiquer quelles hypothèses préférer.

Ensuite, nous pouvons espérer que ces connaissances supplémentaires rendront la recherche plus *efficace* : les biais désignent des régions particulières de l'espace de recherche ; si la recherche se réduit à ces zones, elle sera indéniablement plus efficace qu'une exploration exhaustive.

Or, l'algorithme *générer-et-tester* s'il permet l'utilisation de biais de langage n'en est pas pour autant plus efficace. C'est donc à améliorer l'aspect efficacité des biais de langage dans l'algorithme *générer-et-tester* que nous allons consacrer la première partie de cette thèse.

Une hypothèse est en général représentée par une conjonction mais il peut arriver qu'une telle conjonction ne suffise pas à séparer les exemples des contre-exemples. Dans ce cas, il faut recourir à une définition de concept qui sera une disjonction de conjonctions et l'on parle alors d'apprentissage *disjonctif*. Comme nous le verrons, ce type d'apprentisage est beaucoup plus difficile que la simple recherche d'une solution conjonctive.

Nous proposerons dans la seconde partie de cette thèse deux méthodes originales : l'une de type stochastique qui construit les éléments de la solution aléatoirement, l'autre utilisant l'évolution de créatures appelées *zamis* pour découvrir une solution.

Le plan complet de la thèse illustrant ses deux lectures possibles est donné à la Figure 1.

Le chapitre 1 discute la position de l'Apprentissage automatique en Intelligence Artificielle. Nous y présentons ensuite quelques fondements de l'Apprentissage automatique et plus particulièrement les éléments qui seront utilisés dans la suite de la thèse : l'influence des langages servant à décrire les problèmes d'apprentissage, ainsi que le rôle d'opérations clefs sur ces langages comme le test de subsomption et le calcul de moindre généralisé. Nous parlerons également de l'évaluation du résultat de l'apprentissage et du *No Free Lunch Theorem* qui indique que toutes les méthodes d'apprentissage sont strictement équivalentes.

Dans le chapitre 2 nous présentons les deux notions que nous allons marier ensuite : les biais de langage et l'algorithme *générer-et-tester*. Des biais, nous dirons leur nature, leur nécessité et leur justification. Puis, nous décrirons l'algorithme *générer-et-tester* montrant par là-même que cet algorithme est adapté à la résolution d'un problème d'apprentissage. Enfin, nous parlerons de cet algorithme et des biais dans un contexte particulier, celui de la Programmation Logique Inductive où les exemples et hypothèses sont des formules logiques du premier ordre.

Au chapitre 3, nous déterminons les conditions sous lesquelles un biais de langage permet un élagage dynamique de l'espace de recherche, sans risque de perdre une solution. Une propriété vérifiant ces conditions sera dite *privée*. Cette notion et son utilisation seront illustrées à travers la définition du système Grandma qui intègre grammaires et propriétés privées.

Le chapitre 4 caractérise les relations qui rendent les propriétés privées : il s'agira des *relations naturelles* des propriétés. Puis, nous cherchons les opérateurs de raffinements qui satisfont ces relations naturelles et plus particulièrement des opérateurs idéaux : bien que de tels opérateurs n'existent pas sous θ-subsomption, nous exhiberons un opérateur idéal pour certaines de nos relations naturelles.

Le chapitre 5 abandonne les opérateurs de raffinement classiques pour pouvoir

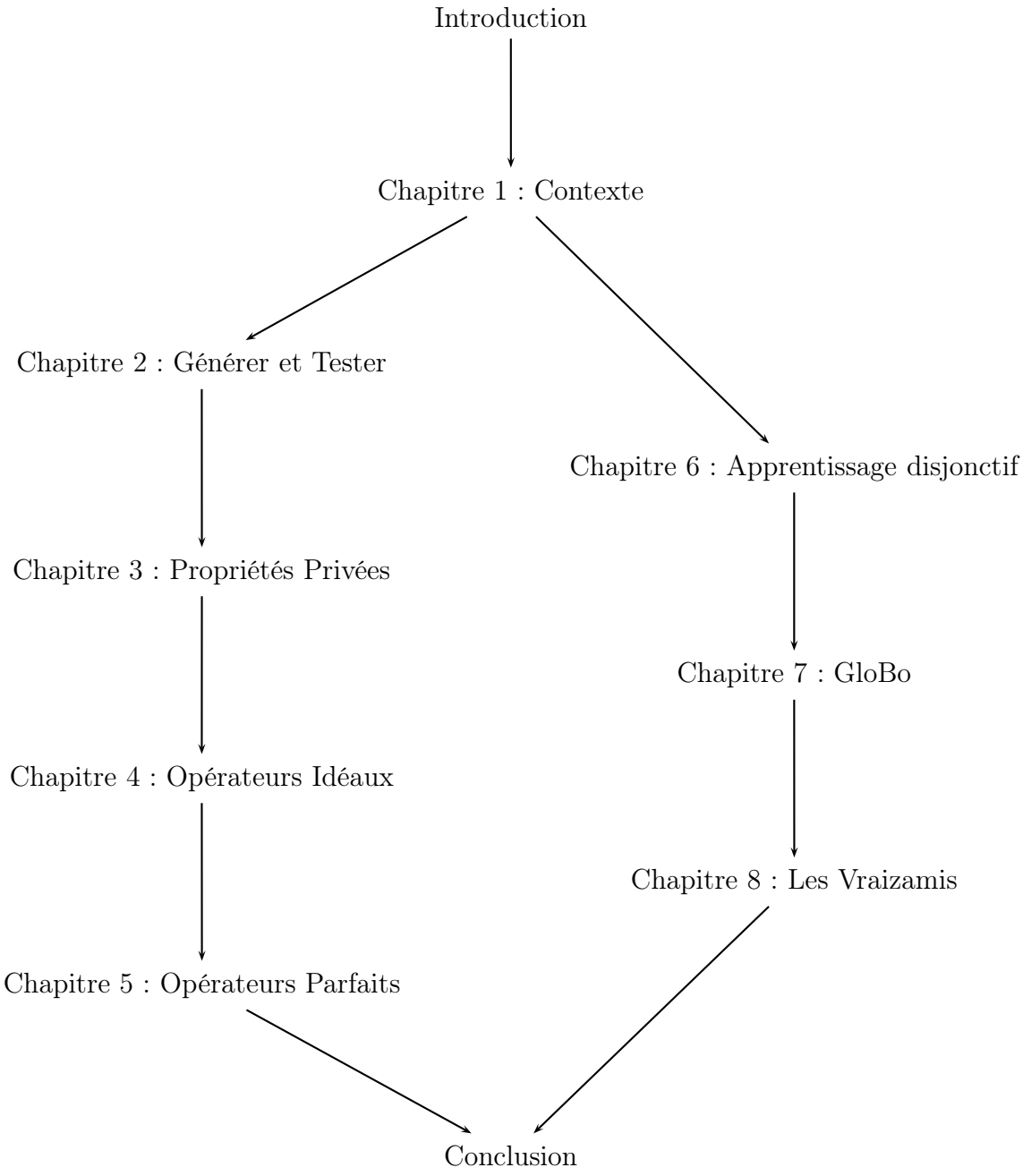


FIGURE 1 – Plan de la thèse

intégrer les biais de langage à tous les niveaux de l'algorithme *générer-et-tester* : en fonction de ces biais nous définirons la relation de généralité, les hypothèses qui seront les points de départ de la recherche et les opérateurs, que nous nommerons alors *opérateurs parfaits*.

Le chapitre 6 entame la seconde partie de cette thèse consacrée à l'apprentissage supervisé disjonctif en présentant ce problème : ses difficultés propres, les méthodes qui lui sont dédiées et leurs défauts. Cela nous permettra de caractériser le type de disjonction que nous allons chercher à obtenir dans les chapitres suivants.

Le chapitre 7 présente le système GloBo qui constitue une approche stochastique du problème d'apprentissage supervisé disjonctif. Nous décrivons l'algorithme de GloBo de complexité cubique. Nous définissons ensuite une mesure permettant d'évaluer la probabilité que GloBo avait de découvrir ce qu'il a appris, ou autrement dit, de mesurer l'adéquation entre les ensembles d'apprentissage, GloBo et la solution découverte. Nous validons cette méthode à travers plusieurs expérimentations ; en particulier, nous présentons les résultats de GloBo confronté au *PTE Challenge*. Enfin, nous terminons par une version générique de GloBo dont les instanciations vont de l'apprentissage non supervisé au traitement de données bruitées, en passant par une version de complexité linéaire.

Dans le chapitre 8, nous explorons la voie de l'éco-résolution pour résoudre un problème d'apprentissage disjonctif. Nous présentons le monde des Vraizamis où les individus se rassemblent suivant un critère d'amitié et dans lequel nous allons plonger les exemples du concept à découvrir. Nous utilisons les Vraizamis pour apprendre en liant cette amitié à la correction vis-à-vis des contre-exemples. Enfin, nous discutons de la complexité de la méthode, de ses risques de blocage, de ses liens avec l'algorithme GloBo, et de l'influence de la topologie du monde des Vraizamis.

CHAPITRE 1

Contexte de la thèse

Dans ce premier chapitre, nous esquissons la position de l'Apprentissage automatique dans l'Intelligence Artificielle et, plus généralement, la position de cette dernière en Informatique. En particulier, nous décrirons le goulot d'étranglement de l'IA à travers la problématique des systèmes à base de connaissances.

Cela nous permettra de motiver l'Apprentissage automatique et d'introduire les langages utilisés pour décrire les problèmes d'apprentissage, ainsi que les opérations de base de l'apprentissage sur ces langages.

Ensuite, nous discuterons des différents critères permettant d'évaluer le résultat de l'apprentissage et des différentes manières de le faire. Puis, nous présenterons le *No Free Lunch Theorem* : celui-ci indique que toutes les méthodes d'apprentissage sont strictement équivalentes.

Malgré cela, nous terminerons ce chapitre introductif par la présentation de quelques problèmes réels auxquels l'apprentissage automatique a apporté des solutions très intéressantes. En particulier, nous verrons à cette occasion que l'apprentissage automatique est capable de découvertes scientifiques et que la première problématique de l'apprentissage est donc dépassée : l'apprentissage automatique constitue aujourd'hui un domaine autonome de l'Intelligence Artificielle.

1.1 Intelligence Artificielle et Science-fiction

L'expert d'échecs n'est pas le héros à imiter,
l'expert, c'est le bébé.
Francisco Varéla, 1997.

Père...
tu as caché ces choses aux sages et aux intelligents
et tu les as révélées aux petits enfants.
Luc 10.21.

1.1.1 Motivations

Le terme d'*informatique* a été officialisé et précisément défini par l'Académie française en avril 1966. La définition donnée par les académiciens est la suivante :

> Science du traitement rationnel, notamment par machines automatiques, de l'information considérée comme le support des connaissances humaines et des communications dans les domaines techniques, économiques et sociaux.

Nous constatons d'emblée que l'Informatique n'est pas réduite à l'utilisation d'ordinateurs et il est naturel de se demander si l'homme par lui-même peut réaliser ce traitement informatique. Même si l'on restreignait la définition aux *machines automatiques* évoquée par l'Académie, il n'est pas évident que celles-ci ne comprennent pas le cerveau humain. Considérons, en effet, la définition du mot « automatique » :

1. Qui fonctionne sans intervention humaine.
2. Qui s'exécute sans la participation de la volonté.
3. Qui intervient de manière régulière ou inéluctable.

Le premier sens fait clairement allusion aux machines, en revanche, le deuxième indique qu'un être humain peut agir de manière automatique. Si bien que Muhammad ibn Mūsā al-Khwārizmī, mathématicien de la première moitié du IX^{e}siècle, est souvent considéré comme le premier informaticien pour avoir donné son nom à la notion, plus ancienne, d'algorithme : son besoin était de transmettre par écrit des méthodes mathématiques de manière à ce qu'elles soient applicables par d'autres personnes, sans qu'il soit nécessaire de leur expliquer quoi que ce soit, ni même qu'elles aient compris les fondements de la méthode.

D'où la question posée par l'Intelligence Artificielle : si le cerveau est, comme l'ordinateur, une machine, sont-ils équivalents ? Naturellement, leurs modes de fonctionnement sont trop différents pour que l'on se risque à une quelconque analogie. Cependant, la question de l'équivalence peut être posée à un niveau plus abstrait : peuvent-ils réaliser la même chose, quels que soient les moyens mis en œuvre par chacun d'eux ?

L'Intelligence Artificielle étudie donc l'esprit en tant que système de traitement de l'information dans le but de reproduire les facultés mentales *ordinaires*. Autrement dit, ne pas reproduire le fonctionnement interne d'un cerveau mais simuler son comportement observable de l'extérieur. Et pas celui d'un champion d'échecs,

mais d'un individu ordinaire. C'est là tout l'esprit du test de Turing [Turing, 1950]. Qu'importe si un esprit peut être ou non implanté dans un ordinateur, l'essentiel est que chaque capacité mentale puisse être simulée sur l'une de nos machines.

1.1.2 Historiques croisés

Nous avons évoqué les débuts de l'informatique avec al-Khwārizmī, voyons maintenant quelques événements plus proches de nous.

En 1818, Mary Shelley évoque avec *Frankenstein* la possibilité d'assembler un être pensant. Quelques années plus tard, en 1833, apparaît la notion de *programme*, c'est-à-dire l'idée d'appliquer un algorithme de manière automatique à l'aide d'une machine. Charles Babbage conçoit sa machine analytique, programmable à l'aide de cartes perforées, à la manière du métier à tisser de Jacquard. Augusta Ada King, comtesse de Lovelace, a écrit bon nombre de programmes pour la machine de Charles Babbage. À propos des possibilités de la machine, elle indiqua :

> La Machine analytique n'a aucune prétention de créer quoi que ce soit, elle peut exécuter tout ce que nous savons ordonner pour la faire fonctionner.

Nous le voyons, la possibilité d'une Intelligence Artificielle soulevait déjà des objections, alors que la Machine analytique ne devait jamais être construite. Alan Turing nommera cette idée l'« Objection de Lady Lovelace ».

En 1835, Edgar Allan Poe doit démontrer qu'un automate champion d'échecs ne peut pas être une « pure machine » mais qu'il contient en réalité un être humain (*Le joueur d'échecs de Maelzel*, 1835).

En 1847, George Boole propose son algèbre qui, un siècle plus tard, servira de base aux ordinateurs. Il indiquait que les axiomes de son algèbre décrivaient les opérations élémentaires de l'esprit humain.

Venons-en à l'informatique contemporaine avec la naissance des ordinateurs
— le Z3 de Konrad Zuse le 12 mai 1941,
— Harvard Mark I de Howard Aiken en janvier 1943,
— l'Eniac en 1946,
— l'Edsac de Maurice Wilkes à Cambridge en 1949.

Le formidable bond en avant qui a suivi a débridé bon nombre d'imaginations : puisqu'une machine peut effectuer rapidement et sans fatigue ses opérations élémentaires, elle devrait, au minimum, égaler les capacités humaines.

Dès 1943, McCulloch et Pitts proposent l'architecture des réseaux de neurones pour simuler l'intelligence. En 1949, Orwell publie *1984* où les ordinateurs prennent une place importante.

L'année 1950 est très riche pour l'Intelligence Artificielle : Alan Turing propose son test [Turing, 1950] et Shannon conçoit un programme d'échecs, événement qui est généralement considéré comme la naissance de l'Intelligence Artificielle.

C'est aussi en 1950 qu'Isaac Asimov publie le premier volume du cycle des *Robots* avec ses célèbres trois lois de la robotique. Moins souvent cité en référence à l'Intelligence Artificielle, le second cycle d'Isaac Asimov, *Fondation*, ne peut qu'interpeller les apprentistes : grâce aux règles de la psychohistoire, Hary Seldon est capable de prédire le devenir de l'humanité sur plusieurs millénaires.

À la conférence de Dartmouth en 1956, où sont entre autres présents Minsky, Simon et Newell, McCarthy emploie le terme d'Intelligence Artificielle : c'est la naissance *officielle* du domaine suivie d'une période d'euphorie. C'est alors qu'a été clamée la résolution prochaine de nombreux problèmes par les ordinateurs : la compréhension du langage naturel, la traduction automatique en sont les exemples les plus célèbres. La naissance et la prolifération de *systèmes experts*, c'est-à-dire de machines capables d'égaler les experts humains dans de nombreux domaines, étaient elles aussi annoncées.

Puis, il faut constater que ces objectifs sont hors de portée et vont le rester longtemps : c'est le désenchantement, en particulier avec Hubert Dreyfus qui établit un mauvais bilan pour l'Intelligence Artificielle dans son livre *What Computers Can't Do*.

1.1.3 Bilan

Où en est-on aujourd'hui ? Même si les techniques développées dans le domaine sont maintenant reconnues et utilisées, il est bien clair que l'Intelligence Artificielle n'a pas été atteinte mais faut-il s'en contrarier ? Bien au contraire, on assiste à une surenchère ! Ainsi, la Vie Artificielle se propose de construire des machines entièrement autonomes présentant toutes les caractéristiques d'un être vivant... et naturellement, ces machines devront être dotées d'intelligence ! Nous-mêmes, à un niveau bien moindre, nous présenterons au chapitre 8 (page 139), un sytème d'apprentissage proche de la Vie Artificielle puisque ce système utilise l'éco-résolution.

Sur la route de l'Intelligence Artificielle, bon nombre d'avancées ont trouvé des applications pratiques. Ainsi, l'Intelligence Artificielle a interagi avec l'architecture des machines : la *société de l'esprit* décrite dans [Minsky, 1988] donne naissance, un an plus tard, à la *Connection Machine*.

Nous allons maintenant nous intéresser plus particulièrement aux systèmes experts, rebaptisés *systèmes à base de connaissances*. Force est de constater aujourd'hui qu'ils sont bien loin d'avoir remplacé les experts humains, tout comme la compréhension du langage naturel et la traduction automatique sont toujours hors de portée à ce jour. Cependant, nous allons voir que le cas des systèmes à base de connaissances, est différent : des méthodes efficaces existent mais c'est la connaissance de base, celle sur laquelle les systèmes doivent raisonner, qui fait défaut.

1.2 Systèmes à base de connaissances

Les systèmes à base de connaissances sont représentatifs de la deuxième phase qu'a connue l'Intelligence Artificielle, centrée sur la manipulation de la connaissance.

Classiquement, il est possible de considérer que l'utilisation d'un système à base de connaissances fait intervenir trois entités.

1. Tout d'abord, la *base de règles* regroupe les connaissances sur le domaine sous une forme qui permet de raisonner.
2. Précisément, le *moteur d'inférences* regroupe les mécanismes qui contrôlent l'application de ces règles.

3. Enfin, *la base de faits* décrit la situation actuelle, ou autrement dit, le problème à résoudre.

Pour peu que la base de règles soit disponible auprès d'un expert humain, il reste à définir le moteur d'inférences, c'est-à-dire la manière de raisonner. La logique nous montre que certains raisonnements peuvent être vus comme des opérations purement syntaxiques et, en effet, des mécanismes simples peuvent être mis en œuvre. Par exemple, le chaînage avant, qui se résume à appliquer autant de fois que possible le *modus ponens* sur la base de faits, est une méthode complète dans le cas où la négation n'est utilisée ni dans la base de faits, ni dans la base de règles : le chaînage avant permet alors d'obtenir tous les faits qui sont logiquement impliqués par les règles et les faits.

Nous le voyons, le cas des systèmes à base de connaissances est différent de celui de la compréhension du langage naturel ou de la traduction automatique. Ces tâches nécessitent de trouver le sens d'une phrase et d'en donner une représentation informatique, problème tellement difficile que les recherches s'attachent actuellement à résoudre des approximations des problèmes originaux.

Ici, dans le cas de bases n'utilisant pas la négation, les mécanismes sous-jacents sont simples, d'une complexité tout à fait raisonnable et connus depuis longtemps : le *modus ponens*, par exemple, fut identifié et utilisé dès l'antiquité !

Il n'est donc pas étonnant que les premiers systèmes opérationnels soient apparus très tôt. En 1971, a été proposé le système Dendral qui fournissait les structures chimiques de molécules à partir des résultats de différentes mesures sur ces molécules. Le système expert le plus célèbre, Mycin, est lui paru en 1976 et propose des diagnostics médicaux.

Pourtant, nous l'avons signalé, ces systèmes ne sont pas aussi répandus que prévu. L'explication est qu'une difficulté inattendue est survenue : l'obtention de la base de règles auprès de l'expert s'est révélée être une tâche terriblement difficile.

Cette acquisition des connaissances constitue ce que Edward A. Feigenbaum a identifié comme le goulot d'étranglement de l'Intelligence Artificielle [Feigenbaum, 1977].

> *The second insight from the past work concerns the nature of the knowledge that an expert brings to the performance of a task. Experience has shown us that this knowledge is largely heuristic knowledge, experiential, uncertain – mostly "good guesses" and "good practice", in lieu of facts and rigor. Experience has also taught us that much of this knowledge is private to the expert, not because he is unwilling to share publicly how he performs, but because he is unable. He knows more than he is aware of knowing.*

Ce talon d'Achille des systèmes à base de connaissances est donc la faiblesse de toute l'Intelligence Artificielle et a conduit à la troisième phase de l'Intelligence Artificielle : l'heure n'est plus à la manipulation de la connaissance, mais à son acquisition. Cela n'a rien d'étonnant puisque chez l'être humain, comme dans les systèmes à base de connaissances, il faut des connaissances de base pour pouvoir raisonner et exercer son intelligence. Et tout comme chez l'être humain, ces connaissances peuvent être obtenues par apprentissage.

L'*apprentissage automatique* est une voie qui se propose de résoudre ce problème d'acquisition des connaissances : à partir des faits et des conclusions appropriées, il s'agit d'apprendre les règles qui permettront de prendre les bonnes décisions dans des situations à venir.

Cette approche fut rapidement envisagée puisque un Meta-Dendral a été présenté dans le papier même où le goulot d'étranglement était identifié [Feigenbaum, 1977]. Celui-ci devait découvrir des règles utilisables par Dendral. Pour cela, il fallait fournir à Meta-Dendral la description de molécules bien connues : cette description comprenait les mesures qui constitueraient les entrées requises par Dendral, ainsi que la structure chimique de la molécule, c'est-à-dire la conclusion attendue de Dendral.

Nous verrons que l'apprentissage a aujourd'hui dépassé ce cadre originel, pour devenir une branche de l'Intelligence Artificielle autonome. En particulier, les méthodes actuelles d'apprentissage sont capables de réaliser de véritables découvertes scientifiques comme nous le verrons à la section 1.8 (page 19).

1.3 L'apprentissage automatique

Dans l'enseignement des sciences, l'exemple vaut mieux que la théorie.
Isaac Newton, d'après « 60 gags de Boule et Bill » volume 1.

L'apprentissage peut se réduire à appliquer le principe d'induction, dont la problématique suivante : établir des règles générales à partir de cas particuliers.

Cette branche de l'Intelligence Artificielle existe depuis plus de 20 ans. Une borne majeure est la publication du premier volume de *Machine Learning* en 1983.

Bien que les paradigmes d'apprentissage soient multiples [Kodratoff, 1989, Cornuéjols et Moulet, 1997], les recherches se sont principalement attachées à explorer deux voies.

- L'*apprentissage supervisé* cherche la définition d'un concept particulier, à partir d'exemples qui sont étiquetés *positif* ou *négatif*, selon qu'ils appartiennent au concept ou non.
- En *apprentissage non supervisé* (ou encore *classification*), les exemples disponibles ne sont pas étiquetés. Une fois identifiées les classes de base, il est courant de chercher des classes plus générales et de construire ainsi une hiérarchie de concepts.

Dans la suite, nous discuterons essentiellement d'apprentissage supervisé. Selon Tom Mitchell [Mitchell, 1982], cette tâche nécessite :

- un langage $\mathcal{L}_e$ permettant de décrire les exemples ;
- un langage $\mathcal{L}_h$ pour les hypothèses, c'est-à-dire une représentation des éventuelles définitions du concept ; ce langage constitue l'espace de recherche de l'apprentissage ;
- un test de subsomption $\succeq$ qui, à partir d'une hypothèse $h \in \mathcal{L}_h$ et d'un exemple $e \in \mathcal{L}_e$, indique si h implique ou non e (dans la suite, nous dirons que h couvre ou non e) ;
- un ensemble fini A^+ d'exemples appartenant au concept étudié et un ensemble fini A^- de contre-exemples.

À partir de ces données, le problème qui va nous occuper est défini comme suit.

Définition 1.1 (apprentissage supervisé)
Étant donnés $(\mathcal{L}_e, \mathcal{L}_h, A^+, A^-, \succeq)$*, trouver une hypothèse* $h \in \mathcal{L}_h$ *telle que*

$$\forall e \in A^+ : h \succeq e$$
$$\forall e \in A^- : h \not\succeq e$$

Cette définition pose que la solution doit être *correcte* (elle ne couvre pas d'exemples négatifs) et *complète* (elle couvre tous les exemples positifs). Ces notions de correction et complétude peuvent être affaiblies, en particulier lorsqu'il est connu que les données sont *bruitées*, c'est-à-dire que certains exemples ont été mal étiquetés.

Outre sa caractérisation par une définition D de $\mathcal{L}_h$, un concept peut être vu comme une partition particulière de tous les exemples de $\mathcal{L}_e$ en deux sous-ensembles : l'un regroupant les exemples du concept, l'autre les contre-exemples. Le premier sera noté $\mathcal{L}_e^+$, le second $\mathcal{L}_e^-$:

$$\mathcal{L}_e^+ = \{e \in \mathcal{L}_e \mid D \succeq e\}$$
$$\mathcal{L}_e^- = \{e \in \mathcal{L}_e \mid D \not\succeq e\}$$

Nous allons maintenant discuter des langages utilisés pour $\mathcal{L}_e$ et $\mathcal{L}_h$.

1.4 Rôle des langages de représentation

- Si je les ai mangées, qui m'a donné une bouche ?
- Il n'y a pas de réponse dans ton vocabulaire.
William Golding, Chris Martin.

Nous avons vu que deux langages devaient être distingués au cours du processus d'apprentissage : $\mathcal{L}_e$ pour représenter les exemples et $\mathcal{L}_h$ pour les hypothèses.

1.4.1 Subsomption et généralisation

[Levesque et Brachman, 1989] indiquent qu'en Intelligence Artificielle un compromis est nécessaire entre l'expressivité et la maniabilité d'un langage. La maniabilité correspond à la complexité d'opérations clefs dans ce langage.

En apprentissage, ces opérations dépendent bien sûr de la méthode, mais il en est une qui, par force, est commune à tous les systèmes. Il s'agit du test de subsomption. Celui-ci va donc nous permettre d'évaluer la maniablité d'un langage, indépendamment de la méthode.

Définition 1.2 (subsomption)
Étant données deux hypothèses H_1 *et* H_2*, on dit que* H_1 *subsume* H_2 *si tous les exemples de* $\mathcal{L}_e$ *couverts par* H_2*, sont aussi couverts par* H_1 *:*

$$\{e \in \mathcal{L}_e : H_2 \succeq e\} \subseteq \{e \in \mathcal{L}_e : H_1 \succeq e\}$$

Par abus de langage, cette subsomption entre H_1 *et* H_2 *sera notée* $H_1 \succeq H_2$*.*

La relation $\succeq$ étant un pré-ordre, elle induit une relation d'équivalence entre hypothèses.

Définition 1.3 (équivalence)
Deux hypothèses H_1 et H_2 sont dites équivalentes ($H_1 \sim H_2$) si $H_1 \succeq H_2$ et $H_2 \succeq H_1$, c'est-à-dire si H_1 et H_2 couvrent exactement les mêmes exemples de $\mathcal{L}_e$.

Cette définition en extension de la subsomption est rarement utilisée grâce à l'astuce de représentation unique (*single-representation trick* de [Cohen et Feigenbaum, 1982]) que nous adoptons immédiatement : cette astuce pose que le langage des exemples est inclus dans le langage des hypothèses ($\mathcal{L}_e \subseteq \mathcal{L}_h$). Par conséquent, les discussions suivantes concernant les hypothèses valent pour les exemples (puisqu'ils sont des hypothèses).

En particulier, à partir de maintenant, le test de subsomption $\succeq$ sera défini sur $\mathcal{L}_h$. Ce test est donc utilisable entre une hypohèse et un exemple, aussi bien qu'entre deux hypothèses.

Nous avons fait le choix de présenter $\succeq$ comme un test de subsomption bien qu'il s'agisse en même temps d'une relation, nommée *relation de généralité*, sur $\mathcal{L}_h$. Cependant, nous tenons à faire une distinction entre ces deux visions de $\succeq$ car une relation de généralité n'est pas toujours calculable, ce qui est la moindre des choses pour un test. Ainsi, nous verrons que l'implication logique est une relation de généralité entre clauses mais que le problème qui consiste à établir si une clause en implique une autre est indécidable.

L'aspect relationnel du test sera tout de même utilisé. Il sera parfois commode de considérer que l'espace de recherche est ordonné par $\succeq$. Par exemple, il est courant de l'utiliser pour ordonner la recherche : du plus général au plus spécifique ou l'inverse.

Une seconde opération cruciale est le calcul des moindres généralisés d'un ensemble d'hypothèses que nous définissons comme les éléments minimaux de l'ensemble des majorants (pour $\succeq$) de ces hypothèses.

Définition 1.4 (moindre généralisé)
Un moindre généralisé G de $H_1, \ldots, H_n$ doit vérifier :

$$\forall H_i : G \succeq H_i$$
$$[\exists G' : (\forall H_i : G' \succeq H_i) \wedge (G \succeq G')] \Rightarrow (G \sim G')$$

Le calcul d'un moindre généralisé n'est pas utilisé par tous les systèmes. Il offre néanmoins un intérêt théorique général.

Hypothèse 1.1 (existence d'une solution)
S'il n'existe pas de moindre généralisé de A^+ qui soit correct par rapport à A^-, alors le problème d'apprentissage n'a pas de solution.

Nous avons considéré jusqu'ici qu'un ensemble d'hypothèses pouvait posséder plusieurs moindres généralisés. Nous supposons maintenant qu'il est unique (nous verrons que c'est le cas pour la majorité des langages). Avec cette restriction, l'hypothèse 1.1 se réduit à :

Corollaire 1.1
Si le moindre généralisé de A^+ est correct par rapport à A^- alors le moindre généralisé de A^+ est solution, sinon il n'y a pas de solution.

Dans le cas où le moindre généralisé est solution, il n'est pas nécessairement l'unique solution : les hypothèses plus générales que G restent complètes par rapport à A^+ et certaines d'entre elles peuvent être correctes par rapport à A^-.

Nous avons donc fait apparaître qu'une solution n'existe pas toujours : $\mathcal{L}_h$ peut ne pas contenir d'hypothèse correcte et complète. La solution passe alors par un apprentissage disjonctif.

1.4.2 Apprentissage disjonctif

Une interprétation de cette absence de solution dans $\mathcal{L}_h$ est que les exemples positifs ne sont pas tous positifs pour la même raison, et que ces différentes raisons ne peuvent être résumées par une unique hypothèse de $\mathcal{L}_h$.

Il faut recourir à plusieurs hypothèses de $\mathcal{L}_h$, chacune étant correcte par rapport à A^-, mais ne couvrant qu'une partie de A^+. Chaque exemple de A^+ est couvert par l'une ou l'autre des hypothèses de la solution. Il s'agit donc typiquement d'une disjonction.

Le problème d'apprentissage que nous avons décrit à la définition 1.1 doit donc être reformulé. Nous avons le choix entre garder le même problème avec $2^{\mathcal{L}_h}$ comme langage des hypothèses, ou bien conserver le même $\mathcal{L}_h$ mais chercher une disjonction composée de plusieurs éléments de $\mathcal{L}_h$. Dans l'absolu, ces deux approches sont équivalentes. Dans la pratique, les algorithmes d'apprentissage privilégient la seconde.

Définition 1.5 (apprentissage supervisé disjonctif)
Étant donnés $(\mathcal{L}_e, \mathcal{L}_h, A^+, A^-, \succeq)$*, trouver un ensemble d'hypothèses* $\mathcal{H} \subseteq \mathcal{L}_h$ *tel que*

$$\forall e \in A^+, \exists h \in \mathcal{H} : h \succeq e$$
$$\forall e \in A^-, \forall h \in \mathcal{H} : h \not\succeq e$$

Cette fois-ci, l'existence d'une solution est garantie par l'astuce de représentation unique que nous avons évoquée ci-dessus, et c'est pourquoi cette convention est présentée comme une astuce. Dans le cas où cette convention est adoptée, en effet, il y a au moins la solution constituée de la disjonction des exemples positifs, solution qualifiée d'*apprentissage par cœur*.

Cette garantie d'existence d'une solution se paye par de nouvelles difficultés que nous décrirons au chapitre 6 (page 95). En particulier, il faudra éviter l'apprentissage par cœur puisque celui-ci se contente de restituer ce qui lui a été fourni et rien de plus.

Nous avons défini subsomption et moindres généralisés dans l'absolu, nous allons maintenant considérer ces opérations pour des langages particuliers.

1.5 Exemples de langages

1.5.1 Ensembles finis de valeurs discrètes

Nous nous plaçons ici dans le cas où $\mathcal{L}_e$ et $\mathcal{L}_h$ sont égaux à l'ensemble des parties d'un ensemble fini de valeurs Ω. C'est un des langages les plus simples que l'on puisse considérer et qui présente tout de même quelque intérêt.

TABLE 1.1 – Tirages du loto français

A^+						A^-					
7	14	27	34	37	47	4	14	16	25	34	40
7	14	18	25	44	45	18	25	39	40	45	46
4	9	13	16	23	49	2	19	26	32	43	45
8	9	15	30	31	48	11	21	36	39	41	43
9	13	26	35	37	42	3	6	17	29	37	43
5	9	18	23	27	49	5	23	24	30	43	47
4	14	18	24	28	30	3	19	29	31	33	45
2	9	14	27	35	46	10	15	25	32	37	38
8	15	25	28	35	49	20	24	37	41	44	49
7	11	19	37	42	46	25	30	32	35	39	47

À titre d'illustration, nous utilisons les ensembles d'apprentissage définis à la Table 1.1 Ces exemples sont des tirages du loto français : Ω est l'ensemble des entiers de 1 à 49. Pour ce problème, A^+ est constitué de tirages qui ont eu beaucoup de gagnants, A^- de tirages qui en ont eu très peu. Il s'agit donc de trouver les numéros qui, lorsqu'ils sont présents dans un tirage, font qu'il y aura beaucoup de gagnants.

Subsomption

La relation de généralité est tout naturellement l'inclusion entre ensembles. Par exemple, $(7, 13)$ subsume $(7, 9, 13, 33, 38, 43)$ mais pas $(2, 7, 11, 23, 45, 47)$. L'hypothèse la plus générale est l'ensemble vide puisqu'il subsume (est inclus dans) n'importe quel ensemble.

Moindre généralisé

Le moindre généralisé doit mettre en évidence ce qu'il y a de commun à plusieurs hypothèses : il s'agit de l'intersection entre ensembles. Il est aisé de montrer que cette intuition correspond bien à la définition de moindre généralisé que nous avons donné.

Preuve

Soit E_i des ensembles et I l'intersection de ces ensembles. Par définition, I est inclus dans chacun des E_i et, pour tout i, on a donc $I \succeq E_i$. Supposons maintenant qu'il existe G telle que G subsume tous les E_i et que I soit plus général que G ($I \succeq G$). Si G subsume tous les E_i alors $G \subseteq \cap_i L_i$, c'est-à-dire $G \succeq I$ et l'on a supposé que $I \succeq G$. Par suite, I et G sont équivalents (et même égaux dans ce langage). □

Ainsi, le moindre généralisé de $(7, 9, 13, 33, 38, 43)$ et $(2, 7, 11, 23, 45, 47)$ est (7).

Toutes les relations de généralité, et en particulier celles que nous allons voir dans la suite, sont essentiellement des relations d'inclusion, tout comme le moindre généralisé ressemblera souvent à une intersection.

1.5.2 Langage attribut-valeur

Dans le langage attribut-valeur, un exemple de $\mathcal{L}_e$ est un ensemble de couples (attribut,valeur). Les attributs $a_1, a_2, \ldots, a_n$ sont présents dans chaque exemple et chaque attribut a_i prend une valeur v_i dans le domaine de l'attribut noté $\mathcal{D}_i$. Par suite, la langage des exemples $\mathcal{L}_e$ est le produit cartésien des domaines $\mathcal{D}_i$.

Une hypothèse reprend les mêmes attributs mais avec une valeur supplémentaire dans chacun des domaines : il s'agit de la valeur `?` indiquant que l'attribut n'est pas jugé pertinent dans l'hypothèse. Pour chaque attribut a_i, une hypothèse opère un test sur sa valeur, l'expression de ce test est nommée *sélecteur*. Ainsi, selon le langage $\mathcal{L}_h$ choisi, des sélecteurs valides sur l'attribut a_i pourront être par exemple

$$\begin{array}{lcll} a_i & = & \texttt{?} & \\ a_i & = & v_i & \text{avec } v_i \in \mathcal{D}_i \\ a_i & \neq & v_i & \text{avec } v_i \in \mathcal{D}_i \\ a_i & \in & [v_{i_1}; v_{i_2}] & \text{avec } v_{i_1}, v_{i_2} \in \mathcal{D}_i \end{array}$$

Que ce soit pour la subsomption ou le moindre généralisé, nous allons opérer attribut par attribut, considérant dans chaque hypothèse le sélecteur qui lui correspond.

Subsomption

Une hypothèse H_1 subsume une hypothèse H_2 si chaque sélecteur de H_1 subsume le sélecteur du même attribut dans H_2 ;

- Si l'attribut est discret, v_1 subsume v_2 si $v_1 = v_2$ ou si v_1 vaut `?`.
- L'attribut peut-être discret mais avec une hiérarchie, dont le sommet est `?`. v_1 subsume v_2 si $v_1 = v_2$, ou si v_1 est un ancêtre de v_2 dans la hiérarchie.
- Pour un attribut continu, les choix sont multiples. Il faut, en tout cas, revenir sur la définition de $\mathcal{L}_h$. On peut par exemple décider que la valeur d'un attribut est un intervalle. Dans un exemple, une valeur v devient $[v; v]$. Dans ce cas, v_1 subsume v_2 si $v_2 \subseteq v_1$.

Moindre généralisé

La généralisation de deux hypothèses est l'hypothèse obtenue en généralisant, pour chaque attribut, les deux sélecteurs correspondant à cet attribut.

Considérons les différentes généralisations de deux sélecteurs d'un même attribut selon le type de cet attribut.

- Si l'attribut est discret et les valeurs de v_1 et v_2 différentes alors leur moindre généralisé est `?`, sinon c'est le même sélecteur qui apparaît dans les deux hypothèses et il est conservé comme moindre généralisé.
- Si l'attribut est discret avec une hiérarchie, le moindre généralisé de deux valeurs est leur premier ancêtre commun dans la hiérarchie.
- Pour un attribut continu, les choix sont multiples. Le moindre généralisé de deux intervalles v_1 et v_2 est, par exemple, le plus petit intervalle qui contient à la fois v_1 et v_2 (cela revient à retenir la plus petite et la plus grande borne de v_1 et v_2).

Nous allons maintenant passer à des langages plus expressifs avec la logique du premier ordre. À noter que nous avons déjà considéré la logique propositionnelle à travers l'attribut-valeur : si les attributs sont tous de type booléen, alors ce langage est équivalent à la logique propositionnelle.

1.5.3 Logique du premier ordre

La logique du premier ordre a été envisagée lorsqu'il est apparu que la description de certaines structures exigeaient un tel langage. Le cas le plus typique est celui des molécules, par exemple lorsqu'il s'agit de caractériser des molécules cancérigènes : dans ce cas, $\mathcal{L}_h$ doit être assez puissant pour décrire des graphes, ce qui n'est pas le cas des langages attribut-valeur. L'exemple des molécules illustrent aussi toute la difficulté de travailler dans ce langage : le test de subsomption entre hypothèses va se ramener à de l'appariement entre graphes !

Ce paradigme a donné naissance à la Programmation Logique Inductive [Muggleton, 1991, Muggleton et De Raedt, 1994], qui se propose en plus d'utiliser au mieux une éventuelle théorie du domaine. C'est à cette branche de l'apprentissage qu'appartient les travaux décrits dans la première partie de cette thèse. Dans ce cas précis, la formalisation du langage est empruntée à la Programmation Logique [Lloyd, 1987]. Les concepts de base qui vont nous servir dans les chapitres suivants, essentiellement les clauses et les substitutions, sont rappelés en annexe, section A.1.

Le langage $\mathcal{L}_h$ sera un ensemble de clauses définies, engendrées par des constantes, des variables, des symboles de fonction et des symboles de prédicat, tous en nombre fini sauf dans le cas des variables. Nous restreindrons $\mathcal{L}_h$ aux clauses dont la tête a pour symbole de prédicat le concept que l'on cherche à caractériser.

Subsomption

La plus souhaitable des relations de subsomption entre clauses serait l'implication logique. Malheureusement, cette relation entre clauses est indécidable dans le cas général [Schmidt-Schauss, 1988], et le reste même si nous nous restreignons aux clauses de Horn [Marcinkowski et Pacholski, 1992].

Si bien qu'il a fallu proposer d'autres relations qui, elles, sont traitables. La définition de la θ-subsomption dans [Plotkin, 1970] marque sans doute la naissance de la Programmation Logique Inductive même si la création officielle de ce domaine de recherche ne devait survenir que dans [Muggleton, 1991].

Définition 1.6 (θ-subsomption)
On dit que C θ-subsume D ($C \succeq D$) si et seulement s'il existe une substitution θ telle que $C\theta \subseteq D$.

Cette relation de généralité entre les hypothèses est la plus utilisée mais NP-difficile [Kapur et Narendran, 1986]. Cependant, elle est équivalente à l'implication logique sur les clauses non récursives [Gottlob, 1987].

Dans les langages précédents, l'équivalence entre deux hypothèses signifiaient leur égalité. Ici, deux clauses peuvent être équivalentes en étant pourtant différentes : soit elles sont identiques modulo un renommage de variables, soit au moins l'une des deux n'est pas réduite.

Définition 1.7 (clause non réduite)
Une clause C est dite non réduite s'il existe une substitution θ telle que $C\theta \subset C$. On dit alors que $C\theta$ est un facteur subsumant de C.

Exemple 1.1 (clauses équivalentes)
Considérons les trois clauses suivantes :

$$C_1 : c(X) \leftarrow p(X, s(0))$$
$$C_2 : c(X) \leftarrow p(X, s(0)), p(A, B)$$
$$C_3 : c(X) \leftarrow p(X, s(0)), p(C, s(D)), p(X, E)$$

C_1 est réduite, tandis que C_2 et C_3 ne le sont pas : en leur appliquant respectivement les substitutions

$$\theta_2 = \{A/X; s(0)/B\}$$
$$\theta_3 = \{C/X; 0/D; s(0)/E\}$$

nous retrouvons C_1. Ces trois clauses sont donc équivalentes.

C'est la θ-subsomption que nous adopterons dans la suite. Citons cependant d'autres définitions qui ont été introduites pour diverses raisons : prise en compte de la théorie du domaine, complexité plus faible, etc.

— la subsomption relative de [Plotkin, 1971] ;
— l'utilisation de la SLD-résolution par [Muggleton et Buntine, 1988] ;
— la subsomption généralisée de [Buntine, 1988] ;
— la T-implication de [Idestam-Almquist, 1995] ;
— la subsomption empirique [Champesme et al., 1995b].

Ajoutons à la liste, la subsomption stochastique de [Sebag et Rouveirol, 1997], basée sur la θ-subsomption de Plotkin et qui utilise la notion de substitutions *compatibles* : nous dirons que deux substitutions sont compatibles si elles n'assignent pas une même variable à deux objets différents. Détaillons cet algorithme qui prend en entrée deux clauses C et D, et qui doit déterminer si C θ-subsume D. Pour chaque littéral L de C, l'ensemble des substitutions qui permettent d'envoyer L sur un littéral de D doit être construit. L'algorithme est le suivant. Pour chaque littéral de C, on choisit aléatoirement l'une de ses substitutions ; cette substitution doit être compatible avec les substitutions choisies précédemment ; si cela n'est pas le cas, on revient au premier littéral pour une nouvelle tentative. L'algorithme peut s'arrêter dans deux cas : soit on est parvenu à trouver une substitution compatible pour chaque littéral, auquel cas C θ-subsume D, soit le nombre maximal tentatives a été atteint sans trouver de telles substitutions et C ne θ-subsume pas D.

Lorsque cette procédure indique que C θ-subsume D, c'est bien le cas et nous disposons même de la substitution correspondante ; en revanche, il est possible qu'elle indique que C ne θ-subsume pas D alors qu'il existe bien une substitution θ telle que $C\theta \subseteq D$.

Moindre généralisé

Venons-en maintenant aux moindres généralisés en logique d'ordre un. Contre toute attente, il a été montré récemment que le moindre généralisé sous implication

logique est défini [Nienhuys-Cheng et de Wolf, 1996b, Nienhuys-Cheng et de Wolf, 1996a], même si l'algorithme proposé est de complexité exponentielle.

Sous θ-subsomption, un algorithme polynomial existe, qui permet de calculer le moindre généralisé de plusieurs clauses. Cet algorithme a été fourni par Plotkin, en même temps que la définition de la θ-subsomption [Plotkin, 1970]. Dans ce même article, il indiquait aussi les limitations de ce calcul : la clause construite est souvent non réduite (c'est-à-dire qu'elle est équivalente à l'un de ses sous-ensembles). Plus précisément, la taille du moindre généralisé est le produit des tailles des clauses dans l'ensemble de départ. Or nous avons vu que décider si une clause en θ-subsume une autre est un problème NP-difficile ; par suite, il ne sera pas possible d'augmenter indéfiniment un moindre généralisé tout en l'utilisant dans des tests de subsomption.

Variations sur l'ordre un

Nous avons pu observer deux tendances diamétralement opposées en Programmation Logique Inductive.

1. Les premiers estiment que le langage est trop puissant et que les coûts de traitement de ces langages sont inacceptables. Par exemple, l'*identité d'objets* [Esposito et al., 1996] posent que deux variables distinctes dans une clause font référence à des objets différents. La θ-subsomption en est simplifiée puisque les substitutions doivent être injectives. Cependant, déterminer si une hypothèse en θ-subsume une autre, reste un problème NP-complet. Pour s'en convaincre, il suffit de reprendre la preuve de [Kapur et Narendran, 1986] qui s'adapte parfaitement à ce nouveau cas. De plus, l'identité d'objets induit plusieurs moindres généralisés pour un ensemble d'hypothèses.

2. Les seconds, au contraire, pensent qu'un pas de plus est nécessaire dans l'expressivité : certains concepts nécessitent un langage plus riche que les clauses de Horn. [Goncalves, 1996, Goncalves, 1999] indique qu'il est intéressant de quantifier universellement certaines variables du corps. Dans le formalisme proposé, une séquence de quantificateurs, $\forall$ ou $\exists$, portant sur les variables n'apparaissant pas dans la tête, est introduite avant les littéraux du corps. Dans le même ordre d'idée, [Martin, 1996] propose d'utiliser la négation dans le corps des clauses et [De Raedt et Bruynooghe, 1993] étendent leur langage aux programmes normaux.

1.6 Évaluation de l'apprentissage

Il est important de signaler maintenant que l'apprentissage consiste à découvrir la définition d'un concept, mais aussi à évaluer la confiance que l'on peut accorder à cette définition. C'est pourquoi, outre les ensembles d'exemples A^+ et A^- qui seront utilisés pendant la phase d'apprentissage, il est courant de se donner deux autres ensembles, notés T^+ et T^-, servant à tester la définition du concept apprise.

Les langages et ensembles intervenant dans un problème d'apprentissage sont

rappelés à la Table 1.2. Ces ensembles doivent vérifier :

$$A^+ \cup T^+ \subseteq \mathcal{L}_e^+$$
$$A^- \cup T^- \subseteq \mathcal{L}_e^-$$

et, de plus,

$$A^+ \cap T^+ = \emptyset$$
$$A^- \cap T^- = \emptyset$$

En effet, il est tout simplement *fair-play* qu'il n'y ait pas d'exemples qui soient à la fois utilisés pour apprendre et pour évaluer le résultat de cet apprentissage. À ce sujet [Scheffer et Herbrich, 1997] ont évalué le gain obtenu en apprenant sur les ensembles d'apprentissage, mais en réglant les paramètres du système en fonction de la qualité des prédictions sur les ensembles de test. Pour notre part, il est bien clair que les ensembles de test n'interviennent d'aucune façon durant l'apprentissage.

1.6.1 Mesure de précision

Nous supposons que le résultat de l'apprentissage effectué sur A^+ et A^- nous permet d'affecter une classe, + ou −, à n'importe quel nouvel exemple. Nous l'utilisons donc pour étiqueter les exemples de l'ensemble de test et obtenons alors une partition de cet ensemble en deux sous-ensembles, C^+ et C^-, respectivement les exemples estimés positifs et négatifs. Ces ensembles vérifient :

$$C^+ \cup C^- = T^+ \cup T^-$$
$$C^+ \cap C^- = \emptyset$$

À l'aide de ces prédictions, nous pouvons mesurer la précision de l'apprentissage.

Définition 1.8 (précision)
La précision est le pourcentage d'exemples bien classifiés.

$$\textit{précision}\ (C^+, C^-, T^+, T^-) = \frac{|C^+ \cap T^+| + |C^- \cap T^-|}{|T^+| + |T^-|}$$

Et de manière duale, l'erreur est définie comme le pourcentage d'exemples mal classifiés.

TABLE 1.2 – Ensembles pour l'apprentissage

$\mathcal{L}_e$	langage des exemples
$\mathcal{L}_h$	langage des hypothèses
$\mathcal{L}_e^+$	exemples du concept
$\mathcal{L}_e^-$	contre-exemples du concept
A^+	exemples d'apprentissage
A^-	contre-exemples d'apprentissage
T^+	exemples de test
T^-	contre-exemples de test

Définition 1.9 (erreur)

$$\textit{erreur}\ (C^+, C^-, T^+, T^-) = \frac{|C^+ \cap T^-| + |C^- \cap T^+|}{|T^+| + |T^-|}$$

ou encore

$$\textit{erreur}\ (C^+, C^-, T^+, T^-) = 1 - \textit{précision}\ (C^+, C^-, T^+, T^-)$$

Un autre moyen de visualiser la couverture des ensembles de test par le résultat de l'apprentissage est de constituer une *table de contingence* comme suit :

	estimés positifs	estimés négatifs
réellement positifs	a	b
réellement négatifs	c	d

Nous y retrouvons les quantités sur lesquelles nous avons travaillé pour définir la précision :

$$a = |C^+ \cap T^+|$$
$$d = |C^- \cap T^-|$$

Classiquement, ces deux quantités sont respectivement nommées la *spécificité* et *sensibilité* de la définition. Nous verrons, section 7.3.4 (page 123), des problèmes pour lesquels la spécificité et la sensibilité sont des mesures plus pertinentes que la précision elle-même.

Ces mesures sont bien sûr influencées par les exemples choisis pour le test. Ce choix, quand bien même serait-il aléatoire, peut avantager la définition apprise, ou au contraire la désavantager.

Il s'agit donc maintenant d'obtenir une évaluation objective d'un système par rapport à un problème et, plus particulièrement, de pouvoir départager deux systèmes sur un problème donné.

1.6.2 Constitution des ensembles de test

À partir de toutes les données étiquetées disponibles, il s'agit de construire les ensembles (A^+, A^-, T^+, T^-), disjoints deux à deux. Nous nous autorisons même à recommencer plusieurs fois cette construction, tout cela dans le but d'obtenir une évaluation significative de l'algorithme d'apprentissage.

La procédure la plus simple consiste à partager aléatoirement l'ensemble des exemples disponibles, à hauteur de 70% pour l'apprentissage et 30% pour le test, par exemple. Cette partition et l'apprentissage sont renouvelés 10 ou 20 fois, puis la moyenne des précisions obtenues est calculée. D'autres mesures peuvent être effectuées comme les précisions minimale et maximale, l'écart-type, etc.

La *validation croisée* consiste à couper aléatoirement l'ensemble de données en plusieurs fragments de même taille. Puis, successivement, un de ces fragments est utilisé comme test et l'union des autres comme l'ensemble d'apprentissage.

Cette méthode a donc deux paramètres : le nombre de coupures et le nombre d'itérations. Le nombre d'apprentissage à effectuer est le produit de ces deux valeurs ; ensuite, il est possible comme précédemment, de calculer moyenne, écarts, etc.

Le cas le plus simple est celui où l'on croise deux fois seulement. Les exemples sont partagés en deux sous-ensembles de même taille. Le premier sert pour l'apprentissage, le second pour le test, puis l'inverse.

Pour un exposé plus détaillé de ces méthodes, on consultera par exemple le chapitre dédié à l'évaluation des hypothèses dans [Mitchell, 1997].

1.6.3 Le test *5×2cv*

Thomas Dietterich a montré que ces mesures pouvaient ne pas être significatives. En particulier lorsqu'un système A obtient une meilleure précision moyenne sur un problème qu'un système B, [Dietterich, 1998] a montré expérimentalement qu'il n'est pas toujours légitime de penser que A est meilleur que B sur le problème considéré.

Pour remédier à ce problème, Thomas Dietterich propose, toujours dans [Dietterich, 1998], un test nommé *5x×2cv* dont il montre expérimentalement qu'il est plus rigoureux. Ce test permet de déterminer lequel de deux systèmes est le meilleur sur un problème particulier. Comme son nom l'indique, cette méthode est basée sur une validation croisée deux fois, itérée cinq fois, et cela pour chacun des deux algorithmes en compétition. À l'issue, nous obtenons un pourcentage représentant la confiance avec laquelle nous pouvons dire qu'un système est meilleur que l'autre. Les détails de cette procédure et des calculs à effectuer sont donnés en annexe, section A.2.

1.6.4 Compréhensibilité de la solution

Ainsi, nous allons rechercher une bonne précision, c'est-à-dire une bonne prédiction de la classe d'exemples non rencontrés lors de l'apprentissage.

Une autre préoccupation majeure est la *compréhensibilité* : les prédictions doivent pouvoir être justifiées, expliquées et appréhendables par un expert humain du domaine.

Nous rejoignons ici l'esprit du test de Turing : pour être considéré comme intelligent, il faut pouvoir discuter et se faire comprendre.

Malheureusement, cette qualité est plus subjective et ne peut pas être évaluée aussi simplement que la précision de la définition. Le plus souvent cette tâche sera laissée à l'expert humain.

Le meilleur moyen de tendre vers une parfaite compréhensibilité de la solution est de bien choisir le langage de hypothèses $\mathcal{L}_h$. Ce choix de $\mathcal{L}_h$ est donc crucial de ce point de vue et à nouveau c'est l'expert humain qui sera mis à contribution.

À plusieurs reprises, nous avons déjà pu mesurer l'influence du langage $\mathcal{L}_h$ sur l'apprentissage : c'est lui qui décide de l'existence d'une solution, de la nécessité d'une solution disjonctive et maintenant de la compréhensibilité. À noter que dans le cas d'un apprentissage disjonctif, la compréhensibilité passera aussi par une taille raisonnable de la solution : celle-ci devra, pour couvrir A^+, utiliser un nombre minimal d'hypothèses de $\mathcal{L}_h$. C'est là le point de vue que nous développerons dans la seconde partie de cette thèse consacrée à l'apprentissage disjonctif.

Certains ont jugé le pouvoir de $\mathcal{L}_h$ trop important et ont préféré négliger la production d'une définition intelligible. Citons, parmi beaucoup d'autres :

— les méthode de type *plus proches voisins* ;

— les réseaux de neurones ;
— les machines à support de vecteurs [Schölkopf et al., 1998] ;
— les comités d'experts [Nok et O. Gascuel, 1995] ;
— les classifieurs qui procèdent par vote d'un très grand nombre de définitions, par exemple le système Monkey [Sebag, 1999b] ;
— la classification par portée [Lachiche, 1997].

Tout en étant toujours capables de classifier de nouveaux exemples, ces systèmes ont perdu la possibilité d'exhiber une définition compréhensible du concept qui induirait cette classification. Cependant, il semblerait que, pour certains, cette perte de compréhensibilité soit compensée par un gain significatif de précision.

1.7 No Free Lunch Theorem

Y a pas d'cheval qu'on peut pas monter,
mais y a pas d'homme qu'il peut pas jeter.
Alors quand tu voudras monter un mustang,
oublie pas qu' c'est p't-être pas toi qui tiendras d'ssus.
Marion Zimmer Bradley, Redécouverte, page 303.

Nous savons maintenant évaluer une méthode sur un problème particulier et aussi déterminer laquelle de deux méthodes est la meilleure, toujours sur un problème précis.

Qu'en est-il dans l'absolu ? Certaines méthodes sont-elles intrinsèquement meilleures que d'autres ? Autrement dit, existe-t-il des méthodes qui soient meilleures que d'autres, quel que soit le problème considéré ?

Un élément de réponse a été apporté par le *No Free Lunch Theorem* [Wolpert et Macready, 1995]. Il s'agit d'un résultat général pour les algorithmes de recherche d'un optimum. Ce théorème est adaptable à l'apprentissage [Schaffer, 1994] et est connu dans ce cas sous le nom de *Selective superiority problem.*

Il indique que, quelle que soit la méthode utilisée, la moyenne des précisions obtenues sur l'ensemble des problèmes possibles est de 0.5. Toujours en moyenne, toute méthode est donc équivalente à n'importe quelle autre et, en particulier à un tirage aléatoire pour le classement de nouveaux exemples.

Donnons rapidement l'intuition de la preuve de ce résultat. Considérons un problème d'apprentissage donné par $(\mathcal{L}_e, \mathcal{L}_h, A^+, A^-, \succeq)$ et soient (T^+, T^-), les ensembles utilisés pour le test.

Construisons maintenant un nouveau problème. (A^+, A^-) restent les ensembles fournis à la méthode pour l'apprentissage. En revanche, les étiquettes des exemples non vus pendant l'apprentissage sont inversées : T^+ devient T^- et T^- devient T^+.

Puisque A^+ et A^- sont inchangés le classifieur construit par la méthode reste le même que dans le premier cas et induit donc la même partition (C^+, C^-) sur l'ensemble de test ; en revanche, la précision va se trouver modifiée par notre changement d'étiquettes.

En gardant T^+ et T^- pour dénoter les ensembles de test du premier problème, il vient :

$$\text{précision}(C^+, C^-, T^-, T^+) = \frac{|C^+ \cap T^-| + |C^- \cap T^+|}{|T^+| + |T^-|}$$

En utilisant les propriétés de nos ensembles, nous avons :

$$\begin{aligned} |C^+ \cap T^-| &= |C^+ \cap T| - |C^+ \cap T^+| \\ &= |C^+| - |C^+ \cap T^+| \end{aligned}$$

et, de même,

$$|C^- \cap T^+| = |C^-| - |C^- \cap T^-|$$

Par suite,

$$\text{précision}(C^+, C^-, T^-, T^+) = \frac{|C^+| + |C^-|}{|T^+| + |T^-|} - \frac{|C^+ \cap T^+| + |C^- \cap T^-|}{|T^+| + |T^-|}$$

ou encore,

$$\text{précision}(C^+, C^-, T^-, T^+) = 1 - \text{précision}(C^+, C^-, T^+, T^-)$$

Et nous obtenons finalement une précision moyenne sur ces deux problèmes de 0.5. Comme on peut construire un pareil dual pour n'importe quel problème, nous sommes contraints de conclure que la précision moyenne sur l'ensemble des problèmes est aussi de 0.5.

Ce résultat semble donc indiquer qu'il ne peut pas exister de méthode qui obtienne de bons résultats sur tous les problèmes. Nous reviendrons à plusieurs reprises sur cette conclusion négative pour soulever quelques objections. En attendant, nous allons évoquer quelques problèmes réels sur lesquels l'apprentissage automatique a obtenu d'excellents résultats.

1.8 Problèmes réels

Comme nous l'avons dit, la motivation première de l'apprentissage automatique était de dépasser le goulot d'étranglement de l'IA, et cela en acquiérant la connaissance d'un expert sans que celui-ci ait besoin de l'exprimer explicitement.

Durant la dernière décennie, il est apparu que l'apprentissage automatique pouvait exceller hors de ce cadre, dans des domaines où les experts humains ne possèdent pas, même implicitement, la connaissance à découvrir. Ces nouveaux problèmes peuvent être scindés en deux familles :

- d'une part, il y a les problèmes dont les données sont de taille trop importante pour envisager autre chose qu'un traitement automatique ;
- d'autre part, il y a les problèmes qui tiennent de la découverte scientifique ; dans ce cas, les données sont de taille relativement raisonnables mais les scientifiques humains ne sont pas parvenus à bâtir une théorie expliquant les données.

La reconnaissance d'objets stellaires devient un problème de la première catégorie lorsque le nombre de photographies devient conséquent.

C'est le cas pour les observatoires impliqués dans des projets de cartographie complète du ciel : un nombre considérable de photographies du ciel est disponible, chacune avec une précision tout aussi considérable.

C'est à un tel problème que s'attaque le système Skycat [Fayyad et al., 1993] : environ trois tera octets de données sont disponibles et une estimation indique que plusieurs dizaines d'années-hommes seraient nécessaires pour dépouiller ces photographies.

Skycat a appris des arbres de décision qui ont finalement permis de traiter les images du ciel et le résultat a été jugé pertinent par les experts : un catalogue produit par Skycat est maintenant d'utilisation courante parmi les astrophysiciens.

The Predictive Toxicology Evaluation Challenge est un concours initié en 1997 [Srinivasan et al., 1997], invitant les chercheurs en apprentissage automatique à participer à une découverte scientifique en chimie. Dans ce problème, nous disposons de quelques centaines de molécules dont le caractère cancérigène ou non cancérigène a été établi par des expérimentations menées sur des rats et des souris. Une telle expérience pour une unique molécule est bien trop longue (de l'ordre de deux ans) pour que l'on envisage de l'appliquer à l'ensemble des molécules apparaissant chaque année dans notre environnement.

Il s'agit donc d'utiliser les résultats obtenus sur quelques molécules pour apprendre à reconnaître les molécules dangereuses. Signalons une semblable application, un peu plus ancienne : le problème de la *mutagénèse* [Srinivasan et al., 1994].

La contribution de la communauté d'apprentissage automatique est significative : les règles obtenues parviennent à de meilleures prédictions que les meilleurs experts humains. Cependant, les définitions apprises manquent de clarté pour les chimistes et c'est pourquoi le concours a été reconduit cette année [Srinivasan et al., 1999].

Nous nous sommes intéressés à ce problème et y avons appliqué le système GloBo que nous décrirons au chapitre 7 (page 113). Les résultats obtenus sur ce problème seront analysés à la section 7.3 (page 120).

1.9 Bilan

Nous avons commencé ce chapitre en montrant la nécessité de l'apprentissage pour l'Intelligence Artificielle.

Nous avons ensuite discuté des langages intervenant en cours d'apprentissage. Nous avons vu les les opérations importantes sur ces langages comme la subsomption et le calcul de moindre généralisé, ainsi que l'influence de ces langages : selon le langage choisi pour les hypothèses, une solution disjonctive pourra être nécessaire. C'est le problème auquel nous consacrerons la seconde partie de cette thèse.

Puis, nous avons donné des exemples précis de langages permettant de représenter exemples et hypothèses. En particulier, nous avons présenté la Programmation Logique Inductive qui travaille avec des clauses définies ; ce paradigme sera adopté pour la première partie de cette thèse.

Nous avons discuté de l'évaluation des solutions, des critères pertinents (essentiellement la précision et la compréhensibilité) et des moyens de les mesurer. Nous avons détaillé le *No Free Lunch Theorem*, résultat montrant que tous les systèmes

sont, sur les ensembles des problèmes possibles, équivalents.

Enfin, nous avons terminé en montrant que les systèmes d'apprentissage s'appliquaient d'ores et déjà brillament à des problèmes réels.

Bibliographie

[Buntine, 1988] Buntine, W. (1988). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36(2) :375–399.

[Champesme et al., 1995] Champesme, M., Brézellec, P., et Soldano, H. (1995). Réduction de l'espace de recherche en apprentissage : résultats théoriques et expérimentaux. In Nicolas, J., éditeur, *Dixièmes Journées Francophones sur l'Apprentissage*, pages 65–84.

[Cohen et Feigenbaum, 1982] Cohen, P. R. et Feigenbaum, E. A. (1982). *The Handbook of Artificial Intelligence*, volume 3. HeurisTech Press and William Kaufmann, Stanford, California and Los Altos, California.

[Cornuéjols et Moulet, 1997] Cornuéjols, A. et Moulet, M. (1997). Machine learning : a survey. In Tzafestas, S., éditeur, *Knowledge based sytems*, chapitre 2, pages 61–86. World Scientific.

[De Raedt et Bruynooghe, 1993] De Raedt, L. et Bruynooghe, M. (1993). A theory of clausal discovery. In Bajcsy, R., éditeur, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7) :1895–1924.

[Esposito et al., 1996] Esposito, F., Laterza, A., Malerba, D., et Semeraro, G. (1996). Refinement of datalog programs. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD)*, pages 73–94.

[Fayyad et al., 1993] Fayyad, U. M., Weir, N., et Djorgovski, S. (1993). SKICAT : A machine learning system for automated cataloging of large scale sky surveys. In Utgoff, P., éditeur, *Proceedings 10th International Conference on Machine Learning*, pages 112–119. Morgan Kaufmann.

[Feigenbaum, 1977] Feigenbaum, E. A. (1977). The art of artificial intelligence : Themes and case studies of knowledge engineering. In Reddy, R., éditeur, *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 1014–1029. Morgan Kaufmann.

[Goncalves, 1996] Goncalves, M.-E. (1996). Handling quantifiers in ILP. In Muggleton, S., éditeur, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 76–96. Stockholm University, Royal Institute of Technology.

[Goncalves, 1999] Goncalves, M.-E. (1999). *Vers des opérateurs d'apprentissage pour la logique du premier ordre - Étude de leurs propriétés de correction, complétude et minimalité.* Thèse de doctorat, Université Paris-Sud.

[Gottlob, 1987] Gottlob, G. (1987). Subsumption and implication. *Information Processing Letters*, 24(2) :109–111.

[Idestam-Almquist, 1995] Idestam-Almquist, P. (1995). Generalization of clauses under implication. *Journal of Artificial Intelligence Research*, 3 :467–489.

[Kapur et Narendran, 1986] Kapur, D. et Narendran, P. (1986). Np-completeness of the set unification and matching problems. In *Proceedings of 8th Conference on Automated Deduction*, volume 230, pages 489–495. Springer-Verlag.

[Kodratoff, 1989] Kodratoff, Y. (1989). Characterising machine learning programs, a european compilation. Rapport interne 507, Laboratoire de Recherche en Informatique, Université Paris Sud.

[Lachiche, 1997] Lachiche, N. (1997). *De l'induction confirmatoire à la classification : contribution à l'apprentissage automatique*. Thèse de doctorat, Université Henri Poincaré - Nancy 1.

[Levesque et Brachman, 1989] Levesque, H. J. et Brachman, R. J. (1989). A fundamental tradeoff in knowledge representation and reasoning (revised version). In Brachman et Levesque, éditeurs, *Readings in knowledge representation*, chapitre 4, pages 41–70. Morgan Kaufmann.

[Lloyd, 1987] Lloyd, J. W. (1987). *Foundations of Logic Programming*. Springer, Berlin, 2 edition.

[Marcinkowski et Pacholski, 1992] Marcinkowski, J. et Pacholski, L. (1992). Undecidability of the horn-clause implication problem. In IEEE, éditeur, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 354–362, Pittsburgh, PN. IEEE Computer Society Press.

[Martin, 1996] Martin, L. (1996). *Induction de programmes logiques avec négation*. Thèse de doctorat, Université d'Orléans.

[Minsky, 1988] Minsky, M. (1988). *La société de l'esprit*. InterEditions, Paris.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18 :203–226.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

[Muggleton, 1991] Muggleton, S. (1991). Inductive logic programming. *New Generation Computing Journal*, 8(4) :295–317.

[Muggleton et De Raedt, 1994] Muggleton, S. et De Raedt, L. (1994). Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19 :629–679.

[Muggleton et Buntine, 1988] Muggleton, S. H. et Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings 5th International Conference on Machine Learning*, pages 339–352, San Mateo, CA. Morgan Kaufmann.

[Nienhuys-Cheng et de Wolf, 1996a] Nienhuys-Cheng, S. et de Wolf, R. (1996a). Least generalizations and greatest specializations of sets of clauses. *Journal of Artificial Intelligence Research*, 4 :341–363.

[Nienhuys-Cheng et de Wolf, 1996b] Nienhuys-Cheng, S. et de Wolf, R. (1996b). Least Generalizations under Implication. In Muggleton, S., éditeur, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 262–363. Stockholm University, Royal Institute of Technology.

[Nok et O. Gascuel, 1995] Nok, R. et O. Gascuel, O. (1995). On learning decision committees. In Prieditis, A. et Russell, S., éditeurs, *Proceedings 12th International Conference on Machine Learning*, pages 413–420. Morgan Kaufmann.

[Plotkin, 1970] Plotkin, G. (1970). A note on inductive generalization. In Meltzer, B. et Mitchie, D., éditeurs, *Machine Intelligence*, volume 5, pages 153–165. Edinburgh University Press.

[Plotkin, 1971] Plotkin, G. (1971). A further note on inductive generalization. In *Machine Intelligence*, volume 6. Edinburgh University Press.

[Schaffer, 1994] Schaffer, C. (1994). A conservation law for generalization performance. In Cohen, W. W. et Hirsh, H., éditeurs, *Proceedings 11th International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann.

[Scheffer et Herbrich, 1997] Scheffer, T. et Herbrich, R. (1997). Unbiased assessment of learning algorithms. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 798–803.

[Schmidt-Schauss, 1988] Schmidt-Schauss, M. (1988). Implication of clauses is undecidable. *TCS : Theoretical Computer Science*, 59(3) :287–296.

[Schölkopf et al., 1998] Schölkopf, B., Burgess, C., et Smola, A. (1998). *Advances in Kernel Methods*. MIP Press.

[Sebag, 1999] Sebag, M. (1999). Constructive induction : A version space-based approach. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

[Sebag et Rouveirol, 1997] Sebag, M. et Rouveirol, C. (1997). Tractable induction and classification in FOL via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–892.

[Srinivasan et al., 1999] Srinivasan, A., King, R., et Bristol, D. (1999). An assessment of submissions made to the predictive toxicology evaluation challenge. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 270–276. Morgan Kaufmann.

[Srinivasan et al., 1997] Srinivasan, A., King, R. D., Muggleton, S. H., et Sternberg, M. J. E. (1997). The predictive toxicology evaluation challenge. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 4–9. Morgan Kaufmann.

[Srinivasan et al., 1994] Srinivasan, A., Muggleton, S., King, R. D., et Sternberg, M. J. E. (1994). Mutagenesis : ILP experiments in a non-determinate biological domain. In Wrobel, S., éditeur, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH.

[Turing, 1950] Turing, A. M. (1950). Computing machinery and Intelligence. *Mind*, 59 :433–460.

[Wolpert et Macready, 1995] Wolpert, D. H. et Macready, W. G. (1995). No free lunch theorems for search. Rapport interne SFI-TR-95-02-010, The Santa Fe Institute.

CHAPITRE 2

Biais et générer-et-tester

D'une part, l'algorithme *générer-et-tester* est une méthode de l'Intelligence Artificielle communément associé à la résolution des problèmes de recherche. D'autre part, de multiples informations sur la solution cherchée, autres que ses propriétés de correction et de complétude, sont souvent disponibles. Ces connaissances a priori sont nommées *biais*.

Dans ce chapitre, nous présentons ces deux entités en commençant par discuter l'utilisation des biais en apprentissage : de leur nature, de leur nécessité et de leur justification. Pour cela, nous nous référons au travail de Tom Mitchell [Mitchell, 1980].

Nous décrivons ensuite l'algorithme générer-et-tester, en reprenant cette fois-ci [Mitchell, 1982] pour montrer que cet algorithme est adapté à la résolution d'un problème d'apprentissage.

La fin du chapitre est consacrée au générer-et-tester appliqué à la Programmation Logique Inductive (PLI). Tout d'abord, nous évoquons les biais qui sont communément utilisés en PLI. Puis, nous terminons avec les opérateurs de raffinement utilisés en PLI et, plus particulièrement, avec les caractérisations des opérateurs *optimaux* [De Raedt et Bruynooghe, 1993] et des opérateurs *idéaux* [van der Laag et Nienhuys-Cheng, 1994b].

2.1 Biais et apprentissage

2.1.1 Saut inductif

L'imagination est plus importante que le savoir.
Albert Einstein.

Le but de l'apprentissage est de pouvoir décider si un exemple, jamais vu auparavant, appartient au concept étudié ou non. Pour cela, nous avons à notre disposition, les ensembles d'exemples A^+ et A^-. Leur disjonction constitue ce que nous avons décrit à la section 1.4 comme un apprentissage par cœur, et ne disent rien sur la classe d'un nouvel exemple : il faut impérativement généraliser les exemples positifs de A^+, en évitant de couvrir les contre-exemples de A^-.

Cette généralisation est appelée un *saut inductif*, mais ce saut est périlleux. Il revient, en quelque sorte, à *imaginer* l'ensemble du concept en n'observant qu'une partie de ce concept. Le mot *imaginer* peut paraître déplacé dans un cadre scientifique, pourtant ce choix se justifie puisqu'il reflète toute la difficulté de fonder la généralisation.

Le *No Free Lunch Theorem*, que nous avons présenté section 1.7, indique même que cette tâche est impossible : quelle que soit la méthode de généralisation, le taux de bonnes prédictions sera en moyenne de 0.5 sur l'ensemble des problèmes possibles. Si bien que toute méthode de généralisation est aussi fondée que le classement aléatoire des nouveaux exemples.

Laissons de côté, pour l'instant, ce résultat négatif pour discuter du rôle des biais dans le saut inductif.

2.1.2 Nécessité des biais

The power of a generalization system follows directly from its biases.
Tom Mitchell.

Avoir tous les avis, c'est n'en avoir aucun.
Françoise Hardy, horoscope du 25 mai 1998, RFM.

Tom Mitchell définit un biais comme tout moyen de préférer une généralisation à une autre, chacune d'elles étant correcte et complète vis-à-vis de A^+ et A^-. En d'autres mots, un biais nous permet de choisir entre deux généralisations qui sont équivalentes pour le problème d'apprentissage tel que nous l'avons posé à la définition 1.1.

D'après Tom Mitchell, les biais peuvent opérer à deux niveaux :

— sur le langage des hypothèses, en fixant la forme que devra avoir la solution ;
— sur la procédure de recherche d'une généralisation satisfaisante dans le langage des hypothèses, en indiquant comment cette recherche doit opérer.

Pour se convaincre de l'intérêt des biais et plus encore, de leur absolue nécessité, reprenons la démonstration qu'en a donnée Tom Mitchell dans [Mitchell, 1980], en construisant un système non biaisé, c'est-à-dire dont les deux niveaux évoqués ci-dessus sont exempts de biais.

— Tout d'abord, il nous faut un langage $\mathcal{L}_h$ permettant de représenter les hypothèses qui soit absolument sans biais : pour cela, ce langage doit permettre de caractériser n'importe quelle partition du langage des exemples $\mathcal{L}_e$ en deux sous-ensembles, C^+ et C^-. Dans la terminologie de Vapnik [Vapnik, 1995], nous voulons que $\mathcal{L}_h$ vaporise $\mathcal{L}_e$. De plus, nous voulons définir un tel $\mathcal{L}_h$ sans pour autant poser quoi que ce soit sur la forme des hypothèses. Pour cela, nous choisissons de voir une hypothèse comme l'ensemble C^+ des exemples qu'elle couvre ; ainsi, chaque sous-ensemble du langage des exemples $\mathcal{L}_e$ est une hypothèse.

— L'étape suivante consiste à trouver une procédure de recherche sans biais, c'est-à-dire qui ne favorise aucune généralisation en particulier. Il nous faut donc construire l'ensemble de toutes les généralisations des exemples positifs qui ne couvrent pas de négatifs, ensemble que Tom Mitchell nomme *espace des versions* [Mitchell, 1977]. Dans notre langage des hypothèses, une hypothèse appartient à l'espace des versions si et seulement si elle inclut tous les exemples positifs de A^+, et aucun des exemples négatifs de A^-. Les hypothèses de l'espace des versions se distinguent les unes des autres par les exemples étrangers à l'ensemble d'apprentissage qu'elles contiennent ou non. L'espace des versions $\mathcal{E}$ est donc défini par

$$\mathcal{E} = \left\{ H \mid H = A^+ \cup E \text{ où } E \subseteq \mathcal{L}_e - (A^+ \cup A^-) \right\}$$

Voyons maintenant comment il est possible de classifier de nouveaux exemples à partir de cet espace des versions, en prenant toujours garde à ne pas introduire de biais.

— Comme nous nous refusons à choisir une des généralisations candidates, nous ne pouvons garantir qu'un exemple est positif que s'il est couvert par *toutes* les généralisations de l'espace des versions. Or, si un exemple apparaît dans chacune des hypothèses de l'espace des versions, cela ne peut signifier qu'une chose : l'exemple est un exemple positif de l'ensemble d'apprentissage ; en effet :

$$\bigcap_{H_i \in \mathcal{E}} H_i = A^+$$

— De même, un exemple sera étiqueté négatif s'il n'est couvert par *aucune* des généralisations de l'espace des versions. À nouveau, cela n'est possible que si l'exemple était connu comme négatif au moment de l'apprentissage car :

$$\mathcal{L}_e - \bigcup_{H_i \in \mathcal{E}} H_i = A^-$$

— Dans les autres cas, les généralisations de l'espace des versions ne sont pas unanimes et nous ne pouvons pas trancher sans introduire de biais : l'exemple reste de classe inconnue.

Ceci montre, très simplement, qu'un apprentissage sans biais ne permet de classifier que les exemples dont la classe est déjà connue. Autrement dit, sans autre information que les exemples étiquetés, il ne peut pas y avoir de saut inductif et donc pas d'apprentissage.

Finalement, il n'y a pas d'autre choix que de biaiser l'apprentissage, et ce que nous avons nommé *imagination* passe par ces biais.

2.1.3 Justification des biais

L'imagination, ce n'est pas le mensonge.
Daniel Pennac, Messieurs les enfants.

Maintenant qu'il est établi que les biais doivent être utilisés, reste à déterminer quels sont les biais disponibles et, parmi ces derniers, identifier ceux qui ont un sens pour le problème considéré. Tom Mitchell identifie plusieurs biais *justifiables* :

— Tout d'abord, il y a les connaissances liées à un problème, à un domaine particulier, et rassemblées sous le nom *théorie du domaine*. Par exemple, dans un problème lié à la caractérisation de liens familiaux, le fait qu'un individu ne peut pas être à la fois *père* et *mère* pourra être utilisé.

— Ensuite, la manière dont le résultat sera utilisé est parfois une information pertinente. Dans le cas de molécules cancérigènes, il est préférable d'indiquer à tort qu'une molécule est cancérigène et de ne pas l'utiliser, plutôt que de cataloguer comme non cancérigène une molécule cancérigène et ainsi autoriser la diffusion d'un produit dangereux. Pour le concept `est_cancérigène`, les définitions les plus générales possibles de ces molécules seront favorisées.

— La connaissance de la source des exemples peut être utilisée en cours d'apprentissage : l'apprenti peut essayer de deviner ce que l'on veut lui faire apprendre, connaissant l'enseignant.

— Il est courant de privilégier la simplicité de la solution. La validité de ce biais est une simple application du rasoir d'Occam. Sa mise en œuvre est souvent plus délicate mais cela dépasse le cadre de cette discussion. Nous en discuterons plus avant au chapitre 6.

— Enfin, l'analogie avec des cas déjà traités peut amener des biais intéressants, par exemple en découvrant une caractérisation des généralisations les plus pertinentes pour un certain type de problème.

2.2 Apprentissage et algorithme générer-et-tester

2.2.1 L'apprentissage comme problème de recherche

Un problème de recherche est donné par un espace de recherche qui regroupe l'ensemble des états possibles, un état initial et la description d'un état final. Notez que les états finaux sont caractérisés par l'ensemble des propriétés qu'ils doivent présenter.

Pour résoudre un tel problème, il est nécessaire de se donner des *opérateurs* : un opérateur transforme un état en plusieurs nouveaux états. L'idée est de parcourir l'espace de recherche en appliquant successivement l'opérateur, en commençant par l'appliquer à l'état initial.

L'idée de considérer l'apprentissage comme un problème de recherche apparaît dans [Mitchell, 1982]. L'espace de recherche est constitué par l'ensemble des hypothèses de $\mathcal{L}_h$. L'état initial peut être l'hypothèse la plus générale de $\mathcal{L}_h$ ou une hypothèse très spécifique comme un exemple de A^+. Enfin l'état final est caractérisé par des propriétés liées à la correction et à la complétude vis-à-vis de A^+ et de A^- et d'éventuelles connaissances sur le domaine.

Dans le cas de l'apprentissage, les opérateurs appelés pour l'occasion *opérateurs de raffinement*, transforment une hypothèse de $\mathcal{L}_h$ en d'autres hypothèses de $\mathcal{L}_h$. Ces opérateurs peuvent agir de deux manières distinctes sur l'hypothèse courante H, distinction elle aussi introduite par Tom Michell [Mitchell, 1982].

— Soit l'opérateur travaille en fonction de H mais aussi des exemples d'apprentissage. Par exemple, si H ne couvre pas un exemple e^+ de A^+, l'opérateur pourra fournir des hypothèses plus générales que H qui couvrent e^+. Inversement, si H couvre un exemple e^- de A^-, l'opérateur produira des raffinements de H plus spécificiques qui écartent e^-. On dit dans ce cas que l'apprentissage est dirigé par les données (*data driven*).

— Soit il est totalement indépendant des données. Appliqué à H, il produira toujours les mêmes hypothèses, quels que soient A^+ et A^-. Une fois produites, il faudra évaluer la correction et la complétude de ces nouvelles hypothèses. Cette approche est appelée générer-et-tester (*generate-and-test*).

C'est ce dernier type de recherche qui va nous occuper dans la première partie de cette thèse.

2.2.2 Algorithme générer-et-tester

Bon nombre de systèmes sont basés sur cette approche. Parmi les plus connus, nous trouvons FOIL [Quinlan, 1990], MARVIN [Sammut et Banerji, 1986] ou encore CLINT [De Raedt, 1992]. Enfin, citons HAIKU [Nédellec et Rouveirol, 1994], un système générique qui se propose d'englober l'ensemble des algorithmes basés sur le générer-et-tester.

Plutôt que de passer en revue les systèmes existants, nous allons travailler à partir d'une version simplifiée de l'algorithme générique d'HAIKU qui, après instanciation, peut s'identifier à chacun de ces systèmes.

Cet algorithme est constitué de deux boucles imbriquées. La boucle interne (Algorithme 2.1) parcourt $\mathcal{L}_h$, à l'aide d'un opérateur, à la recherche d'une hypothèse satisfaisant certains critères, typiquement une hypothèse correcte par rapport à A^-. La boucle externe (Algorithme 2.2) fait appel à la boucle interne, récupère l'hypothèse découverte et cela jusqu'à ce que la disjonction formée des différentes hypothèses obtenues soit jugée satisfaisante (en général, la disjonction devra couvrir tous les exemples de A^+).

Algorithme 2.1 (Boucle interne du générer-et-tester)
L'ensemble des hypothèses courantes est initialisé avec des hypothèses fournies.

1. *Choisir l'une des hypothèses courantes et la tester : si elle est satisfaisante, elle est renvoyée comme définition partielle et l'on sort de cette boucle.*
2. *Appliquer l'opérateur de raffinement sur cette hypothèse pour obtenir un ensemble de nouvelles hypothèses.*
3. *Parmi ces nouvelles hypothèses*
   (a) *Enlever les hypothèses qui ne sont pas cohérentes par rapport aux hypothèses ayant déjà échouées et celles qui sont redondantes avec des hypothèses courantes.*

(b) *Enlever celles qui ne sont pas valides vis-à-vis des exemples.*

(c) *Dans les hypothèses courantes, remplacer l'hypothèse choisie au point (1) par les hypothèses issues de l'étape (3b). Puis, on retourne en (1).*

Algorithme 2.2 (Boucle externe du générer-et-tester)

1. *Initialiser la disjonction, définition du concept, à* $\emptyset$.
2. *Tant que la disjonction n'est pas satisfaisante :*
   (a) *Calculer les hypothèses de départ.*
   (b) *Explorer les hypothèses accessibles depuis les hypothèses de départ et récupérer une hypothèse interessante (algorithme 2.1).*
   (c) *Ajouter cette hypothèse, solution partielle, à la définition du concept.*
3. *Renvoyer la disjonction, définition du concept.*

Ces deux boucles devront être instanciées pour devenir un algorithme d'apprentissage. Les éléments génériques, comme le choix de l'hypothèse à raffiner, l'opérateur ou les critères d'arrêt, finalement instanciés constituent les biais de l'algorithme.

2.2.3 Biais et algorithme générer-et-tester

Nous l'avons vu, les biais sont des connaissances a priori, justifiées ou non, sur la solution. Nous l'avons vu aussi, il existe différents types de biais.

Dans le cadre du générer-et-tester, la classification établie par [Nédellec et al., 1996], sépare les biais en trois catégories. Nous allons y retrouver les biais dans le langage des hypothèses $\mathcal{L}_h$, catégorie de biais déjà identifiée par Tom Mitchell. En revanche, les biais qu'il avait caractérisés comme opérant dans la procédure de recherche de la solution, se partagent maintenant entre deux nouvelles classes.

1. Les *biais de langage* sont des informations sur la *forme* de la solution. Le choix de $\mathcal{L}_h$ est le premier des biais de langage mais il n'est pas le seul. Sans revenir sur la définition de $\mathcal{L}_h$, nous pouvons donner des informations sur les hypothèses de $\mathcal{L}_h$ qui vont nous intéresser. Par exemple, il est possible d' imposer qu'un attribut a soit utilisé dans la solution ; les hypothèses de $\mathcal{L}_h$ contenant $a =?$ ne seront donc pas des solutions potentielles bien qu'elles appartiennent toujours à l'espace de recherche.
2. Les *biais de recherche* indiquent comment parcourir l'espace de recherche, quelles hypothèses évaluer en priorité. Dans ce cadre, nous pouvons entre autre spécifier les opérateurs et le type de parcours : en largeur ou en profondeur d'abord, par exemple.
3. Les *biais de validation* permettent en particulier de réaliser que la solution a été découverte et que l'apprentissage peut prendre fin. Dans le cas d'un apprentissage disjonctif, ils permettront de décider si une hypothèse doit faire partie de la disjonction solution ou non. Ces biais sont typiquement les critères de correction et de complétude qui peuvent être affaiblis ou non, par exemple en présence de bruit.

Dans les sections suivantes, nous détaillons les biais majeurs utilisés en Programmation Logique Inductive : les biais de langage propres à l'ordre un, puis les opérateurs de raffinement travaillant sur des clauses.

2.3 Biais de langage en PLI

La Programmation Logique Inductive est un domaine de recherche qui s'est constitué autour d'un biais de langage posant que le langage $\mathcal{L}_h$ est formé de formules logiques du premier ordre et plus particulièrement de clauses définies. Les autres biais de langage utilisés en Programmation Logique Inductive se scindent classiquement en deux groupes :

1. Les *biais syntaxiques* sont des restrictions sur la forme des hypothèses. Ils définissent les clauses bien formées. La limitation de la taille des clauses, par exemple, est un biais syntaxique.
2. Les *biais sémantiques* nécessitent des informations qui ne sont pas contenues dans l'hypothèse elle-même. Ces informations proviennent soit de la théorie du domaine, soit des exemples. Ainsi, la théorie peut contenir la signature de ses prédicats ; nous pouvons alors refuser les clauses qui ne vérifient pas ces signatures.

Comme initié dans [Torre, 1995]), nous diviserons les biais syntaxiques en deux sous-classes selon leur manière d'opérer : les limitations numériques et les conditions sur des ensembles de variables.

2.3.1 Limitations numériques

Étant donnée une fonction numérique sur le langage $\mathcal{L}_h$, les biais que nous allons expliquer maintenant bornent la valeur de cette fonction. Une hypothèse ne pourra être considérée comme une éventuelle solution que si elle vérifie ces limitations.

Toutes les fonctions sur lesquelles reposent les biais suivants peuvent être vues comme des mesures de complexité et ces biais cherchent donc à limiter la complexité des hypothèses que le système devra considérer.

Pour certains des biais, cette limitation de l'espace de recherche est leur seule motivation. D'autres, en revanche, relèvent d'une connaissance propre au problème d'apprentissage considéré.

Nombre de littéraux dans le corps des clauses

Si ce nombre est limité à 1, il sera possible d'apprendre

$$member(X, L) \leftarrow head(L, X)$$

mais pas

$$member(X, L) \leftarrow tail(L, T), member(X, T)$$

Nombre de variables existentielles

Dans une clause de Horn, les variables de la tête sont implicitement quantifiées universellement et les autres existentiellement. Par exemple, la clause

$$\begin{aligned} intersection(X,Y,Z) \leftarrow\ & head(X,HX),\\ & member(HX,Y),\\ & tail(X,TX),\\ & intersection(TX,Y,W),\\ & cons(HX,W,Z) \end{aligned}$$

comporte trois variables existentielles : HX, TX et W.

Profondeur

Ce biais limite le nombre d'imbrications des foncteurs ; cette valeur est donnée par la fonction *Depth* suivante.

$$\begin{aligned} Depth(V) &= 0\\ Depth(c) &= 1\\ Depth(f(t_1,\ldots,t_n)) &= 1 + max(\{Depth(t_1),\ldots,Depth(t_n)\})\\ Depth(p(t_1,\ldots,t_n)) &= max(\{Depth(t_1),\ldots,Depth(t_n)\})\\ Depth(H \leftarrow L_1 \wedge \ldots \wedge L_k) &= max(\{Depth(H),\ldots,Depth(L_k)\}) \end{aligned}$$

Ainsi, la clause $odd(s(s(X))) \leftarrow odd(X)$ a la profondeur du terme $s(s(X))$, c'est-à-dire 2.

Par exemple, le système GOLEM [Muggleton et Feng, 1990] est capable d'utiliser ce biais à travers son paramètre i (GOLEM utilise un biais, nommé ij-détermination, qui combine plusieurs de nos biais élémentaires).

Ce biais possède une justification très forte : pour qu'une hypothèse H subsume un exemple e, il est nécessaire que la profondeur de H soit inférieure ou égale à celle de e. Par conséquent, la profondeur minimale observée dans A^+ fournit une borne supérieure à la profondeur de hypothèses.

Degré

L'idée intuitive est de limiter la longueur d'une chaîne d'instanciations aboutissant à l'instanciation d'une variable.

$$\begin{aligned} Level(c) &= 0\\ Level(V) &= 0 \text{ si V est une variable de la tête}\\ Level(V) &= 1 + min(\{Level(T) \mid T \text{ et } V \text{ dans un même littéral}\})\\ Level(C) &= max(\{Level(V) | V \in C\}) \end{aligned}$$

Par exemple, dans la clause suivante

$$\begin{aligned} \text{grand-père(GF)} \leftarrow\ & \text{homme(GF)},\\ & \text{parent(GF,C)},\\ & \text{parent(C,LC)} \end{aligned}$$

GF, C et LC ont respectivement les degrés 0, 1 et 2 (la clause a donc le degré 2). Il s'agit du paramètre j de GOLEM [Muggleton et Feng, 1990].

2.3.2 Conditions sur les variables

Range-restricted

Toutes les variables de la tête doivent apparaître dans le corps. La clause

$$\begin{aligned} arche(X,Y,Z) \quad &\leftarrow \quad cube(X), \\ &\qquad\quad cube(Y), \\ &\qquad\quad cube(Z) \end{aligned}$$

est satisfaisante,

$$\begin{aligned} arche(X,Y,Z) \quad &\leftarrow \quad cube(X), \\ &\qquad\quad cube(Y) \end{aligned}$$

ne l'est pas puisque la variable Z n'apparaît pas dans le corps.

Connexion

Celui-ci demande que les hypothèses soient liées. Les constantes et les variables de la tête sont liées ; de même pour une variable apparaissant dans un littéral avec un terme lié. Une clause est dite liée si toutes ses variables sont liées. Typiquement, ce biais permet d'éviter les clauses du type

$$\begin{aligned} \text{père(F)} \quad &\leftarrow \quad \text{parent(F,C)}, \\ &\qquad\quad \text{homme(X)} \end{aligned}$$

Cette clause est *range-restricted* mais la variable X n'est pas liée (et par suite la clause non plus). À noter le lien très fort entre ce dernier biais et le degré : le degré d'une clause est défini si et seulement si cette clause est connectée. L'utilisation simultanée de ces deux biais n'a donc pas d'intérêt puisque une borne sur le degré impose la connexion.

On trouve parfois une version plus forte de la connexion. Ainsi, le système FOIL [Quinlan, 1990] parcourt la clause de gauche à droite et exige que chaque littéral rencontré soit lié au début de la clause. La clause donnée ici est connectée pour la première définition mais pas pour la seconde :

$$\begin{aligned} length(L,1) \quad &\leftarrow \quad null(TL), \\ &\qquad\quad tail(L,TL) \end{aligned}$$

Cela tient au fait que la variable TL est liée après une première apparition où elle n'est pas liée.

2.3.3 Biais sémantiques

Comme à la section précédente, les conditions vont porter sur des ensembles de variables ; la différence est que ces biais vont utiliser de plus une information extérieure à la clause (c'est cela qui leur vaut le nom de *sémantique*).

Cette information prend la forme d'une étiquette pour chacun des arguments de chacun des prédicats pouvant apparaître dans les hypothèses. Comme précédemment, et à l'aide de ces étiquettes, il est possible de définir des conditions sur les

variables d'une hypothèse. Du fait de l'étiquetage, ces biais sont naturellement plus fins que les biais syntaxiques décrits ci-dessus.

Deux types d'étiquetages existent principalement :

1. le premier exploite le typage des prédicats ;
2. le second utilise l'aspect fonctionnel des prédicats.

MILES [Stahl et Tausend, 1994] utilise un biais sémantique qui fait intervenir le typage. Ce biais nous intéressera moins car il est, en général, implicitement exploité par les opérateurs ou directement dans le langage des hypothèses.

En revanche, nous allons détailler les biais liés à la fonctionnalité ou modes *input/output*. Ces modes sont utilisés dans les systèmes FILP [Bergadano et Gunetti, 1993] et TRACY [Bergadano et Gunetti, 1995] mais aussi dans PROGOL [Muggleton, 1995].

À chaque prédicat P d'arité n est associé un mode fonctionnel :
— m arguments de P sont déclarés comme *inputs* ;
— les $(n - m)$ autres arguments sont considérés comme *outputs* ;
Pour le prédicat *intersection* par exemple, la déclaration naturelle de mode est

$$intersection(in, in, out)$$

où l'on signifie que les deux premiers arguments sont des entrées et que le dernier contiendra le résultat. Par la suite, nous supposerons les modes suivants :

null(out)
head(in,out)
tail(in,out)
member(in,in)
notmember(in,in)

Avec ces informations, nous pouvons contraindre la forme des hypothèses de différentes manières.

Utilisation des entrées de la tête

Dans ce cas, nous demandons que toutes les variables *inputs* de la tête soient présentes dans le corps. Après confrontation avec ce biais, la clause

$$\begin{array}{lll} intersection(X, Y, Z) & \leftarrow & null(X), \\ & & null(Z) \end{array}$$

sera rejetée puisque l'*input* Y n'apparaît pas dans le corps.

Apparition des sorties de la tête dans le corps

À travers ce biais est traduite la volonté que les variables *outputs* prennent une valeur dans le corps, ce qui revient à exiger l'apparition de ces variables dans le corps en position d'output. La clause

$$\begin{array}{lll} intersection(X, Y, Z) & \leftarrow & tail(X, TX), \\ & & intersection(TX, Y, W) \end{array}$$

n'est pas satisfaisante puisque l'*output* Z ne prend jamais de valeur.

À noter que ce biais et le précédent sont des versions plus fines de la *range-restriction*.

Utilisation des résultats intermédiaires du corps

Ici, il faut que chaque résultat produit par un prédicat du corps soit utilisé : soit dans la suite du corps avec le statut *input*, soit comme *output* de la tête. Par exemple, dans la clause ci-dessous, la variable HL prend une valeur mais n'est pas utilisée.

$$\begin{array}{lll} member(X, L) & \leftarrow & head(L, HL), \\ & & tail(L, TL), \\ & & member(X, TL) \end{array}$$

Unicité de l'instanciation des sorties

Dans ce cas, c'est la surcharge des *outputs* qui est interdite : une variable ne doit prendre qu'une seule fois le statut *output*. Grâce à ce biais, nous éviterons par exemple la clause suivante

$$\begin{array}{lll} intersection(X, Y, Z) & \leftarrow & tail(X, TX), \\ & & head(X, HX), \\ & & intersection(TX, Y, Z), \\ & & intersection(TX, Y, W), \\ & & cons(HX, W, Z) \end{array}$$

puisque Z prend par deux fois une valeur.

Instanciation des entrées

Il serait souhaitable que chacune des variables utilisées comme *inputs* dans le corps ait été instanciée auparavant (soit parce que cette variable est un *input* de la tête, soit parce qu'elle est apparue en *output* dans un littéral précédent). La clause

$$\begin{array}{lll} intersection(X, Y, Z) & \leftarrow & head(X, HX), \\ & & intersection(TX, Y, W), \\ & & cons(HX, W, Z) \end{array}$$

sera rejetée car la variable TX est utilisée comme *input* alors qu'elle n'a jamais pris de valeur.

Après avoir détaillé les biais de langage les plus couramment utilisés en PLI, nous passons maintenant aux principaux biais de recherche que sont les opérateurs de raffinement.

2.4 Opérateurs de raffinement

2.4.1 Définitions, notations et discussion

En discutant précédemment de la résolution d'un problème de recherche par une approche générer-et-tester, nous avons donné l'intuition de ce qu'est un opérateur

de raffinement et du parcours de l'espace de recherche qu'il permet.

Il s'agit maintenant d'en donner une définition formelle et précise. Nous retrouvons la difficulté que nous avions rencontrée avec $\succeq$, qui peut être vu à la fois comme un test de couverture et comme une relation de généralité. De la même manière, un opérateur de raffinement présente deux aspects : l'un calculatoire, l'autre relationnel.

Nous donnons la définition dans ses deux versions, toutes deux largement inspirées de la définition originelle de Shapiro [Shapiro, 1981].

Définition 2.1 (opérateur (version calculatoire))
Un opérateur de raffinement *est un algorithme qui, à partir d'une hypothèse de $\mathcal{L}_h$, calcule un nombre fini d'hypothèses de $\mathcal{L}_h$.*

Définition 2.2 (opérateur (version relationnelle))
Un opérateur de raffinement *est une relation binaire sur l'espace de recherche $\mathcal{L}_h$.*

Ces deux définitions ne sont pas équivalentes : une relation peut ne pas être calculable. Cependant, même s'il est bien évident qu'en définitive nous aurons besoin d'opérateurs calculables, nous ne choisirons pas entre ces deux définitions : dans la suite, nous conserverons l'ambiguïté, utilisant à chaque fois la vision de l'opérateur qui se prête le mieux à la discussion.

Notation 2.1
- *Pour un opérateur $\mathcal{O}$, nous noterons $\mathcal{O}(H)$ les raffinements de l'hypothèse H.*
- *Du point de vue relationnel, $\mathcal{O}$ est l'ensemble des paires (H, H') telles que $H' \in \mathcal{O}(H)$.*
- *Si $\mathcal{O} \subseteq \mathcal{R}$, nous dirons que l'opérateur $\mathcal{O}$ respecte la relation $\mathcal{R}$.*

Les opérateurs de raffinements utilisent systématiquement la relation de généralité définie sur $\mathcal{L}_h$: soit ils parcourent cet espace du spécifique au général, soit l'inverse.

Définition 2.3 (descendant/ascendant)
- *Un opérateur de raffinement $\mathcal{O}$ est dit* descendant *si et seulement si $\mathcal{O}$ respecte $\succeq$, autrement dit :*

$$\forall C, D \in \mathcal{L}_h : D \in \mathcal{O}(C) \Rightarrow C \succeq D$$

- *Symétriquement, $\mathcal{O}$ est un opérateur* ascendant *si et seulement si $\mathcal{O}$ respecte la relation réciproque de $\succeq$:*

$$\forall C, D \in \mathcal{L}_h : D \in \mathcal{O}(C) \Rightarrow D \succeq C$$

Exemple 2.1
Par rapport à la θ-subsomption, l'ajout de littéraux est un opérateur de spécialisation, donc descendant. En revanche, l'abandon constitue une généralisation : c'est un opérateur ascendant.

Définissons un opérateur un peu plus complexe que les simples ajout ou abandon de littéraux : l'absorption [Muggleton et Buntine, 1988].

Définition 2.4 (absorption)
L'application de l'absorption à la clause initiale $T_i \leftarrow C_i$, étant donnée la clause de la théorie $T_t \leftarrow C_t$ telle qu'il existe une substitution θ et que $C_t\theta \subseteq C_i$, fournit la clause :

$$T_i \leftarrow Rel \wedge T_t\theta \text{ avec } C_i = C_t\theta \wedge Rel$$

Exemple 2.2
L'absorption sur l'hypothèse

$$\begin{array}{rcl} grand\_pere(tom, liz) & \leftarrow & pere(tom, helen), \\ & & femme(helen), parent(helen, liz) \end{array}$$

à l'aide de la clause de la théorie du domaine

$$mere(X, Y) \leftarrow femme(X), parent(X, Y)$$

fournit la clause (à travers la substitution $\theta = \{X/helen, Y/liz\}$) :

$$\begin{array}{rcl} grand\_pere(tom, liz) & \leftarrow & pere(tom, helen), \\ & & mere(helen, liz) \end{array}$$

Notons qu'une hypothèse et ses raffinements par absorption ne seront, en général, comparables que pour une relation de généralité faisant intervenir la théorie du domaine, comme la subsomption relative [Plotkin, 1971] ou la subsomption généralisée [Buntine, 1988].

Ainsi, grâce à ces opérateurs ascendants ou descendants, nous allons pouvoir parcourir l'espace de recherche de manière monotone par rapport à la relation de généralité. Reste à savoir de quelle(s) hypothèse(s) nous allons partir. Ce point constitue un élément générique du générer-et-tester que nous avons décrit à l'Algorithme 2.1. Les solutions sont donc multiples. Pour les approches descendantes, l'hypothèse la plus générale peut être utilisée comme départ de la recherche et elle est, de plus, facile à identifier : il s'agit de la clause dont le corps est vide et dont la tête est l'atome construit en utilisant le symbole de prédicat du concept à définir et des variables distinctes. Symétriquement, pour les ascendantes, il faut découvrir une clause maximalement spécifique mais cela est plus délicat. En général, c'est un exemple positif, éventuellement modifié, qui est utilisé. Certaines modifications fournissent une clause équivalente à l'exemple initial ; c'est le cas de PROGOL [Muggleton, 1995] ou ITOU [Rouveirol, 1992] qui utilise la saturation. D'autres fournissent une clause plus générale, par variabilisation de l'exemple.

Remarquons que nous avons cherché d'emblée les clauses maximalement générales et maximalement spécifiques de $\mathcal{L}_h$ susceptibles d'être solution. Cela traduit un souci de complétude : nous voulons pouvoir atteindre toutes les solutions potentielles à partir des hypothèses de départ, par application de l'opérateur.

La seule restriction que nous ayons posée sur les opérateurs de raffinement est le respect d'une relation de généralité, dans un sens ou dans l'autre. Cette condition est tellement faible que de nombreux opérateurs de raffinement sont possibles pour une relation donnée et d'autres conditions sur le comportement des opérateurs,de motivations diverses peuvent être posées.

Comme nous venons de l'évoquer informellement, nous voulons pouvoir parcourir tout $\mathcal{L}_h$ si besoin est. Une telle notion de complétude était déjà présente dans le travail de Shapiro [Shapiro, 1981] même s'il est apparu ensuite, que l'opérateur qu'il proposait n'était pas complet [Niblett, 1993].

Outre la complétude, nous sommes aussi soucieux de l'efficacité de notre parcours : par exemple, il n'est pas nécessaire, ni souhaitable, de rencontrer plusieurs fois les mêmes hypothèses.

Ces considérations ont conduit à la création de deux grandes familles d'opérateurs, les opérateurs *optimaux* et les opérateurs *idéaux*, chacune reposant sur des notions de complétude différentes.

2.4.2 Opérateurs optimaux

La définition de ces opérateurs a été introduite dans [De Raedt et Bruynooghe, 1993]. Tout d'abord, les auteurs donnent une définition des opérateurs de raffinement plus contraignante que la nôtre et qui nécessite de définir de nouvelles notions :

— L'hypothèse la plus générale de l'espace de recherche est notée $\top$.
— Nous notons $\mathcal{O}^*$ la clôture réflexive et transitive de l'opérateur $\mathcal{O}$, c'est-à-dire que $\mathcal{O}^*(H)$ regroupe l'ensemble des hypothèses obtenues par applications successives de $\mathcal{O}$ sur H et ses raffinements.

Définition 2.5 (opérateur DRB)
Un opérateur de raffinement DRB $\mathcal{O}$*, pour un espace de recherche* $\mathcal{L}_h$ *contenant* $\top$*, est une fonction de* $\mathcal{L}_h$ *dans* $2^{\mathcal{L}_h}$ *telle que, pour toute clause* C *de* $\mathcal{L}_h$*,* $\mathcal{O}(C)$ *ne contient que des spécialisations maximales de* C *et* $\mathcal{O}^*(\top) = \mathcal{L}_h$*.*

Plus explicitement, cette définition pose que
— l'espace de recherche $\mathcal{L}_h$ doit posséder un élément maximal $\top$;
— un opérateur est nécessairement descendant et procède par plus petits pas ;
— un opérateur est toujours complet, ce qui signifie dans ce cas précis, que toute hypothèse de $\mathcal{L}_h$ peut être atteinte par raffinements successifs de l'élément maximal $\top$.

L'optimalité pour ces opérateurs est donnée par la condition suivante.

Définition 2.6 (opérateur optimal)
Un opérateur $\mathcal{O}$ *est* optimal *si* $A \in \mathcal{O}^*(B)$ *et* $A \in \mathcal{O}^*(C)$ *implique soit* $C \in \mathcal{O}^*(B)$*, soit* $B \in \mathcal{O}^*(C)$*.*

Avec un tel opérateur, il y a une et une seule suite d'applications de l'opérateur qui mène de $\top$ à n'importe quelle clause donnée de l'espace de recherche. Cela implique en particulier que le graphe de raffinement d'un opérateur optimal est un arbre. Cette caractéristique rend ces opérateurs indéniablement efficaces.

2.4.3 Opérateurs idéaux

Les opérateurs idéaux sont beaucoup plus liés à la relation ordonnant l'espace de recherche. Ils sont basés sur l'idée que toutes les clauses comparables pour cette relation doivent pouvoir être dérivées l'une de l'autre, d'au moins une façon.

La définition de l'idéalité [van der Laag et Nienhuys-Cheng, 1994b] repose sur trois notions élémentaires. Un opérateur est *localement fini* si le raffinement d'une hypothèse produit un ensemble fini et calculable. Cela assure que l'opérateur a un intérêt pratique. Le raffinement d'une hypothèse est *strict* s'il ne fournit pas de clause équivalente à cette hypothèse. Ainsi, nous sommes assurés que l'opérateur fait réellement progresser la recherche. Enfin, dans le cadre de l'idéalité, l'opérateur est dit *complet* si les hypothèses comparables peuvent être liées par raffinements successifs. Sur ces bases, la définition de l'idéalité est la suivante [van der Laag et Nienhuys-Cheng, 1994b].

Définition 2.7 (opérateur idéal)
Un opérateur $\mathcal{O}$ est idéal *pour $\succeq$ s'il est à la fois localement fini, strict et complet par rapport à cette relation. Formellement,*

- *$\mathcal{O}$ est localement fini, signifie que pour toute hypothèse H de $\mathcal{L}_h$, $\mathcal{O}(H)$ est calculable.*
- *$\mathcal{O}$ est strict si pour toute H, $\mathcal{O}(H)$ ne contient pas de clauses équivalentes à H pour la relation $\succeq$:*

$$\forall H, H' \in \mathcal{L}_h : H' \in \mathcal{O}(H) \Rightarrow (H \succeq H') \wedge (H' \not\succeq H) \ ;$$

- *La complétude est vérifiée si pour tout couple d'hypothèses comparables par rapport à $\succeq$, elles peuvent être reliées par applications successives de O :*

$$\forall H, H' \in \mathcal{L}_h : H \succeq H' \Rightarrow H' \in \mathcal{O}^*(H) \ ;$$

Il a été prouvé que de tels opérateurs n'existaient pas pour des espaces non restreints ordonnés par la θ-subsomption ou l'implication logique [van der Laag et Nienhuys-Cheng, 1994b, van der Laag et Nienhuys-Cheng, 1994a, van der Laag, 1995]. L'intérêt pratique des opérateurs idéaux peut donc paraitre très limité. Pourtant, nous verrons que, pour certaines relations, ces opérateurs peuvent exister et nous en proposerons nous mêmes au chapitre 4.

2.5 Bilan

Nous avons démontré la nécessité des biais et décrit l'algorithme générer-et-tester, en général puis en explicitant les biais et les opérateurs dans le cas particulier de la programmation logique inductive. Cependant, nous n'avons pas évoqué leurs interactions.

Une fois une relation de généralité définie sur $\mathcal{L}_h$ et l'opérateur de raffinement choisi, que va produire l'utilisation d'un biais de langage ? Naturellement, le biais contraint la solution mais la recherche en est-elle plus efficace ? Ou, au contraire, le biais impose-t-il un coût supplémentaire ?

À l'inverse, le langage $\mathcal{L}_h$ et les autres biais de langage étant fixés, y a-t-il des opérateurs de raffinement plus pertinents que d'autres ? Peut-on construire un opérateur ad hoc ? Cette approche, dirigée par les biais de langage, est-elle compatible avec les opérateurs qui ont été adoptés par la communauté, à savoir les optimaux et les idéaux ?

Ces questions feront l'objet des prochains chapitres.

— Dans le chapitre 3, nous verrons comment les biais de langage peuvent s'intégrer à l'algorithme générer-et-tester. En particulier, à travers la notion de *propriété privée* que nous définirons, nous constaterons que tous les connaissances sur la solution ne peuvent être utilisées simultanément pour élaguer l'espace de recherche.
— Nous étudierons, au chapitre 4, comment modifier la relation de généralité pour permettre l'élagage par rapport aux biais disponibles. Nous montrerons que certaines de ces nouvelles relations, baptisées *relations naturelles*, autorisent l'existence d'opérateurs idéaux.
— Enfin, dans le dernier chapitre consacré à l'algorithme générer-et-tester (chapitre 5), nous introduirons notre propre cadre d'apprentissage qui intégrera au mieux les biais de langage et nos opérateurs *parfaits*, tout en satisfaisant les contraintes de complétude et d'efficacité.

Bibliographie

[Bergadano et Gunetti, 1993] Bergadano, F. et Gunetti, D. (1993). An interactive system to learn functional logic programs. In *Proceedings of IJCAI-93*, pages 1044–1049. Morgan Kaufmann.

[Bergadano et Gunetti, 1995] Bergadano, F. et Gunetti, D. (1995). Learning clauses by tracing derivations. In *ILP : from ML to Software Engineering*. MIT Press.

[Buntine, 1988] Buntine, W. (1988). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36(2) :375–399.

[De Raedt, 1992] De Raedt, L. (1992). *Interactive Theory Revision : An Inductive Logic Programming Approach*. Academic Press.

[De Raedt et Bruynooghe, 1993] De Raedt, L. et Bruynooghe, M. (1993). A theory of clausal discovery. In Bajcsy, R., éditeur, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann.

[Mitchell, 1977] Mitchell, T. M. (1977). Version spaces : a candidate elimination approach to rule learning. In Reddy, R., éditeur, *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 305–310. Morgan Kaufmann.

[Mitchell, 1980] Mitchell, T. M. (1980). The need for biases in learning generalizations. In *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann. Published in 1991.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18 :203–226.

[Muggleton, 1995] Muggleton, S. (1995). Inverse entailment and PROGOL. *New Generation Computing Journal*, 13 :245–286.

[Muggleton et Feng, 1990] Muggleton, S. et Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan.

[Muggleton et Buntine, 1988] Muggleton, S. H. et Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings 5th In-*

ternational Conference on Machine Learning, pages 339–352, San Mateo, CA. Morgan Kaufmann.

[Nédellec et Rouveirol, 1994] Nédellec, C. et Rouveirol, C. (1994). Specifications of the HAIKU system. Rapport interne 928, Laboratoire de Recherche en Informatique, Université Paris Sud.

[Nédellec et al., 1996] Nédellec, C., Rouveirol, C., Adé, H., Bergadano, F., et Tausend, B. (1996). Declarative bias in ILP. In De Raedt, L., éditeur, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press.

[Niblett, 1993] Niblett, T. (1993). A note on refinement operators. In Brazdil, P. B., éditeur, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 329–335. Springer-Verlag.

[Plotkin, 1971] Plotkin, G. (1971). A further note on inductive generalization. In *Machine Intelligence*, volume 6. Edinburgh University Press.

[Quinlan, 1990] Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3) :239–266.

[Rouveirol, 1992] Rouveirol, C. (1992). Extensions of inversion resolution applied to theory completion. In Muggleton, S., éditeur, *Inductive Logic Programming*, chapitre 3. Academic Press, London.

[Sammut et Banerji, 1986] Sammut, C. et Banerji, R. (1986). Learning concepts by asking questions. In Michalski, R., Carbonell, J., et Mitchell, T., éditeurs, *Machine Learning : An Artificial Intelligence Approach*, volume 2, chapitre 7, pages 167–192. Morgan Kaufmann.

[Shapiro, 1981] Shapiro, E. Y. (1981). Inductive inference of theories from facts. Rapport interne 192, Yale University Department of Computer Science.

[Stahl et Tausend, 1994] Stahl, I. et Tausend, B. (1994). MILES - a modular inductive logic programming experimentation system. Rapport interne 6020 (ESPRIT BRA ILP).

[Torre, 1995] Torre, F. (1995). Exploitation de biais de langage en apprentissage symbolique automatique dans un cadre logique. Rapport de DEA.

[van der Laag et Nienhuys-Cheng, 1994a] van der Laag, P. et Nienhuys-Cheng, S. H. (1994a). A note on ideal refinement operators in ILP. In Wrobel, S., éditeur, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 247–262. Gesellschaft für Mathematik und Datenverarbeitung MBH.

[van der Laag, 1995] van der Laag, P. R. J. (1995). *An Analysis of Refinement Operators in Inductive Logic Programming*. Thèse de doctorat, Erasmus Universiteit, Rotterdam, the Netherlands.

[van der Laag et Nienhuys-Cheng, 1994b] van der Laag, P. R. J. et Nienhuys-Cheng, S. (1994b). Existence and nonexistence of complete refinement operators. In Bergadano, F. et de Raedt, L., éditeurs, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag.

[Vapnik, 1995] Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

CHAPITRE 3

Propriétés privées

Dans ce chapitre, nous allons déterminer les conditions dans lesquelles un biais peut permettre un élagage de l'espace de recherche, c'est-à-dire les cas où une connaissance a priori va réellement aider l'algorithme générer-et-tester à parvenir plus rapidement à la solution.

Pour cela nous reprenons la notion de *propriété privée* que nous avons déjà définie dans [Torre, 1995]. Cette notion caractérise les propriétés par rapport auxquelles un élagage de l'espace de recherche peut être accompli sans pour autant prendre le risque de perdre une solution. Une telle propriété pourra être, en particulier, un biais de langage.

À titre d'illustration, nous réfléchissons ensuite à une représentation de $\mathcal{L}_h$ qui permette, d'une part, d'inclure toutes les connaissances sur la solution, et d'autre part, de produire efficacement l'ensemble des solutions.

Nous décrivons les approches qui suivent aussi ce paradigme (les ensembles de clauses de [Bergadano et Gunetti, 1995], les schémas de [Tausend, 1994], les modèles de clause [Adé et al., 1995], et les grammaires de [Cohen, 1994]). Enfin, nous détaillons le système Grandma [Torre, 1996] qui intègre grammaires et propriétés privées.

3.1 Propriétés de la solution

Nous nous focalisons ici sur les biais de langage et, plus généralement, nous nous intéressons à toutes les propriétés connues de la solution. À travers le terme de *propriété*, nous rassemblons donc les informations sur la forme de la solution mais aussi les contraintes de correction et complétude vis-à-vis de A^+ et A^-.

Nous allons supposer que toute propriété peut s'exprimer sous la forme $f(H)\mathcal{R}k$ où :

— f est une fonction de l'espace de recherche $\mathcal{L}_h$ dans un domaine quelconque $\mathcal{D}_f$;
— $\mathcal{R}$ est un pré-ordre sur $\mathcal{D}_f \times \mathcal{D}_f$;
— k est un élément de $\mathcal{D}_f$; typiquement, k représente le ou les paramètres de la propriété.

Dans le tableau 3.1, nous exprimons dans ce formalisme les propriétés possibles de la solution en reprenant en particulier les biais de langage de la section 2.3.

Notons dans ce tableau l'absence de la propriété ÉCARTE$_e$, qui consiste à rejeter un exemple. La raison en est que $\not\models$ n'est pas un pré-ordre. Cependant, quand cela sera nécessaire, nous considérerons cette propriété comme la négation de COUVRE$_e$ qui, elle, vérifie bien notre hypothèse de représentation.

3.2 Élagage et propriétés privées

Intuitivement, l'ensemble de ces informations doit restreindre l'espace de recherche parcouru par l'algorithme générer-et-tester. Nous voulons pouvoir nous limiter aux seules zones intéressantes de cet espace et nous voulons surtout nous arrêter dès que nous quittons l'une de ces régions : nommément, il s'agit d'un *élagage dynamique* de l'espace de recherche.

Un élagage, par rapport aux exemples, est bien connu en apprentissage puisqu'il a été introduit par Tom Mitchell en même temps que l'algorithme générer-et-tester [Mitchell, 1982].

Plaçons nous d'abord dans le cas d'une recherche d'une hypothèse correcte, à l'aide d'un opérateur ascendant $\mathcal{O}$, qui généralise donc les hypothèses. Si une hypothèse H couvre un exemple négatif de A^-, il est inutile de lui appliquer $\mathcal{O}$ puisque tous ses raffinements, plus généraux que H couvriront entre autres le même exemple négatif. L'hypothèse H, et tous ses raffinements par $\mathcal{O}$, peuvent donc être abandonnés sans autre forme de procès. De manière duale, si nous voulons une hypothèse qui soit complète, il ne sera pas nécessaire de spécialiser une hypothèse qui ne couvre pas un exemple positif de A^+ : ses raffinements ne le couvriront pas non plus.

C'est ce type d'élagage que nous voulons réaliser pour une propriété quelconque. Considérons une hypothèse H qui ne vérifie pas une propriété attendue. Nous ne pouvons arrêter de raffiner H que si nous avons la certitude qu'aucune descendante de H ne satisfaira cette propriété. Si ça n'est pas le cas, il faut poursuivre le raffinement de H et donc considérer des hypothèses qui ne satisfont pas les propriétés requises. Il est alors clair qu'aucun élagage ne sera effectué et nous risquons de parcourir tout l'espace de recherche.

Cette intuition est illustrée figure 3.1 et nous conduit à la définition suivante.

TABLE 3.1 – Propriétés possibles.

Propriété	Code	Définition
Couverture d'un exemple	COUVRE_e	$H \models e$
Range-restriction	$\text{RANGE-RESTRICTION}_b$	Range-restricted$(H) = b$
Connexion	CONNEXION_b	Connectée$(H) = b$
Réduction	RÉDUCTION_b	Réduite$(H) = b$
Taille maximale	TAILLEMAX_n	$\|H\| \leq n$
Taille minimale	TAILLEMIN_n	$\|H\| \geq n$
Nombre de variables maximal	NBVARSMAX_n	NbVars$(H) \leq n$
Nombre de variables minimal	NBVARSMIN_n	NbVars$(H) \geq n$
Profondeur maximale	PROFMAX_n	Prof$(H) \leq n$
Profondeur minimale	PROFMIN_n	Prof$(H) \geq n$
Degré maximal	DEGRÉMAX_n	Degré$(H) \leq n$
Degré minimal	DEGRÉMIN_n	Degré$(H) \geq n$

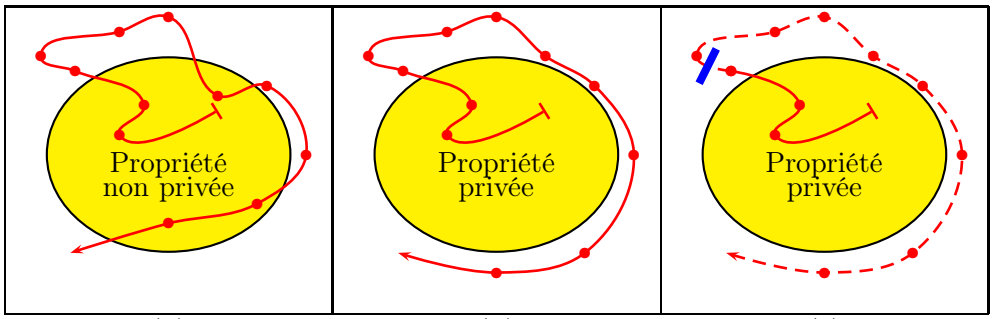


FIGURE 3.1 – Propriétés privée et non privée. La figure (a) démontre que l'élagage sûr n'est pas possible par rapport à une propriété non privée. Dans le cas (b), l'élagage est sûr et (c) exhibe sa réalisation pratique.

Définition 3.1 (propriété privée)
Une propriété P sera dite privée *par rapport à une relation $\mathcal{R}$ si et seulement si*
— *pour tout couple d'hypothèses (H, H') en relation par $\mathcal{R}$,*
— *pour toute valeur des paramètres de la propriété P,*
le fait que H ne vérifie pas la propriété P implique que H' ne la vérifie pas non plus. Plus formellement, P est privée par rapport à $\mathcal{R}$ si et seulement si

$$\forall H, H' \in \mathcal{L}_h : \forall \left[(H \mathcal{R} H') \wedge \overline{P(H)} \Rightarrow \overline{P(H')} \right]$$

Dans cette formule, on voit apparaître un quantificateur universel : celui-ci porte sur les variables libres de la formule. D'après la forme imposée aux propriétés à la section précédente, ces variables libres sont les paramètres de P.

Exemple 3.1 (longueur des hypothèses bornée)
Considérons la propriété TailleMax$_k$ *qui limite le nombre de littéraux d'une clause-solution à k maximum. La propriété* TailleMax$_k$ *est privée par rapport à une relation $\mathcal{R}$ si et seulement si :*

$$\forall H, H' \in \mathcal{L}_h, \forall k \in \mathbb{N} \ : (H \mathcal{R} H') \wedge |H| > k \Rightarrow |H'| > k$$

Une relation $\mathcal{R}$ qui vérifie cette formule est, par exemple, celle définie par : $H \mathcal{R} H'$ si et seulement s'il existe un littéral L tel que $H' = H \cup \{L\}$. Autrement dit, la propriété TailleMax$_k$ *est privée pour l'opérateur d'ajout de littéral.*

La table 3.2 donne les propriétés qui sont privées pour un opérateur d'ajout. Nous constatons, et c'est ce que nous aurions constaté pour n'importe quel autre opérateur classique en Programmation Logique Inductive, que les propriétés se séparent, moitié-moitié, en privées et non privées. Simplement, parce que, en général, si une propriété P est privée par rapport à un opérateur alors la propriété $\overline{P}$ ne l'est pas. Il apparaît donc que les propriétés connues de la solution ne pourront pas être toutes utilisées simultanément pour élaguer l'espace de recherche : certaines ne participeront pas à la recherche et ne seront testées qu'a posteriori, sur le résultat final de la recherche.

Nous allons nous détacher un instant du cadre du générer-et-tester pour illuster l'utilisation des propriétés privées et des non privées.

Nous allons définir un formalisme permettant de représenter $\mathcal{L}_h$ et à la fois de générer les clauses de cet ensemble. Puis, nous intégrerons les propriétés privées à ce formalisme : nous pourrons alors construire les hypothèses de $\mathcal{L}_h$ vérifiant les propriétés privées et cela avec une efficacité maximale. Il restera à tester les propriétés non privées sur ces hypothèses pour obtenir des solutions.

Nous commençons par exposer, à la section suivante, les différents formalismes existants permettant de décrire $\mathcal{L}_h$.

3.3 Langages de biais

Nous passons en revue les différents formalismes permettant de représenter l'espace de recherche. Pour chacun, nous donnerons, à titre d'exemple, le moyen de

TABLE 3.2 – Propriétés privées pour l'ajout.

Propriété	Privée pour l'ajout
Couverture d'un exemple	oui
Rejet d'un exemple	non
Range-restriction	non
Connexion	oui
Réduction	non
Taille maximale	oui
Taille minimale	non
Nombre de variables maximal	oui
Nombre de variables minimal	non
Profondeur maximale	oui
Profondeur minimale	non
Degré maximal	oui
Degré minimal	non

dénoter le langage suivant :

$$
\begin{aligned}
\mathcal{L}_h = \{ \\
&grand\_pere(X,Y) \leftarrow homme(Y), parent(X,Z) \\
&grand\_pere(X,Y) \leftarrow femme(Y), parent(X,Z) \\
&grand\_pere(X,Y) \leftarrow homme(X), parent(X,Z) \\
&grand\_pere(X,Y) \leftarrow femme(X), parent(X,Z) \\
&grand\_pere(X,Y) \leftarrow homme(Y), parent(X,Z), parent(Z,Y) \\
&grand\_pere(X,Y) \leftarrow femme(Y), parent(X,Z), parent(Z,Y) \\
&grand\_pere(X,Y) \leftarrow homme(X), parent(X,Z), parent(Z,Y) \\
&grand\_pere(X,Y) \leftarrow femme(X), parent(X,Z), parent(Z,Y) \\
&grand\_pere(X,Y) \leftarrow homme(Y), parent(X,Z), parent(X,Y) \\
&grand\_pere(X,Y) \leftarrow femme(Y), parent(X,Z), parent(X,Y) \\
&grand\_pere(X,Y) \leftarrow homme(X), parent(X,Z), parent(X,Y) \\
&grand\_pere(X,Y) \leftarrow femme(X), parent(X,Z), parent(X,Y) \\
&grand\_pere(X,Y) \leftarrow homme(Y), parent(X,Z), \\
&\qquad\qquad parent(Z,Y), parent(X,Y) \\
&grand\_pere(X,Y) \leftarrow femme(Y), parent(X,Z), \\
&\qquad\qquad parent(Z,Y), parent(X,Y) \\
&grand\_pere(X,Y) \leftarrow homme(X), parent(X,Z), \\
&\qquad\qquad parent(Z,Y), parent(X,Y) \\
&grand\_pere(X,Y) \leftarrow femme(X), parent(X,Z), \\
&\qquad\qquad parent(Z,Y), parent(X,Y) \\
\}
\end{aligned}
$$

3.3.1 Les ensembles de clauses

Cette représentation a été introduite dans [Bergadano et Gunetti, 1995] pour le système TRACY.

Pour une clause, il faut fournir les prédicats pouvant y apparaître, et pour chacun de ces prédicats les arguments possibles. Finalement l'espace est représenté par un *ensemble de clauses*.

Commençons par définir les ensembles plus simples :

1. Les *ensembles de termes* fournissent les combinaisons d'arguments autorisés pour un prédicat. $est\_stable(\{A, B\}, \{C, D\})$ autorise les littéraux suivants :

$$\begin{array}{l} est\_stable(A, C) \\ est\_stable(A, D) \\ est\_stable(B, C) \\ est\_stable(B, D) \end{array}$$

2. Les *ensembles de prédicats* donnent les combinaisons de prédicats pouvant apparaître dans une clause. $\{cube(X), cube(Y), cube(Z)\}$ autorise les séquences suivantes (en plus de la séquence vide) :

$$\begin{array}{l} cube(X) \\ cube(Y) \\ cube(Z) \\ cube(X), cube(Y) \\ cube(X), cube(Z) \\ cube(Y), cube(Z) \\ cube(X), cube(Y), cube(Z) \end{array}$$

   Ces prédicats peuvent naturellement faire apparaître des *ensembles de termes* dans leurs arguments.

3. Finalement, les deux ensembles précédents permettent de construire des clauses qui, à leur tour, forment un *ensemble de clauses*.

L'espace défini à la section précédente est dénoté par l'*ensemble de clauses* suivant :

$$\begin{array}{lll} \mathcal{L}_h = \{ & & \\ & grand\_pere(X, Y) \leftarrow & homme(\{X, Y\}), parent(X, Z), \\ & & \{parent(\{X, Z\}, Y)\} \\ & grand\_pere(X, Y) \leftarrow & femme(\{X, Y\}), parent(X, Z), \\ & & \{parent(\{X, Z\}, Y)\} \\ \} & & \end{array}$$

3.3.2 Les schémas

Les schémas [Tausend, 1994] utilisent eux des prédicats variables : un symbole générique apparaît à la place du nom de prédicat et l'on indique simplement quelles variables devra utiliser le littéral. Ainsi $P(X)$ dénote l'ensemble des littéraux $P(X)\theta$

où θ est une substitution du second ordre (qui remplace donc P par un symbole de prédicat de la théorie du domaine à un seul argument).

L'exemple qui nous sert de fil conducteur est représenté par les schémas suivants :

$$
\begin{array}{ll}
\mathcal{L}_h = \{ & \\
\quad & grand\_pere(X,Y) \leftarrow \ P(Y), Q(X,Z) \\
\quad & grand\_pere(X,Y) \leftarrow \ P(Y), Q(X,Z), parent(X,Y) \\
\quad & grand\_pere(X,Y) \leftarrow \ P(Y), Q(X,Z), parent(Z,Y) \\
\quad & grand\_pere(X,Y) \leftarrow \ P(Y), Q(X,Z), parent(X,Y), parent(Z,Y) \\
\quad & grand\_pere(X,Y) \leftarrow \ P(X), Q(X,Z) \\
\quad & grand\_pere(X,Y) \leftarrow \ P(X), Q(X,Z), parent(X,Y) \\
\quad & grand\_pere(X,Y) \leftarrow \ P(X), Q(X,Z), parent(Z,Y) \\
\quad & grand\_pere(X,Y) \leftarrow \ P(X), Q(X,Z), parent(X,Y), parent(Z,Y) \\
\} &
\end{array}
$$

Des schémas plus sophistiqués ont été définis : ils permettent de spécifier des conditions sur les prédicats candidats à la substitution (conditions qui peuvent par exemple porter sur l'arité) et sur les variables qui apparaîtront dans le littéral final.

3.3.3 Les modèles de clause

Les deux formalismes précédents sont complémentaires dans le sens où ce qui est facile à représenter dans l'un est difficile à représenter dans l'autre :

- les ensembles de clauses peuvent représenter de manière concise des clauses de longueurs différentes ; en revanche, il faudra autant de schémas qu'il y a de longueurs possibles.
- si nous travaillons avec des clauses de même longueur mais utilisant des prédicats différents, ce sont les schémas qui les représentent de la manière la plus naturelle.

L'idée développée dans [Adé et al., 1995], et reprise dans $\mathcal{D}$LAB [Dehaspe et De Raedt, 1996], est alors d'unifier ces deux représentations, pour en cumuler les avantages, dans des *modèles de clause*.

Ce formalisme utilise les conventions et notations suivantes :

- un atome est de la forme $p(t_1, \ldots, t_n)(n \geq 0)$ où p est un prédicat, les t_i sont des termes ou des ensembles de termes (comme dans les ensembles de clauses) ;
- un atome variabilisé est de la forme $P(t_1, \ldots, t_n)$ où P est un prédicat générique (au sens des schémas).

Définition 3.2 (modèle de clause)
Un modèle de clause est de la forme

$$Head \leftarrow Body, BodySet$$

avec

- *Head, la tête, un atome (variabilisé ou non) ;*
- *Body, une séquence $A_1, \ldots, A_n (n \geq 0)$ où les A_i sont des atomes (variabilisés ou non) ;*

— *BodySet, un ensemble d'atomes.*

Voyons maintenant les clauses décrites par un tel modèle de clause.

1. Si les littéraux du modèle ne contiennent pas d'ensembles de termes, alors ce modèle représente les clauses suivantes.

$$\{Head\theta \leftarrow Body\theta \cup B \mid \begin{array}{ll} \theta & \text{substitution du } 2^{\text{nd}} \text{ ordre} \\ B & \subseteq BodySet\}\end{array}$$

2. Si $BodySet = \{b_1, \ldots, b_n\}$ contient $b_i = p(T_1, \ldots, T_k)$ avec T_j un ensemble de termes $\{t_1, \ldots, t_l\}$, alors on dénote le même langage que la clause $Head \leftarrow Body, BodySet'$ où

$$\begin{array}{rcl} BodySet' & = & (BodySet - \{p(T_1, \ldots, T_k)\}) \\ & \cup & \{p(T_1, \ldots, T_{j-1}, t, T_{j+1}, \ldots, T_k) \mid t \in \{t_1, \ldots, t_l\}\}\end{array}$$

3. Si $Body = b_1, \ldots, b_n$ contient un $b_i = p(T_1, \ldots, T_k)$ où p peut être variable et $T_j = \{t_1, \ldots, t_l\}$, alors les clauses ainsi représentées sont les suivantes

$$\left\{ \begin{array}{rl} Head \leftarrow & b_1, \ldots, b_{i-1}, p(T_1, \ldots, T_{j-1}, t, T_{j+1}, \ldots, T_k), \\ & b_{i+1}, \ldots, b_n, BodySet \mid t \in \{t_1, \ldots, t_l\} \end{array} \right.$$

Conformément à l'idée de départ, les schémas comme les ensembles de clauses sont des cas particuliers de ces modèles :

— les schémas sont des modèles qui n'utilisent ni les ensembles de prédicats, ni les ensembles de termes.

— les ensembles de clause sont des modèles sans prédicat variable.

Ce pouvoir de synthèse est visible sur notre exemple puisqu'il est représenté par une expression extrêmement concise :

$$\mathcal{L}_h = \left\{ \begin{array}{rl} grand\_pere(X,Y) \leftarrow & P(\{X,Y\}), Q(X,Z), \{parent(\{X,Z\},Y)\} \end{array} \right.$$

3.3.4 Les grammaires

L'idée développée dans [Cohen, 1994] est d'utiliser une grammaire hors contexte. Dans notre cadre, les symboles utilisés sont des littéraux : nous allons donc considérer des l-symboles, des l-terminaux et des l-non-terminaux. $body(T)$ est le l-symbole de départ de la grammaire, où T est le concept que l'on cherche à définir. De plus, la convention suivante peut être mise en œuvre :

Notation 3.1
$A \rightarrow B$ where P (où P est un but PROLOG) dénote l'ensemble de règles

$$\left\{ \begin{array}{rcl} A\theta_1 & \rightarrow & B\theta_1 \\ A\theta_2 & \rightarrow & B\theta_2 \\ & \vdots & \\ A\theta_n & \rightarrow & B\theta_n \end{array} \right.$$

où θ_i est une substitution associée à une preuve de P.

Plus qu'une représentation, les grammaires offrent une procédure pour générer les clauses dénotées.

Définition 3.3 (dérivation)
La notion de dérivation est évidemment très semblable à la dérivation classique à ceci près qu'elle fait intervenir l'unification.
— *Si $\alpha A'\beta$ est une chaîne de l-symboles,*
— *s'il existe une règle $A \to \gamma$ dans la grammaire G,*
— *si A et A' ont un plus grand unificateur θ,*
alors on dit que $\alpha A'\beta$ dérive $(\alpha\gamma\beta)\theta$ en une étape dans G :

$$\alpha A'\beta \overset{G}{\Rightarrow} (\alpha\gamma\beta)\theta$$

Notation 3.2 (fermeture réflexive et transitive)
$\alpha_1 \overset{G}{\Rightarrow} \alpha_2 \overset{G}{\Rightarrow} \cdots \overset{G}{\Rightarrow} \alpha_{n-1} \overset{G}{\Rightarrow} \alpha_n$ *se réécrit* $\alpha_1 \overset{G}{\Rightarrow}{}^* \alpha_n$.

Avec cette dérivation, les clauses représentées se définissent comme le langage dans les grammaires classiques.

Définition 3.4 (langage décrit)
Le langage décrit est l'ensemble des chaînes de l-terminaux α tels que $S \Rightarrow^* \alpha$ (où S est le l-symbole de départ). On dit qu'un tel α est un mot du langage.

Ce formalisme est sans aucun doute le plus expressif et le plus souple de ceux présentés ici. La grammaire suivante permet de générer l'espace que nous avons considéré tout au long de ces descriptions.

$$\begin{aligned}
body(grand_pere(X,Y)) &\to sexe(A), parent(X,Z), autre(X,Y,Z)\\
&\quad\ where\ member(A,[X,Y]).\\
sexe(A) &\to homme(A).\\
sexe(A) &\to femme(A).\\
autre(X,Y,Z) &\to [].\\
autre(X,Y,Z) &\to parent(X,Z).\\
autre(X,Y,Z) &\to parent(X,Y), autre(X,Y,Z).
\end{aligned}$$

Comme les modèles, les grammaires de Cohen réunissent les clauses de longueur variable (avec les règles récursives) et les prédicats variables (avec les règles disjonctives).

Nous allons maintenant proposer notre propre formalisme, lui aussi basé sur les grammaires. Nous allons en plus intégrer dans la dérivation le moyen de vérifier les propriétés privées et de stopper cette dérivation en cas d'échec.

3.4 Le système Grandma

3.4.1 Motivations et principes

Notre tâche se divise en deux parties. Tout d'abord, nous devons être capables de représenter un ensemble de clauses. Nous avons choisi d'utiliser pour cela des

grammaires hors-contexte où les symboles sont des littéraux, à la manière de [Cohen, 1994].

D'autre part, nous devons isoler les mots du langage d'une grammaire donnée, c'est-à-dire les clauses de ce langage, vérifiant les propriétés choisies. La technique la plus simple consiste à développer exhaustivement le langage de cette grammaire, puis à tester chacune des clauses obtenues contre les propriétés choisies. Cette méthode n'est évidemment pas viable, en particulier dans le cas où la grammaire dénote un langage infini.

Pour notre part, nous allons essayer d'optimiser le test des propriétés privées. Mais que signifie ici propriété privée ? Observée avec la granularité de la dérivation, une grammaire se comporte comme un opérateur qui ajoute un littéral à la fois, l'ajout se produisant avec l'apparition d'un terminal. Les propriétés qui sont privées pour un tel ajout vont donc pouvoir être testées après chaque terminal de la grammaire. Les propriétés non privées, elles, ne pourront pas être considérées avant la construction d'un mot de $\mathcal{L}_h$.

Nous avons vu à la table 3.2, la liste des propriétés privées pour l'ajout. Cependant, la plupart des propriétés portent sur l'ensemble de la clause et nous ne possédons, au moment de l'apparition d'un terminal, que les informations liées à ce terminal.

L'idée est alors de mettre en place un moyen de communication à l'intérieur même des grammaires dans le but d'amener à chaque terminal des informations sur les littéraux le précédant. Grâce à ces informations, nous n'aurons pas à effectuer de calcul sur l'ensemble de la clause développée et nous éviterons d'énumérer toutes les clauses du langage de la grammaire.

Dans ce but, nous nous proposons d'exploiter un formalisme qui est couramment utilisé pour spécifier la syntaxe et la sémantique des langages de programmation : les grammaires attribuées [Alblas, 1991]. Il s'agit d'une extension des grammaires hors-contexte où l'on associe aux symboles de la grammaire des attributs, un attribut pouvant représenter une quantité quelconque. Les valeurs de ces attributs sont définies par des règles sémantiques associées à chaque règle de la grammaire. Pour un symbole S, deux types d'attributs sont distingués :

— soit l'attribut possède une valeur avant la dérivation de S, cette valeur a été calculée par les ancêtres de S, on dit que l'attribut est *hérité*.
— soit la valeur de l'attribut ne sera définie qu'après la dérivation de S, en fonction des attributs hérités de S et de ses descendants ; dans ce cas, il s'agit d'un attribut *synthétisé*.

Pour vérifier les propriétés privées, nous associons un attribut hérité C_h et un attribut synthétisé C_s à chaque symbole (terminal ou non terminal) de la grammaire hors-contexte. Ces attributs vont nous permettre de transporter des contextes, c'est-à-dire une description de la partie de la clause déjà construite, description qui s'exprime en termes de satisfaction des propriétés attendues pour les définitions de concept.

Par exemple, considérons la propriété TailleMax$_n$, vraie pour les clauses de taille inférieure ou égale à n littéraux. Pour la vérifier, nous allons inscrire dans le contexte, le couple composé de n et du nombre de littéraux déjà présents dans le corps de la clause.

Intuitivement, un symbole S_i va opérer dans le contexte $S_i.C_h$ (contexte qui est hérité de ses prédécesseurs) ; si tout se passe bien, S_i informe ses successeurs de son travail à travers le contexte synthétisé $S_i.C_s$. Plus formellement, à chaque règle $S_0 \to S_1 \dots S_n$ de la grammaire hors-contexte, nous associons la règle sémantique suivante :

$$\begin{array}{ll} S_0.C_s = S_n.C_s & \\ S_1.C_h = S_0.C_h & \\ S_i.C_h = S_{i-1}.C_s & \text{pour } i > 1, \\ S_i.C_s = \mathit{valide}(S_i, S_i.C_h) & \text{pour } S_i \text{ terminal.} \end{array}$$

Dans le cas d'une dérivation vide $S_0 \to$, nous avons simplement :

$$S_0.C_s = S_0.C_h$$

La fonction *valide* teste chaque non terminal vis-à-vis des propriétés présentes dans le contexte. Plus précisément, cette fonction va, pour chaque information I de $S_i.C_h$, considérer la propriété correspondante, c'est-à-dire tester et mettre à jour I en fonction du nouveau littéral S_i. Dans le cas de la limitation de la taille des clauses à n littéraux, *valide* prend en entrée un contexte et y prélève le couple *(n,taille actuelle de l'hypothèse)* ; si cette dernière vaut déjà n, la dérivation échoue ; sinon, on lui ajoute 1, pour prendre en compte l'ajout de S_i), et cette nouvelle valeur constitue alors le contexte synthétisé.

Exemple 3.2 (une grammaire pour grand-père)
Considérons la grammaire suivante dont les non terminaux sont grand_pere, liens, geniteur, paternel et maternel, et dont les terminaux sont parent, pere et mere. ***[]*** *représente le mot vide.*

grand_pere(X,Y) → *liens(X,Y), liens(X,Z), liens(Y,Z).*
liens(A,B) → ***[]***.
liens(A,B) → *geniteur(A,B).*
geniteur(A,B) → ***[parent(A,B)]***.
geniteur(A,B) → ***[parent(B,A)]***.
geniteur(A,B) → *paternel(A,B).*
geniteur(A,B) → *maternel(A,B).*
paternel(A,B) → ***[pere(A,B)]***.
paternel(A,B) → ***[pere(B,A)]***.
maternel(A,B) → ***[mere(A,B)]***.
maternel(A,B) → ***[mere(B,A)]***.

La Figure 3.2 illustre la dérivation d'une clause satisfaisante, tandis que la Figure 3.3 montre le contrôle opéré sur la propriété TAILLEMAX$_2$ *durant cette même dérivation. Sur cette figure, nous avons repris la convention qui consiste à noter les attributs hérités à gauche, les synthétisés à droite.*

De manière générale, lorsque *valide* échoue, c'est que l'ajout du littéral est incompatible avec le début de la définition par rapport à l'une des propriétés privées considérées et, par suite, cette combinaison est abandonnée. Grâce à ces grammaires

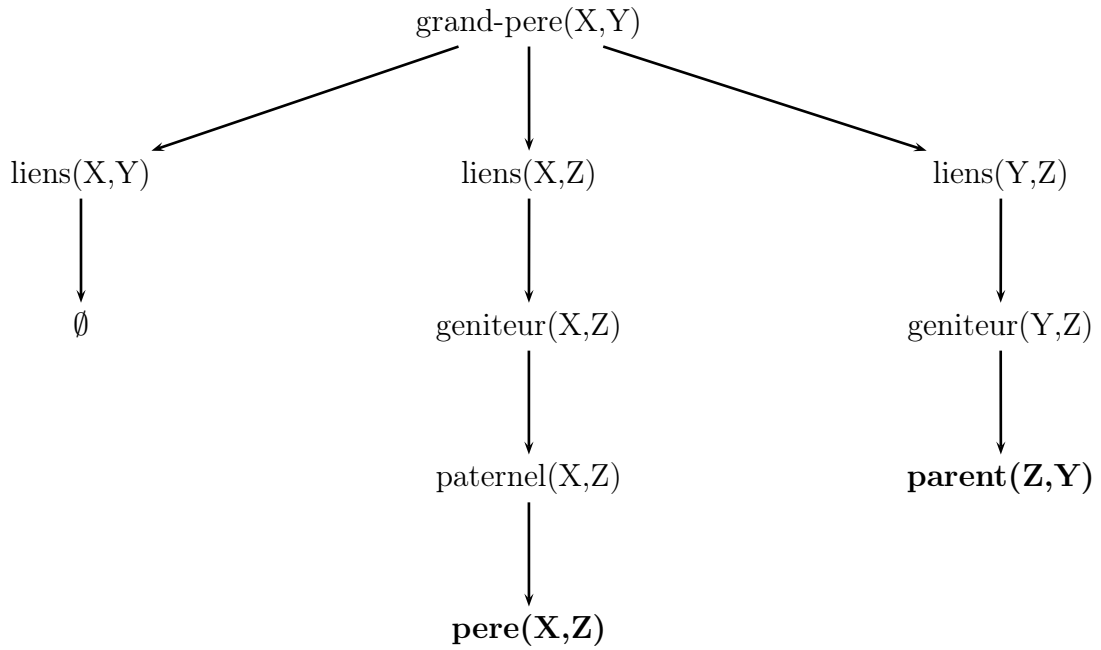


FIGURE 3.2 – Dérivation de grand-pere(X,Y) ← pere(X,Z),parent(Z,Y)

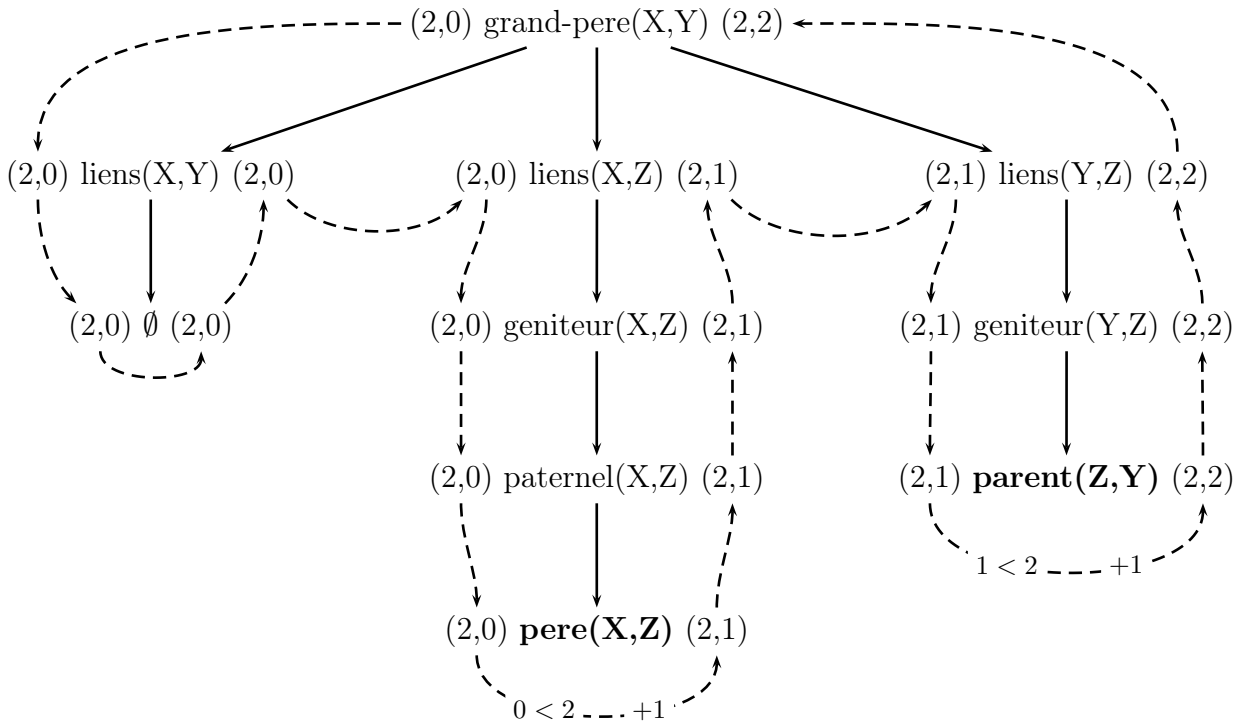


FIGURE 3.3 – Dérivation avec contrôle sur la taille

attribuées, nous allons pouvoir stopper au plus tôt la construction d'une clause non satisfaisante.

Cette méthode de vérification des propriétés nous permet aussi de réduire le coût de vérification d'une propriété. D'une part, pour une clause donnée, le test ne porte pas sur l'ensemble de ses littéraux mais sur le dernier ajouté et sur l'information représentant le début de la clause. D'autre part, le résultat de ce test élémentaire circule à l'aide des attributs, et sera donc connu par toutes les hypothèses construites par ajout sur la clause testée.

Autre aspect positif : au cours de la génération du langage représenté par l'une de nos grammaires contraintes, ne sont considérées que les clauses du langage

1. vérifiant nos propriétés ;
2. ne les vérifiant pas mais construites par adjonction d'un unique littéral à une clause vérifiant ces propriétés.

En particulier, cela signifie que si l'on utilise une propriété qui n'est vraie que pour un ensemble fini de clauses, nous sommes capables d'isoler les clauses satisfaisantes d'un espace infini en n'observant qu'un nombre fini de clauses.

3.4.2 Réalisation pratique

D'un point de vue pratique, de telles grammaires (avec attributs et une instruction *valide*) peuvent être réalisées sous forme de DCG ou *Definite Clause Grammar* (toute implantation de PROLOG possède un module capable de développer le langage dénoté par une DCG). Si nous supposons donnée une grammaire hors-contexte, le passage à une DCG simulant une grammaire dotée d'une instruction *valide*, se fait en ajoutant deux variables à chaque non terminal, la première représentant le contexte hérité, la seconde le contexte synthétisé. Les terminaux, eux, ne sont pas modifiés mais sont suivis d'une instruction *valide*. Cette instruction récupère le dernier contexte et construit, si l'ajout est effectivement validé, le nouveau contexte (qui est donc propagé dans la suite de la règle).

La grammaire suivante, indique que l'on peut introduire n'importe quel littéral construit à partir d'un symbole de prédicat et des variables quelconques.

$$\begin{array}{ll} \text{concept}(\vec{V}) & \rightarrow \text{littéraux}(\vec{V}) \\ \text{littéraux}(L) & \rightarrow [] \\ \text{littéraux}(L) & \rightarrow \{\text{variables}(\vec{V}, L, NL)\}, [P(\vec{V})], \text{littéraux}(NL) \end{array}$$

Explicitons le travail de cette grammaire. *concept* est le prédicat du concept à découvrir. Dans la première règle, $\vec{V}$ regroupe des variables distinctes, leur nombre correspond à l'arité du concept et nous indiquons finalement que la définition va utiliser des littéraux utilisant cet ensemble de variables. Dans la dernière règle, nous indiquons comment construire un tel littéral : à partir d'un symbole de prédicat P quelconque et de variables déjà utilisées et apparaissant dans L ou de nouvelles variables. Enfin, cette liste de variables utilisées est mise à jour.

Cette grammaire triviale peut être obtenue dans tous les cas. Mais, il est tout de même préférable qu'un expert fournisse une grammaire plus pertinente.

Pour une grammaire et donc un langage donnés, Grandma permet de mesurer l'élagage produit par les différents biais sur ce langage : il suffit de générer les différents langages induits par la grammaire et les biais, puis de comparer leurs tailles. Il est en cela une plate-forme de tests dans l'esprit de HAIKU [Nédellec et Rouveirol, 1994] et trouve des échos chez [Utgoff, 1986]. Cet aspect est illustré aux Figures 3.4, 3.5, 3.6, et 3.7, pour les deux grammaires que nous avons vues pour le problème *grand-père* et différentes associations de biais.

Il est ensuite possible de caractériser, à l'aide de nos grammaires, les opérateurs habituellement utilisés. La grammaire elle-même va représenter l'espace potentiellement engendrable par l'opérateur considéré. Ensuite, il faut traduire le comportement de l'opérateur dans cet espace et cela à travers la stratégie de dérivation.

Pour simuler l'ajout de FOIL [Quinlan, 1990], à partir de la grammaire triviale décrite précédemment, il reste à indiquer que l'ajout ne peut se produire qu'en fin de clause : cela est réalisé si l'on dérive notre langage en utilisant une stratégie gauche-droite, profondeur d'abord (la stratégie de PROLOG). Nous pouvons aller jusqu'à coder le critère d'entropie de FOIL pour choisir à chaque fois un unique littéral, mais notre approche y perdrait alors de son intérêt : nous ne serions plus capables que de générer une unique clause.

Pour résumer, dans Grandma, les biais de langage sont codés dans la grammaire, à travers les règles syntaxiques et sémantiques, tandis que les biais de recherche, eux, sont codés dans la stratégie de dérivation.

3.5 Bilan

Dans ce chapitre, nous avons caractérisé, à travers la notion de *propriété privée*, les propriétés qui autorisaient un élagage dynamique de l'espace de recherche, cela par rapport à l'opérateur de raffinement choisi.

Pour illuster l'intérêt de cette notion, nous avons décrit le système Grandma. Celui-ci est simplement un développeur de grammaires contraintes, telles que nous les avons décrites. Ainsi, Grandma peut générer des clauses pour des systèmes travaillant sur une représentation exhaustive de leur espace de recherche, comme TRACY [Bergadano et Gunetti, 1995], par exemple. Mais, si les propriétés font intervenir les propriétés de couverture et de correction, COUVRE$_e$ et ÉCARTE$_e$, Grandma se comporte comme un véritable système d'apprentissage puisque le langage alors dénoté par la grammaire attribuée, correspond aux définitions possibles du concept.

Il est aussi intéressant de constater que cette approche apporte un élément de réponse au No Free Lunch Theorem que nous avons présenté à la section 1.7 (page 18). Celui-ci suppose qu'un système apprend la même définition pour un problème et son dual, sous prétexte que les ensembles d'exemples sont les mêmes. fixe, immuable quel que soit le problème, mais s'adapter à chaque fois à travers les biais disponibles.

Cette démarche pour combattre le résultat du No Free Lunch Theorem rappelle celle de Carla Brodley : celle-ci se donne plusieurs méthodes d'apprentissage, puis, utilise des heuristiques pour choisir, selon le problème, la méthode à appliquer pour apprendre [Brodley, 1993]. Dans notre cas, il n'y a qu'un seul système mais son fonctionnement interne est modifié par ses biais, différents à chaque fois.

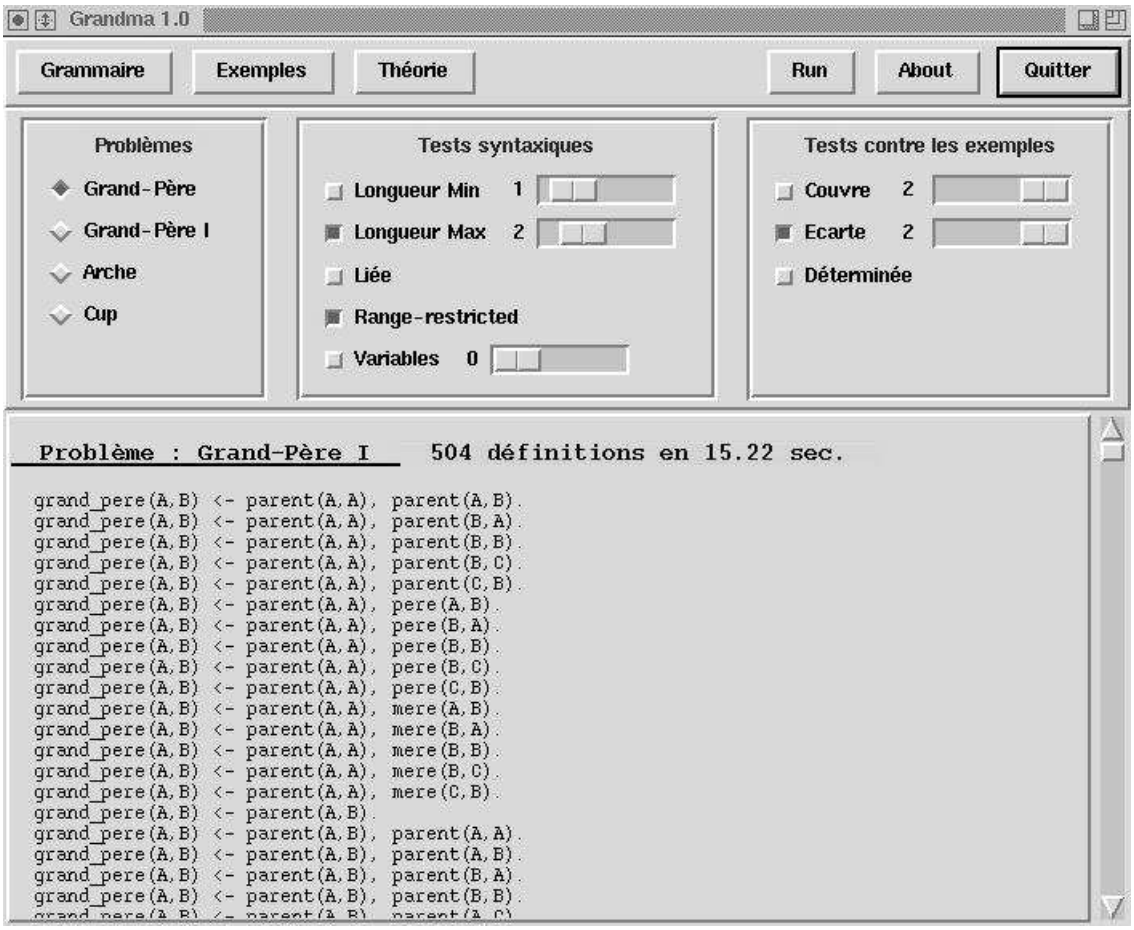


FIGURE 3.4 – Grandma et grand-père avec la grammaire triviale : le langage dénoté est infini, l'utilisation de biais est obligatoire.

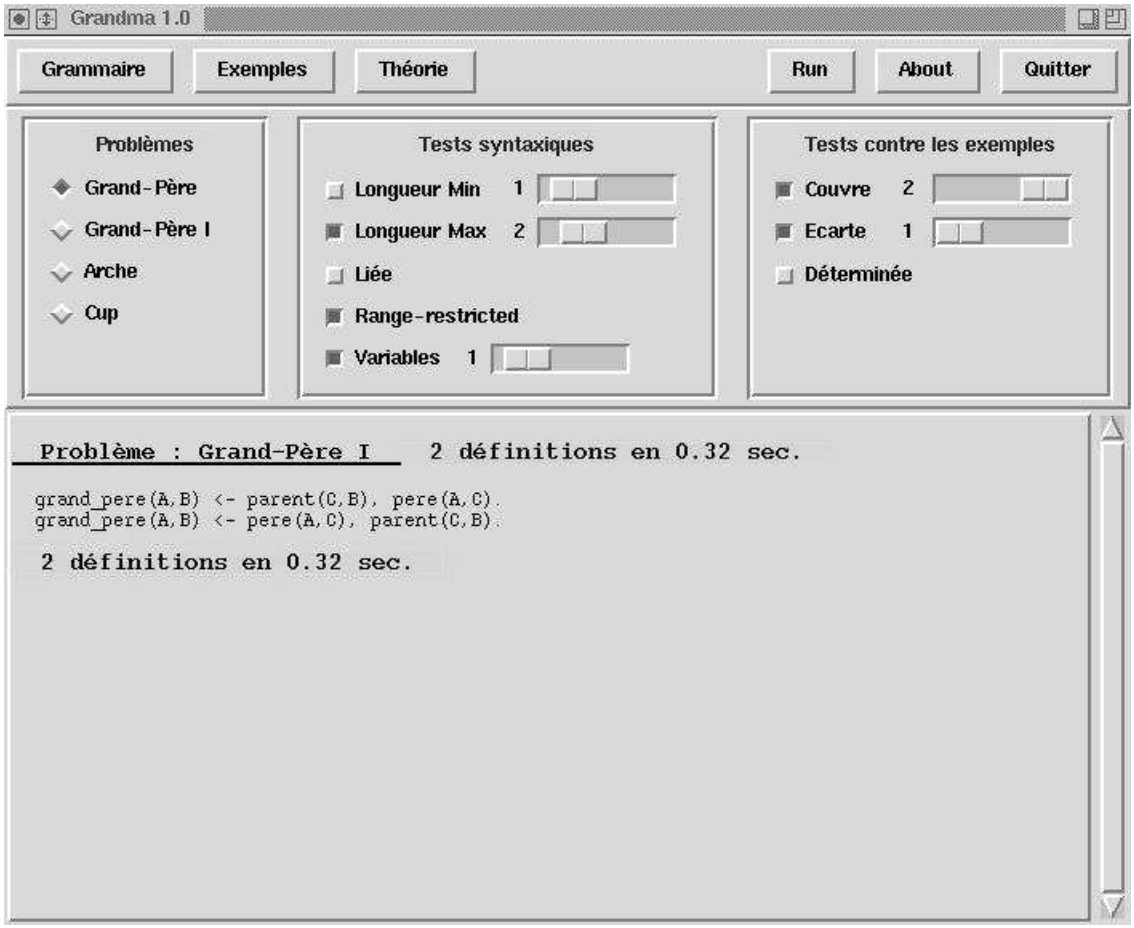


FIGURE 3.5 – Grandma et grand-père avec la grammaire triviale : l'utilisation de toutes les connaissances a priori laisse deux possibilités, permutations l'une de l'autre.

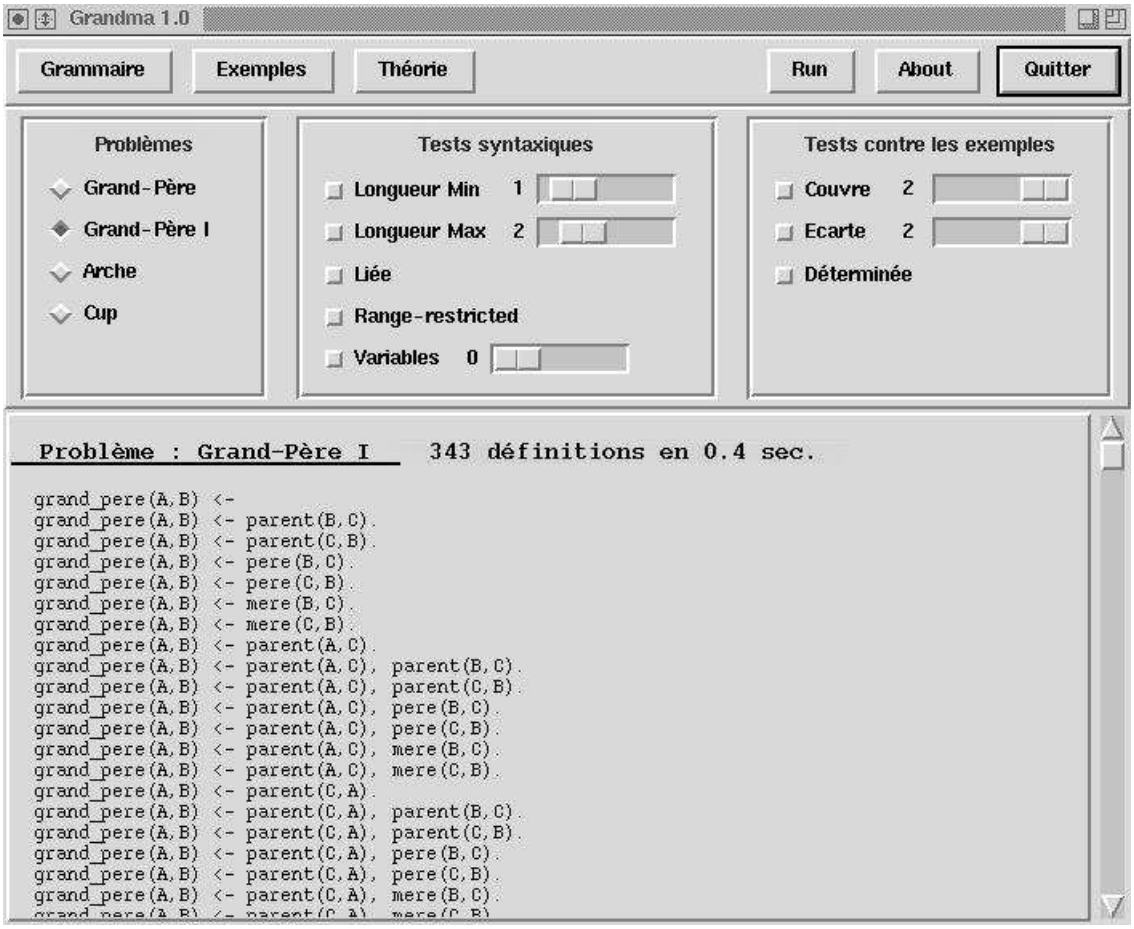


FIGURE 3.6 – Grandma et grand-père avec la grammaire, plus adaptée au problème, de l'exemple 3.2 : le langage dénoté est de taille raisonnable et peut être énuméré sans utiliser d'autres biais.

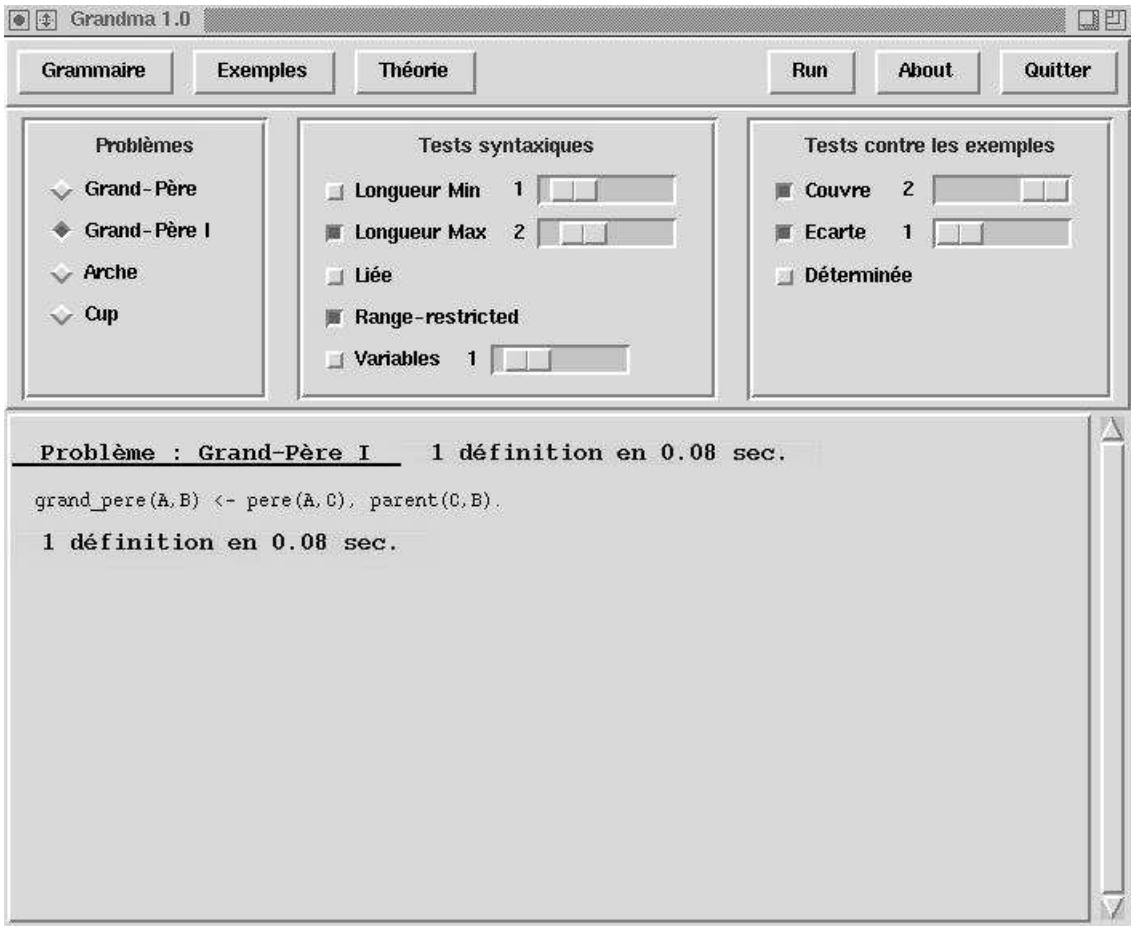


FIGURE 3.7 – Grandma et grand-père avec la grammaire de l'exemple 3.2 : l'utilisation de toutes les connaissances a priori permet de découvrir la solution attendue.

Malgré cette différence, l'esprit est identique : bien que les ensembles d'exemples soient les mêmes, les résultats de l'apprentissage pourront être différents sur un problème et sur son dual. Nous contournons ainsi la barrière posée par le No Free Lunch Theorem : la précision moyenne peut être différente de 50% et donc supérieure à 50%, pour peu que les biais soient bien adaptés à chaque problème.

Nous allons faire un pas de plus dans ce sens en rendant les systèmes d'apprentissage encore plus sensibles aux connaissances disponibles a priori : au chapitre suivant, c'est la relation de généralité elle-même qui va s'adapter aux propriétés connues de la solution.

Bibliographie

[Adé et al., 1995] Adé, H., De Raedt, L., et Bruynooghe, M. (1995). Declarative bias for specific-to-general ILP systems. *Machine Learning*, 20 :119–154.

[Alblas, 1991] Alblas, H. (1991). Introduction to Attribute Grammars. In Alblas, H. et Melichar, B., éditeurs, *Proceedings of International Summer School on Attribute Grammars, Applications and Systems*, volume 545 of *LNCS*, pages 1–15. Springer-Verlag.

[Bergadano et Gunetti, 1995] Bergadano, F. et Gunetti, D. (1995). Learning clauses by tracing derivations. In *ILP : from ML to Software Engineering*. MIT Press.

[Brodley, 1993] Brodley, C. E. (1993). Addressing the selective superiority problem : Automatic algorithm/model class selection. In Utgoff, P., éditeur, *Proceedings 10th International Conference on Machine Learning*, pages 17–24. Morgan Kaufmann.

[Cohen, 1994] Cohen, W. W. (1994). Grammatically biased learning : Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68 :303–366.

[Dehaspe et De Raedt, 1996] Dehaspe, L. et De Raedt, L. (1996). Dlab : A declarative bias formalism. In Rás, Z. W. et Michalewicz, M., éditeurs, *Proceedings of the International Symposium on Methodologies for Intelligent Systems (ISMIS'96)*, volume 1079 of *LNAI*, pages 613–622. Springer-Verlag.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18 :203–226.

[Nédellec et Rouveirol, 1994] Nédellec, C. et Rouveirol, C. (1994). Specifications of the HAIKU system. Rapport interne 928, Laboratoire de Recherche en Informatique, Université Paris Sud.

[Quinlan, 1990] Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3) :239–266.

[Tausend, 1994] Tausend, B. (1994). Representing biases for inductive logic programming. In Bergadano, F. et De Raedt, L., éditeurs, *European Conference on Machine Learning 94*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 427–430. Springer-Verlag.

[Torre, 1995] Torre, F. (1995). Exploitation de biais de langage en apprentissage symbolique automatique dans un cadre logique. Rapport de DEA.

[Torre, 1996] Torre, F. (1996). Utilisation de Grammaires en Programmation Logique Inductive. In Gascuel, O., éditeur, *11èmes Journées Françaises d'Apprentissage (JFA'96), Sète, France*, pages 317–320.

[Utgoff, 1986] Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In Michalski, R. S., Carbonell, J. G., et Mitchell, T. M., éditeurs, *Machine Learning : An Artificial Intelligence Approach*, volume II, pages 107–148. Morgan Kaufmann.

CHAPITRE 4

Relations naturelles et existence d'idéaux

Ces hommes habiles et experts avaient pour principe
qu'il n'y avait pas de manière invariable
pour attaquer ou pour se défendre,
que tout dépendait du terrain sur lequel on se trouvait.
Sun Tse, L'art de la guerre.

Dans le chapitre précédent, nous avons caractérisé les propriétés qui, par rapport à un opérateur fixé, pouvaient élaguer l'espace de recherche en cours de recherche sans risque de perdre une solution : les *propriétés privées*. Cependant, nous avons pu constater que pour les opérateurs traditionnels, peu de propriétés sont privées. Autrement dit, il est possible que l'opérateur utilisé ne rende pas privées les propriétés disponibles.

C'est la raison pour laquelle nous adoptons maintenant l'approche inverse : nous fixons les propriétés qui nous intéressent et dont nous voulons qu'elles produisent un élagage dynamique, puis nous posons la question : quels sont les opérateurs qui rendent ces propriétés privées ?

Nous répondrons en définissant la *relation naturelle* d'une propriété : les opérateurs qui la respectent, rendent la propriété privée.

Cette relation autorise énormément d'opérateurs et, en plus de la privauté des propriétés fixées, nous ajouterons les contraintes de l'*idéalité* définies dans [van der Laag et Nienhuys-Cheng, 1994b], et que nous avons rappelées section 2.4.

Ces contraintes sont très fortes puisque aucun opérateur idéal n'existe pour des relations de généralité comme l'implication logique ou la θ-subsomption. Cependant, nous verrons que certaines relations naturelles autorisent leur existence. Ces opérateurs seront donc nommés *opérateurs naturels idéaux* [Torre et Rouveirol, 1997a, Torre et Rouveirol, 1997b].

4.1 Relations naturelles

La définition que nous avons donnée section 3.2 , page 44, d'une *propriété privée* nous permet de repérer les relations, et donc les opérateurs, qui autoriseront l'élagage par rapport à une propriété. Nous allons maintenant caractériser plus précisément l'ensemble de ces relations.

Beaucoup de relations peuvent rendre une propriété privée. Considérons, par exemple, la relation vide ou la relation identité. Il est clair que ces relations n'ont aucun intérêt pratique pour le parcours d'un espace de recherche quel qu'il soit. Elles rendent pourtant toutes les propriétés privées ! Par conséquent, nous nous focalisons provisoirement sur l'autre extrême : nous cherchons les relations les plus larges (c'est-à-dire maximales pour l'inclusion entre ensembles) qui rendent une propriété privée et nous baptisons une telle relation *relation naturelle* de la propriété.

Définition 4.1 (relation naturelle)
$\mathcal{R}$ *est une* relation naturelle *de la propriété* P *si et seulement si :*
- P *est privée par rapport à* $\mathcal{R}$ *;*
- *s'il existe une relation* $\mathcal{R}'$ *qui rende* P *privée et telle que* $\mathcal{R} \subseteq \mathcal{R}'$ *alors* $\mathcal{R} = \mathcal{R}'$.

Dans cet état, la définition d'une relation naturelle n'est pas d'un grand intérêt pratique. Tâchons de caractériser plus précisément une relation naturelle.

Proposition 4.1 (unicité de la relation naturelle)
Une propriété P *possède une unique relation naturelle, notée* $\succeq_P$. *Cette relation est définie par*

$$\forall H, H' \in \mathcal{L}_h : H \succeq_P H' \Leftrightarrow \forall \left(P(H) \vee \overline{P(H')} \right)$$

La preuve de cette proposition, ainsi que celles des autres résultats de cette section, sont données en annexe, section 9.1.

Par conséquent, une propriété P est privée par rapport à une relation $\mathcal{R}$ si et seulement si $\mathcal{R} \subseteq \succeq_P$. En particulier, un opérateur $\mathcal{O}$ n'autorisera l'élagage dynamique par rapport à une propriété P que si cet opérateur respecte la relation naturelle de P :

$$\mathcal{O} \subseteq \succeq_P$$

ou, autrement dit, si $\mathcal{O}$ est un opérateur descendant par rapport à $\succeq_P$. Cela est illustré Figure 4.1. Notons de plus que la relation $\succeq_P$ est un pré-ordre (elle est à la fois réflexive et transitive).

Poursuivons l'explicitation de la relation naturelle, sous l'hypothèse que toute propriété peut s'exprimer sous la forme $f(H)\ \mathcal{R}\ k$ avec les conditions fixées à la section 3.1. Cette hypothèse permet la simplification suivante.

Proposition 4.2 (définition de la relation naturelle)
Soit P *une propriété définie par*

$$\forall H \in \mathcal{L}_h : P(H) \Leftrightarrow f(H)\ \mathcal{R}\ k$$

La relation naturelle de P *peut s'exprimer sous la forme suivante :*

$$\forall H, H' \in \mathcal{L}_h : H \succeq_P H' \Leftrightarrow f(H)\ \mathcal{R}\ f(H')$$

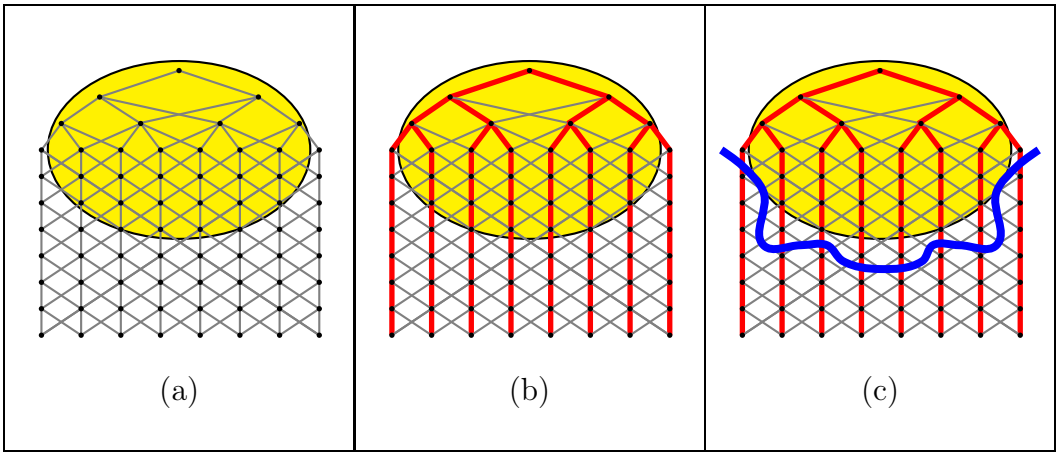


FIGURE 4.1 – Relation naturelle. La figure (a) montre l'organisation de l'espace produite par la relation naturelle, c'est-à-dire la relation la plus large qui rende la propriété privée. Figure (b), on choisit un sous-ensemble de la relation naturelle qui à son tour rend la propriété privée. L'élagage est illustré figure (c).

Exemple 4.1 (relation naturelle de COUVRE$_e$)
Considérons la propriété COUVRE$_e$ *qui requiert qu'une hypothèses H couvre un exemple e, c'est-à-dire $H \models e$. La fonction f est ici l'identité, $\mathcal{R}$ est l'implication logique $\models$ et le paramètre de la propriété est e, l'exemple à couvrir. En appliquant la proposition ci-dessus, on trouve que la relation naturelle de* COUVRE$_e$ *est définie par :*

$$\forall H, H' \in \mathcal{L}_h : H \succeq_{Couvre} H' \Leftrightarrow H \models H'$$

Ainsi, demander qu'un opérateur rende privée la propriété COUVRE$_e$ revient à imposer que cet opérateur respecte une relation de généralité. Les relations naturelles étendent donc la notion de relation de généralité à l'ensemble des propriétés souhaitées pour la solution.

La table 4.1 présente les relations naturelles des propriétés auxquelles nous nous intéressons.

Maintenant que nous connaissons les relations naturelles des propriétés élémentaires, nous allons nous pencher sur les relations naturelles de leurs compositions. En particulier, nous allons considérer la propriété ÉCARTE$_e$, propriété duale de COUVRE$_e$.

Proposition 4.3 (propriété duale)
Soit P une propriété et $\succeq_P$ sa relation naturelle,

$$\forall H, H' \in \mathcal{L}_h : H \succeq_{\overline{P}} H' \Leftrightarrow H' \succeq_P H$$

Exemple 4.2 (relation naturelle de ÉCARTE$_e$)
La propriété ÉCARTE$_e$ *est la propriété duale de* COUVRE$_e$ *; sa relation naturelle est donc :*

$$\forall H, H' \in \mathcal{L}_h : H \succeq_{\acute{E}carte} H' \Leftrightarrow H' \models H$$

Après les propriétés duales, considérons les conjonctions de propriétés.

TABLE 4.1 – Relations naturelles.

COUVRE_e	$H \models H'$
$\text{RANGE-RESTRICTION}_b$	Range-restricted(H) = Range-restricted(H')
CONNEXION_b	Connectée(H) = Connectée(H')
RÉDUCTION_b	Réduite(H) = Réduite(H')
TAILLEMAX_n	$\|H\| \leq \|H'\|$
TAILLEMIN_n	$\|H\| \geq \|H'\|$
NBVARSMAX_n	NbVars(H) $\leq$ NbVars(H')
NBVARSMIN_n	NbVars(H) $\geq$ NbVars(H')
PROFMAX_n	Prof(H) $\leq$ Prof(H')
PROFMIN_n	Prof(H) $\geq$ Prof(H')
DEGRÉMAX_n	Degré(H) $\leq$ Degré(H')
DEGRÉMIN_n	Degré(H) $\geq$ Degré(H')

Proposition 4.4 (conjonction de propriétés)
Soient deux propriétés P_1 et P_2, et leurs relations naturelles respectives $\succeq_{P_1}$ et $\succeq_{P_2}$. La relation naturelle de leur conjonction est définie par

$$\forall H, H' \in \mathcal{L}_h : H \succeq_{P_1 \wedge P_2} H' \Leftrightarrow (H \succeq_{P_1} H') \wedge (H \succeq_{P_2} H')$$

Exemple 4.3 (complétude)
Si l'on exprime la complétude comme une conjonction de propriétés COUVRE_{e_i} *pour chaque e_i de A^+, il vient que la relation naturelle de la complétude est l'implication logique, comme attendu.*

Cela amène une nouvelle justification de ce que disait Peter Idestam-Almquist [Idestam-Almquist, 1995] :

> Implication is the most natural and straightforward basis for generalization in inductive learning, since the concept of induction can be defined as the inverse of logical entailment.

Exemple 4.4 (Longueur maximale et complétude)
On s'intéresse à une association $\mathcal{P}$ définie par

$$\mathcal{P} = \{\text{TAILLEMAX}_k, \text{COUVRE}_e\}$$

La relation naturelle de $\mathcal{P}$ est définie par

$$\forall H, H' \in \mathcal{L}_h : H \succeq_{\mathcal{P}} H' \Leftrightarrow (|H| \leq |H'|) \wedge (H \models H')$$

Nous savons maintenant où chercher nos opérateurs de raffinement pour qu'ils aient un pouvoir élagant : à l'intérieur des relations naturelles des propriétés à satisfaire. Cette indication laisse encore beaucoup de possibilités (Figure 4.1) et il est impératif de spécifier en plus un comportement attendu pour nos opérateurs. Nous avons choisi de nous intéresser aux opérateurs *idéaux*. Nous savons qu'il n'en existe pas sous θ-subsomption ; le reste du chapitre est consacré à la question : peut-il en exister pour nos relations naturelles ?

4.2 Conditions d'existence d'un opérateur idéal

4.2.1 Préliminaires

Nous avons donné la définition des opérateurs idéaux section 2.4 (page 35 et, comme nous l'avions indiqué alors, il a été prouvé que de tels opérateurs n'existaient pas pour des espaces non restreints ordonnés par la θ-subsomption ou l'implication logique. Cette non-existence d'opérateurs idéaux a deux raisons précisément identifiées : les chaînes infinies non couvertes et les ensembles couvrants infinis [van der Laag et Nienhuys-Cheng, 1994b, van der Laag et Nienhuys-Cheng, 1994a, van der Laag, 1995].

Pour appréhender ces deux problèmes, il est nécessaire de définir les notions de *couverture* et d'*ensemble couvrant*.

Définition 4.2 (couverture)
C est une couverture supérieure *de D si et seulement si*

1. *C subsume strictement D : $C \succ D$;*
2. *il n'existe pas de E entre C et D qui ne soit pas équivalente à l'une de ces clauses, c'est-à-dire : $C \succ E \succ D$.*

On dit aussi dans ce cas, que D est une couverture inférieure *de C.*

Définition 4.3 (ensemble couvrant)
Un ensemble couvrant inférieur (resp. supérieur) d'une clause C est un ensemble maximal pour l'inclusion de couvertures non équivalentes inférieures (resp. supérieures) de C.

Ces notions, illustrées Figure 4.2, sont particulièrement pertinentes lorsque l'on s'intéresse à l'idéalité car il a été prouvé qu'un opérateur idéal appliqué à une hypothèse quelconque H fournissait, au minimum, un ensemble couvrant de H. Dès lors, l'impossibilité de calculer l'ensemble couvrant d'une hypothèse va entraîner la non-existence d'opérateurs idéaux. Cette impossibilité peut avoir deux origines.

Le premier cas est bien connu sur l'ensemble des réels (Figure 4.3) et révèle l'impossibilité de faire un plus petit pas (on dit alors qu'il y a une chaîne infinie non

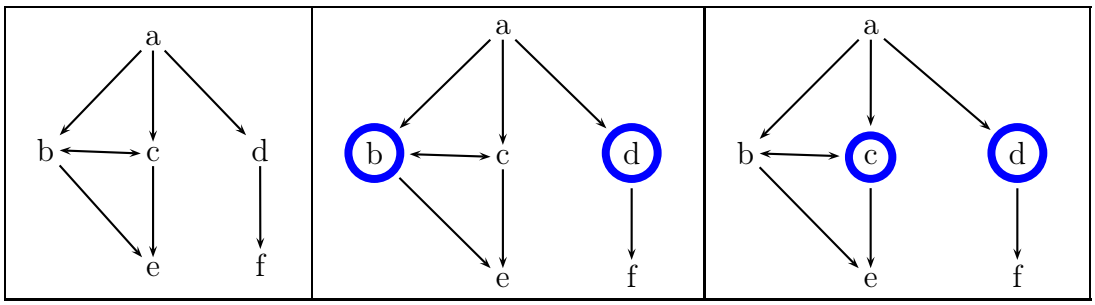


FIGURE 4.2 – Couvertures et ensembles couvrants. a possède trois couvertures qui sont b, c et d. Puisque b et c sont équivalents, ils ne peuvent pas apparaître dans le même ensemble couvrant. a a donc deux ensembles couvrants possibles : $\{b, d\}$ et $\{c, d\}$.

couverte). Considérons l'entier 3 et la suite $\{u_i\}_{i\geq 0}$ définie par $u_i = 3 + \left(\frac{1}{2}\right)^i$: aucun u_i ne couvre 3 car il y a toujours une infinité d'éléments de la suite entre 3 et un u_i quelconque. On dit que $\{u_i\}_{i\geq 0}$ est une chaîne infinie non couverte de 3.

La seconde possibilité est « simplement » que l'ensemble couvrant est infini.

La question est maintenant de savoir si ces phénomènes peuvent apparaître dans des espaces non restreints ordonnés par nos relations naturelles. Les résultats présentés dans cette section sont démontrés en annexe, section 9.2.

Nous allons nous restreindre au cas où l'on utilise une propriété liée aux exemples et un biais de langage. Pour les exemples, nous avons vu que la relation naturelle est l'implication logique. Celle-ci étant indécidable [Schmidt-Schauss, 1988], nous utiliserons la θ-subsomption qui lui est équivalente dans beaucoup de cas pratiques [Gottlob, 1987].

Notation 4.1
La relation naturelle de Couvre$_e$ *et d'une autre propriété* $f(H)\,\mathcal{R}\,k$ *sera noté* $\succeq_f^\theta$.

Commençons par le cas de deux propriétés, l'une impliquant l'autre ; la relation naturelle de l'une est incluse dans celle de l'autre.

Proposition 4.5 (inclusion de relations)
Soient $\succeq_a$ *et* $\succeq_b$ *deux pré-ordres vérifiant :*

$$\forall C, D \in \mathcal{L}_h : C \succeq_a D \Rightarrow C \succeq_b D$$

Un opérateur $\mathcal{O}$ *est idéal pour* $\succeq_a \cap \succeq_b$ *si et seulement si* $\mathcal{O}$ *est idéal pour* $\succeq_a$.

Cette proposition, très simple, a une application directe. Elle nous permet de régler immédiatement le cas de la propriété ProfMax$_p$. La θ-subsomption et la profondeur sont en effet liées par l'implication suivante :

$$\forall C, D \in \mathcal{L}_h : C \succeq^\theta D \Rightarrow \text{Prof}(C) \leq \text{Prof}(D)$$

Par suite, l'utilisation de la propriété ProfMax$_p$ et de sa relation naturelle ne permettent pas de dépasser l'inexistence d'opérateurs idéaux induite par la θ-subsomption.

Voyons maintenant si les autres propriétés dont nous disposons autorisent ou non la présence de chaînes infinies dans $\mathcal{L}_h$.

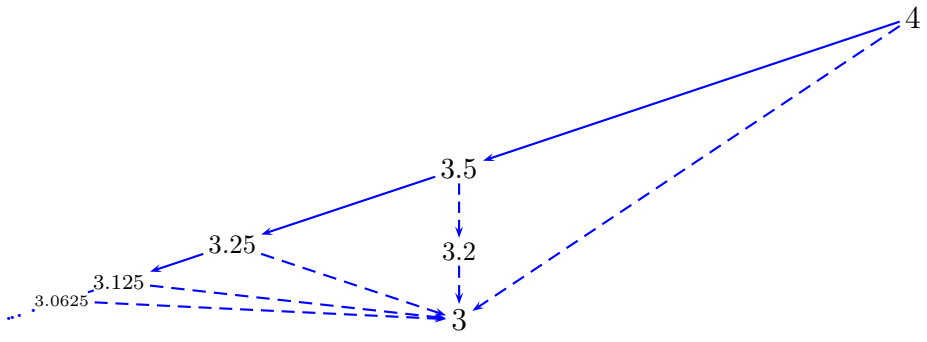


Figure 4.3 – Chaîne infinie dans $\mathbb{R}$.

4.2.2 Chaînes infinies non couvertes

Donnons, tout d'abord une définition formelle de ces chaînes.

Définition 4.4 (chaînes infinies non couvertes)
Une chaîne infinie non couverte strictement descendante *d'une clause C pour $\succeq$ est un ensemble infini $\{D_i\}_{i\geq 1}$ de clauses de $\mathcal{L}_h$ telles que*

$$D_1 \succ D_2 \succ \ldots \succ D_n \succ D_{n+1} \succ \ldots \succ C$$

et C ne possède de couverture supérieure E dans $\mathcal{L}_h$ qui soit couverte par toutes les clauses D_i. Symétriquement, une chaîne infinie non couverte strictement ascendante *d'une clause C pour $\succeq$ est un ensemble infini $\{A_i\}_{i\geq 1}$ de clauses de $\mathcal{L}_h$ telles que*

$$C \succ \ldots \succ A_{n+1} \succ A_n \succ \ldots \succ A_2 \succ A_1$$

et C ne possède de couverture inférieure F dans $\mathcal{L}_h$ qui couvre toutes les clauses A_i.

Ces définitions sont illustrées par la Figure 4.4 et la Figure 4.5 illustre l'exemple suivant sur les clauses.

Exemple 4.5 (chaîne infinie non couverte)
Sous θ-subsomption, $\{D_i\}_{i\geq 2}$ est une chaîne infinie non couverte de C avec :

$$\begin{cases} C & : q(X_1) \leftarrow p(X_1, X_1) \\ D_n & : q(X_1) \leftarrow \{p(X_i, X_j) | 1 \leq i, j \leq n, i \neq j\} \end{cases}$$

Il est important de se convaincre que l'existence d'une telle chaîne est directement liée à la relation utilisée pour ordonner l'ensemble et non pas à cet ensemble lui-même. En particulier, le phénomène décrit à la Figure 4.3 n'est pas implacablement entraîné par le fait que $\mathbb{R}$ n'est pas dénombrable (considérons par exemple $\mathbb{R}$ doté de la relation « x est plus petit que y si x est un préfixe de y»). C'est précisément là qu'est notre espoir : nous travaillons sur le même espace $\mathcal{L}_h$, mais nous allons l'organiser de manière moins conventionnelle : en utilisant les relations naturelles.

Notre démarche consiste à caractériser ces chaînes sous θ-subsomption. Cela nous permettra de déterminer les relations naturelles qui ont une chance de briser ces chaînes.

Lemme 4.1 (augmentation du nombre de variables)
Soit $\{D_i\}_{i\geq 1}$ une chaîne infinie non couverte d'une clause C sous θ-subsomption. Dans $\{D_i\}_{i\geq 1}$, le nombre de variables augmente à l'infini.

Nous avons établi que certaines quantités augmentent *nécessairement* le long d'une chaîne (ascendante ou descendante) sous θ-subsomption : le nombre de variables existentielles mais aussi, par suite, la taille des clauses et le nombre d'occurrences d'au moins un symbole de prédicats (notons que la profondeur ne croît pas dans une chaîne).

Parmi les quantités citées, seule la dernière ne correspond pas à un biais de langage identifié. D'où, l'introduction d'un nouveau biais, appelé MaxOcc$_n$, correspondant à la limitation des occurrences des symboles de prédicat. Cela signifie

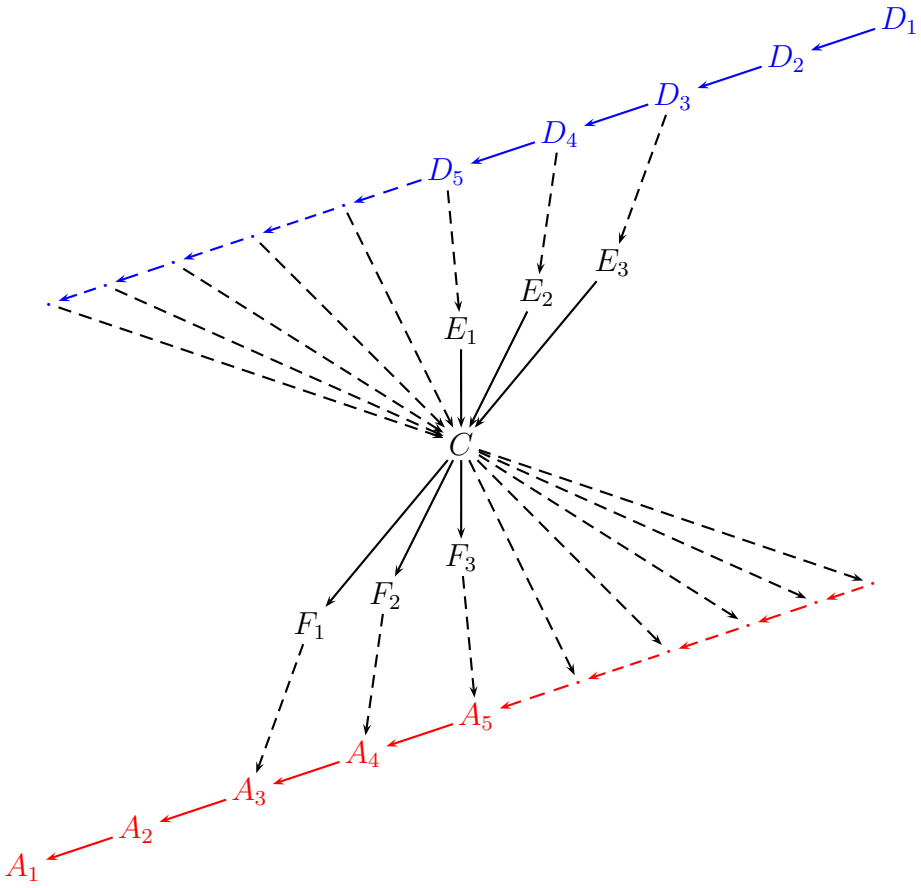


FIGURE 4.4 – Chaînes infinies non couvertes

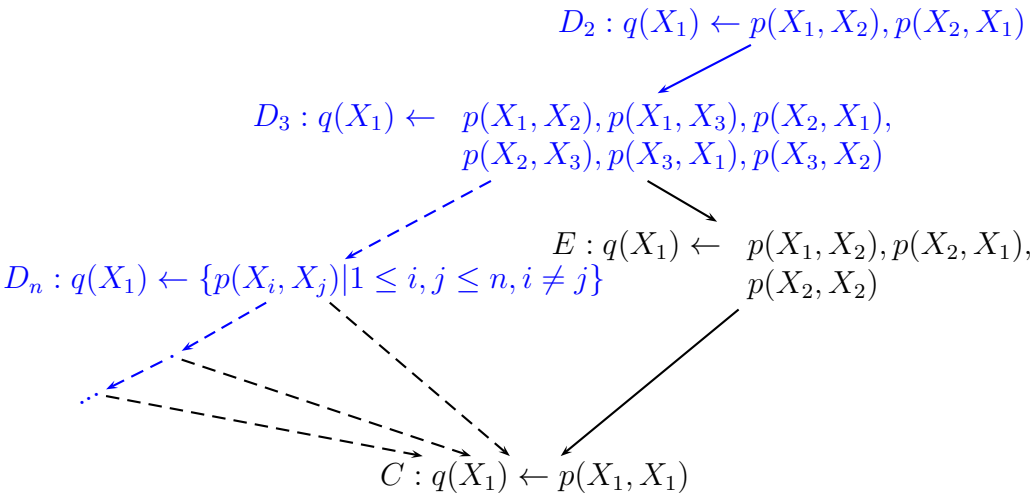


FIGURE 4.5 – Chaînes infinies non couvertes sous θ-subsomption. $\{D_i\}_{i \geq 2}$ est une chaîne infinie non couverte de C.

que l'on peut poser soit une borne valable pour tous les symboles de prédicat, soit une borne différente pour chacun de ces symboles. Ce biais, bâti à partir de l'étude théorique des chaînes infinies non couvertes, a des utilisations bien pratiques. On pensera en particulier à l'application liée à la mutagénèse, pour laquelle on peut avoir envie d'exprimer des contraintes comme : *la définition d'une molécule active doit faire intervenir, au plus, trois cycles benzéniques.*

Revenons à nos chaînes. Dans notre cadre, l'espace est ordonné par la conjonction de la θ-subsomption et d'une relation naturelle $\succeq_P$ définie par

$$C \succeq_P D \Leftrightarrow f(C)\,\mathcal{R}\,f(D)$$

avec f est une fonction dont le domaine $\mathcal{D}_f$ est dénombrable.

Cela nous permet de caractériser les chaînes de $\mathcal{L}_h$ ordonné par $\succeq_f$.

Lemme 4.2 (valeur constante)
Soit $\{D_i\}_{i\geq 1}$ une chaîne infinie non couverte d'une clause C, sous une relation de généralité faisant intervenir $\succeq_f$. Il existe k une valeur de $\mathcal{D}_f$ telle que, pour tout i suffisamment grand, $f(D_i) = k$.

Notre premier lemme montrait que certaines quantités augmentaient le long d'une chaîne sous θ-subsomption. Ce dernier lemme indique qu'une chaîne infinie non couverte dans un espace ordonné par $\succeq_f$ possède une infinité d'éléments qui ont même valeur par f. Il est maintenant aisé de caractériser finement les éventuelles chaînes de $\mathcal{L}_h$ ordonné par $\succeq_f^\theta$.

Théorème 4.1 (chaînes sous $\succeq_f^\theta$)
Il existe une chaîne infinie non couverte dans $\mathcal{L}_h$, ordonné par $\succeq_f^\theta$, si et seulement si il existe un sous-ensemble S de $\mathcal{L}_h$ tel que :
- *toutes les hypothèses de S ont la même valeur par f ;*
- *le nombre de variables dans les hypothèses de S n'est pas borné.*

Par suite, si le fait de fixer la valeur de f empêche le nombre de variables de croître, alors il ne peut exister de chaînes infinies non couvertes dans notre espace. Avec ce critère, nous tenons une caractérisation des propriétés ne faisant pas apparaître de chaînes infinies non couvertes. Les candidates sont, au premier chef, les propriétés qui consistent à borner l'une des quantités pertinentes : limitation du nombre de variables, de la taille des clauses ou du nombre d'occurrences d'un symbole de prédicat. Malheureusement, les autres biais de langage que nous avons cités (profondeur, degré, range-restriction, connexion, etc.) préservent les chaînes infinies non couvertes de la θ-subsomption (en particulier, la chaîne de la Figure 4.5 est toujours valable dans ces cas).

Nous nous sommes limités à une unique propriété mais l'argument reste valable pour un espace ordonné par une conjonction de taille quelconque : si l'une des propriétés empêche l'une des quantités pertinentes de croître, l'espace ne contiendra pas de chaînes infinies non couvertes. Ainsi, nous sommes certains qu'un espace ordonné par une conjonction faisant intervenir la relation naturelle d'une des trois propriétés citées ci-dessus, ne présentera pas de chaînes infinies non couvertes.

Même si nous sommes maintenant capables d'ordonner l'espace de recherche de manière à ne pas avoir de chaîne infinie non couverte, cette espace peut encore contenir des clauses ayant des ensembles couvrants infinis.

4.2.3 Ensembles couvrants infinis

Nous passons donc maintenant au second cas empêchant l'existence d'opérateurs idéaux, celui où des clauses de l'espace possède un ensemble couvrant infini. Par définition, cela se produit si pour une hypothèse on peut trouver une infinité d'hypothèses qui lui soient immédiatement supérieures et incomparables entre elles.

Il est plus délicat d'identifier les propriétés produisant ce phénomène. Nous donnons simplement une condition suffisante pour l'existence d'ensembles couvrants infinis.

Proposition 4.6 (ensembles couvrants infinis)
Si dans $\mathcal{L}_h$ on peut trouver une clause C et une infinité de clause D_i vérifiant :
- *les clauses D_i sont incomparables entre elles sous θ-subsomption ;*
- *C θ-subsume strictement tous les D_i : $C \succ^\theta D_i$;*
- *il n'existe pas de E telle que $C \succ^\theta E \succ^\theta D_i$;*
- *pour toutes les clause D_i : $f(D_i) = f(C)$.*

alors C possède un ensemble couvrant infini sous $\succeq^\theta_f$.

Cette simple reformulation d'un ensemble couvrant infini fournit donc une condition nécessaire pour l'existence d'idéaux.

Cette condition est remplie si l'on utilise les propriétés $\textsc{Couvre}_e$ et $\textsc{NbVarsMax}_n$. La relation naturelle associée à ces propriétés est définie par :

$$\forall C, D \in \mathcal{L}_h : C \succeq^\theta_{\text{NbVars}} D \Leftrightarrow (C \succeq^\theta D) \wedge (\text{NbVars}(C) \leq \text{NbVars}(D))$$

Nous l'avons vu, cette relation empêche l'apparition de chaîne infinie. Considérons maintenant la clause C suivante

$$q \leftarrow$$

et les clauses D_i définies par

$$q \leftarrow p(f^i(a))$$

C couvre chacun des D_i. Les D_i sont incomparables entre elles pour la θ-subsomption et donc pour notre relation. De plus, les D_i n'ont, comme C, aucune variable : par conséquent, aucune clause possédant une variable ne peut se glisser entre C et une clause D_i. Par suite, la clause C possède un ensemble couvrant inférieur infini et il n'y aura pas d'opérateur idéal sous $\succeq^\theta_{\text{NbVars}}$.

En revanche, cela ne survient pas pour les autres propriétés et il nous reste comme « bonnes » propriétés les limitations sur les quantités suivantes :
- la taille des clauses ;
- le nombre d'occurrences d'un prédicat ;
- et toujours le nombre de variables mais uniquement dans le cas ascendant.

Nous avons donné des conditions nécessaires pour l'existence d'opérateurs d'idéaux. Cela nous a permis de trier nos propriétés et d'isoler celles qui sont les plus pertinentes pour le problème de l'idéalité sous relation naturelle. Mais, même dans ces cas,

l'existence d'un opérateur idéal n'est pas garanti. Il n'y a pas, a priori, de méthode de construction universelle et pour chaque relation, il faut rechercher un éventuel opérateur idéal et, ensuite, prouver son idéalité. C'est la démarche que nous allons suivre à la section suivante pour $\rho^{\theta}_{||}$, un opérateur idéal sous $\succeq^{\theta}_{||}$.

4.3 L'opérateur $\rho^{\theta}_{||}$

Nous avons vu qu'un opérateur idéal devait calculer au minimum un ensemble couvrant d'une clause. Notre propos est de calculer *exactement* un ensemble couvrant. Cela n'implique pas pour autant l'idéalité [van der Laag, 1995] qu'il nous faudra prouver par ailleurs. Cependant, cela garantit que l'opérateur sera aussi efficace que possible puisque, pour chaque hypothèse, nous calculerons systématiquement le nombre minimum de raffinements.

4.3.1 Description de $\rho^{\theta}_{||}$

Définition

L'espace de recherche étant ordonné par la relation

$$C \succeq^{\theta}_{||} D \Leftrightarrow (C \succeq^{\theta} D) \wedge (|C| \leq |D|)$$

on définit l'opérateur $\rho^{\theta}_{||}$ comme suit.

Définition 4.5 (opérateur $\rho^{\theta}_{||}$)
Soit C une clause de l'espace de recherche $\mathcal{L}_h$. D est un raffinement de C dans un des cas suivants.

1. *(Ajout) $D = C \cup \{p(\ldots, Y_i, \ldots)\}$, avec p un symbole de prédicat, et les Y_i sont de nouvelles variables, c'est-à-dire des variables n'apparaissant pas dans C.*
2. *(Unification) $D = C\{X_1/X_2\} \cup \{p(\ldots, Y_i, \ldots)\}^*$, C clause réduite et $|C| = |D|$, avec p symbole de prédicat, X_1 et X_2 variables de C, et les Y_i sont de nouvelles variables.*
3. *(Unification) $D = C\{X_1/X_2\} \cup \{p(\ldots, Y_i, \ldots)\}^*$, C clause non réduite et $|C| = |D|$, avec p symbole de prédicat, X_1 et X_2 variables de C telles que $X_1/X_2 \notin \sigma$ pour tout σ vérifiant $C\sigma \subseteq C$, et les Y_i sont de nouvelles variables.*
4. *(Équivalentes) D est obtenue par application d'un des points précédents sur les clauses équivalentes à C (θ-équivalente et de même taille). Nous verrons que ce point ne peut s'appliquer qu'aux clauses non réduites.*
5. *(Substitution de prédicats) D est obtenue par applications successives du point 1 sur un sous-ensemble équivalent de C (cela suppose donc que C n'est pas réduite), $|C| = |D|$ et D contient au moins un symbole de prédicat n'apparaissant pas dans C.*

La preuve de l'idéalité de cet opérateur est donnée, en annexe, à la section 9.3.

Explications

Nous allons illustrer les raffinements effectués par notre opérateur sur le concept grand-père (gp) : les prédicats disponibles sont pa pour parent, p pour père et m pour mère.

Soient les clauses suivantes :

$$C_1 : gp(A, B) \leftarrow p(A, C)$$
$$C_2 : gp(A, B) \leftarrow p(A, C), p(C, A)$$
$$C_3 : gp(A, B) \leftarrow p(A, C), p(D, C)$$

Notons immédiatement que C_1 et C_2 sont réduites, mais que C_3 ne l'est pas.

Le point 1 ajoute des littéraux avec de nouvelles variables. Nous disposons de trois symboles de prédicat. Par suite, il y a trois raffinements possibles de C_1 par cette transformation :

$$gp(A, B) \leftarrow p(A, C), pa(D, E)$$
$$gp(A, B) \leftarrow p(A, C), p(D, E)$$
$$gp(A, B) \leftarrow p(A, C), m(D, E)$$

Le point 2 unifie deux variables d'une clause réduite. Ainsi, sur C_1 qui dispose de trois variables, il y a trois unifications possibles et l'on obtient donc trois raffinements de C_1 par cette transformation :

$$gp(A, A) \leftarrow p(A, C) \quad \text{(unification de } A \text{ et } B\text{)}$$
$$gp(A, B) \leftarrow p(A, A) \quad \text{(unification de } A \text{ et } C\text{)}$$
$$gp(A, B) \leftarrow p(A, B) \quad \text{(unification de } B \text{ et } C\text{)}$$

Un problème peut se poser si l'unification réduit la taille de la clause. Ainsi, sur C_2, l'unification de A et C produit la clause suivante :

$$gp(A, B) \leftarrow p(A, A)$$

Or, notre relation $\succeq^{\theta}_{||}$ ne nous permet pas de revenir à une clause de taille plus petite. Dans ce cas, l'opérateur nous autorise à ajouter des littéraux jusqu'à retrouver la taille de C_2 et il vient les clauses suivantes pour le raffinement de C_2 par unification de A et C.

$$gp(A, B) \leftarrow p(A, A), pa(D, E)$$
$$gp(A, B) \leftarrow p(A, A), p(D, E)$$
$$gp(A, B) \leftarrow p(A, A), m(D, E)$$

Grâce au point 3, on réalise la même chose sur des clauses non réduites mais en s'assurant que l'unification ne produit pas de clauses équivalentes. Par exemple, l'unification de A et D sur C_3 aurait produit la clause équivalente suivante :

$$gp(A, B) \leftarrow p(A, C)$$

L'application du point 4 sur C_3 est plus délicate. On commence par construire les clauses qui lui sont équivalentes :

$$gp(A, B) \leftarrow p(A, C), p(D, E)$$
$$gp(A, B) \leftarrow p(A, C), p(A, D)$$

Puis, on applique les points précédents (ajout et unification sur ces clauses).

Enfin, le point 5, permet de substituer des littéraux redondants. Par exemple, sur C_3, on remplace le littéral redondant $p(D, C)$ par un littéral construit avec un symbole de prédicat qui n'apparaît pas dans C_3 et de nouvelles variables, ce qui donne :

$$gp(A, B) \leftarrow p(A, C), pa(E, F)$$
$$gp(A, B) \leftarrow p(A, C), m(E, F)$$

4.4 Bilan

Notre but dans ce chapitre était de caractériser les relations qui rendaient une propriété privée et, plus particulièrement, de construire des opérateurs permettant l'élagage par rapport à cette propriété.

Nous avons adopté les bases de [van der Laag et Nienhuys-Cheng, 1994b] pour ce qui est la spécification des opérateurs de raffinement. Ce choix nous a confronté à leur résultat de non-existence d'opérateurs idéaux sous θ-subsomption.

Notre démarche était toute différente car motivée par la nécessité d'élaguer l'espace de recherche. Nous avons défini la *relation naturelle* d'une propriété et montré que seul le respect de cette propriété pouvait permettre un élagage dynamique. Nous nous sommes ensuite interrogés sur la manière de construire un opérateur fondé sur cette relation et, en particulier, sur l'obtention d'un opérateur idéal. L'étude de ces opérateurs nous a permis de caractériser les propriétés favorables à leur existence et d'introduire un nouveau biais de langage ayant cette particularité. Finalement, nous avons exhibé un opérateur idéal.

Tout comme d'ailleurs [Champesme et al., 1995a] qui définit une *relation de subsomption empirique* autorisant l'existence d'opérateurs idéaux ou encore [Esposito et al., 1996] qui fait de même avec un pré-ordre dédié aux clauses DATALOG.

Dans l'esprit, notre approche est plus semblable à celle de [Shapiro, 1981]. Celui-ci utilise une mesure nommée *size* pour construire un opérateur de raffinement calculable : il indique que q est un raffinement de p si p implique q et que $\text{size}(p) \leq \text{size}(q)$. Cela ressemble fortement à l'une de nos relations naturelles. Cependant, des restrictions très fortes pèsent sur *size* : cette fonction doit prendre ses valeurs dans $\mathbb{N}$ et, pour tout n entier, l'ensemble des hypothèses H telles que $\text{size}(H) = n$ est nécessairement fini. À l'opposé, nos propriétés sont beaucoup moins contraintes. En contrepartie, elles ne permettent pas toutes l'existence d'un opérateur idéal.

Il faut maintenant constater que $\succeq^{\theta}_{||}$, relation pour laquelle nous avons trouvé un opérateur idéal, est une relation qui était propice à cette découverte. Il est courant d'ajouter des littéraux pour spécialiser sous θ-subsomption. Par conséquent, suivre $\succeq^{\theta}_{||}$, c'est-à-dire spécialiser pour la θ-subsomption tout en augmentant la taille des clauses est un cheminement qui semble parfaitement naturel. Même si la propriété TAILLEMAX$_n$ n'est pas privée pour la θ-subsomption, notre intuition indique que ces propriétés, COUVRE$_e$ et TAILLEMAX$_n$, vont dans le même sens. En particulier, la clause vide vérifie ces propriétés et reste l'élément le plus général de $\mathcal{L}_h$.

C'est ce que nous allons étudier au chapitre suivant. Que se passe t-il si l'on utilise la relation naturelle de propriétés contradictoires ? Peut-on encore trouver des opérateurs de raffinements ? Quelles seront alors les clauses de départ ?

De plus, nous reviendrons sur le choix des opérateurs idéaux dont il faut convenir, comme le font les créateurs mêmes de ce type d'opérateurs [Nienhuys-Cheng et de Wolf, 1997], que les idéaux ont un intérêt pratique très limité : à cause de leur définition de la complétude, un opérateur idéal contraint le générer-et-tester à considérer sans cesse les mêmes clauses. Nous nous tournerons vers l'alliance des relations naturelles avec, cette fois-ci, les opérateurs *optimaux* [De Raedt et Bruynooghe, 1993].

Bibliographie

[Champesme et al., 1995] Champesme, M., Brézellec, P., et Soldano, H. (1995). Empirically conservative search space reductions. In Raedt, L. D., éditeur, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 387–402. Department of Computer Science, Katholieke Universiteit Leuven.

[De Raedt et Bruynooghe, 1993] De Raedt, L. et Bruynooghe, M. (1993). A theory of clausal discovery. In Bajcsy, R., éditeur, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann.

[Esposito et al., 1996] Esposito, F., Laterza, A., Malerba, D., et Semeraro, G. (1996). Refinement of datalog programs. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD)*, pages 73–94.

[Gottlob, 1987] Gottlob, G. (1987). Subsumption and implication. *Information Processing Letters*, 24(2) :109–111.

[Idestam-Almquist, 1995] Idestam-Almquist, P. (1995). Generalization of clauses under implication. *Journal of Artificial Intelligence Research*, 3 :467–489.

[Nienhuys-Cheng et de Wolf, 1997] Nienhuys-Cheng, S. et de Wolf, R. (1997). *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence Tutorial*. Springer-Verlag.

[Schmidt-Schauss, 1988] Schmidt-Schauss, M. (1988). Implication of clauses is undecidable. *TCS : Theoretical Computer Science*, 59(3) :287–296.

[Shapiro, 1981] Shapiro, E. Y. (1981). Inductive inference of theories from facts. Rapport interne 192, Yale University Department of Computer Science.

[Torre et Rouveirol, 1997a] Torre, F. et Rouveirol, C. (1997a). Natural ideal operators in inductive logic programming. In van Someren, M. et Widmer, G., éditeurs, *Proceedings of the ninth European Conference on Machine Learning*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 274–289. Springer-Verlag.

[Torre et Rouveirol, 1997b] Torre, F. et Rouveirol, C. (1997b). Opérateurs naturels en programmation logique inductive. In Soldano, H., éditeur, *12èmes Journées Françaises d'Apprentissage (JFA'97)*, pages 53–64.

[van der Laag et Nienhuys-Cheng, 1994a] van der Laag, P. et Nienhuys-Cheng, S. H. (1994a). A note on ideal refinement operators in ILP. In Wrobel, S., éditeur, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 247–262. Gesellschaft für Mathematik und Datenverarbeitung MBH.

[van der Laag, 1995] van der Laag, P. R. J. (1995). *An Analysis of Refinement Operators in Inductive Logic Programming*. Thèse de doctorat, Erasmus Universiteit, Rotterdam, the Netherlands.

[van der Laag et Nienhuys-Cheng, 1994b] van der Laag, P. R. J. et Nienhuys-Cheng, S. (1994b). Existence and nonexistence of complete refinement operators. In Bergadano, F. et de Raedt, L., éditeurs, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag.

CHAPITRE 5

Opérateurs parfaits

Dans le chapitre précédent, nous avons défini la *relation naturelle* d'une propriété : si elle est respectée par l'opérateur, l'élagage dynamique est possible par rapport à cette propriété. Pour un espace de recherche ordonné par l'une de ces relations naturelles, nous avons trouvé un opérateur idéal, alors qu'il n'en existe pas sous θ-subsomption.

Nous revenons maintenant à notre préoccupation première : l'efficacité de l'algorithme du générer-et-tester et à l'utilisation des propriétés connues de la solution pour améliorer la recherche de celle-ci.

Nous commencerons par montrer que les opérateurs idéaux et optimaux ne peuvent finalement pas nous convenir :

— les premiers ne sont pas efficaces ;
— les seconds sont plus efficaces mais mal adaptés, dans leur définition originelle, à notre volonté d'intégration des biais de langages.

Nous allons donc définir nos propres opérateurs, les *opérateurs parfaits*. En même temps, nous aurons à cœur de caractériser plus formellement les propriétés qui semblent contradictoires et de les utiliser tout de même. Enfin, nous verrons que même les propriétés non privées peuvent rendre l'apprentissage plus efficace et cela à travers la définition des clauses de départ.

Nous le voyons maintenant, notre quête d'une intégration efficace des biais va nous mener plus loin qu'une définition d'un nouveau type d'opérateurs de raffinement. Nous allons décrire un nouveau cadre pour l'utilisation de l'algorithme générer-et-tester alliant efficacité du parcours et élagage sûr (dynamique ou non) de l'espace de recherche à l'aide de toutes les propriétés (privées ou non).

5.1 Motivations

Nous revenons à notre premier souci : rendre efficace l'algorithme générer-et-tester, cette fois-ci en élargissant cette préoccupation au-delà de l'opérateur.

Étant donné l'ensemble de propriétés $\mathcal{P}$ caractérisant la solution, quel est l'opérateur utilisant au mieux ces propriétés pour permettre la recherche la plus efficace possible ? Comment utiliser pratiquement ces propriétés et quelles sont les clauses à utiliser pour démarrer cette recherche ?

Commençons par revenir sur les deux grandes familles d'opérateurs utilisés en Programmation Logique Inductive et dont nous avons déjà parlé :

— les *opérateurs optimaux* définis dans [De Raedt et Bruynooghe, 1993] et à la définition 2.6 ;

— les *opérateurs idéaux* définis dans [van der Laag et Nienhuys-Cheng, 1994b] et à la définition 2.7 ; le chapitre précédent leur était consacré.

Avant de définir nos propres opérateurs, passons en revue ce que nous estimons être leurs défauts respectifs vis-à-vis de nos objectifs.

Dans le cas des opérateurs optimaux, ces défauts ne sont pas propres à la notion d'optimalité mais au choix des auteurs d'associer cette notion à leurs opérateurs particuliers (cf définition 2.5, page 38).

Ainsi, cette définition suppose l'existence dans $\mathcal{L}_h$ d'un élément $\top$, plus général. Cette existence d'un unique élément plus général n'est pas justifié : nous verrons des relations de généralité qui induisent plusieurs éléments maximalement généraux.

Plus grave encore, la recherche doit démarrer à partir de $\top$: l'opérateur est nécessairement descendant et la transposition en ascendant est délicate, en particulier parce que l'on dispose rarement d'élément maximalement spécifique dans l'espace de recherche.

Enfin, la complétude des opérateurs optimaux amène un autre défaut qui lui, est très lié à notre approche : l'optimalité pose que l'ensemble de $\mathcal{L}_h$ doit pouvoir être atteint ; or nous n'avons pas besoin d'atteindre les clauses qui ne satisfont pas nos propriétés.

Dans le cas des opérateurs idéaux, le défaut majeur est, à notre sens, leur notion de complétude : elle est trop liée à la relation utilisée pour ordonner l'espace de recherche. Explicitons à nouveau la notion de complétude des opérateurs idéaux :

> un opérateur $\mathcal{O}$ est dit complet si, pour toutes hypothèses H et H' de l'espace de recherche, le fait que H et H' soient comparables pour la relation de généralité utilisée implique que, soit $H \in \mathcal{O}^*(H')$, soit $H' \in \mathcal{O}^*(H)$.

Au premier chef, cette complétude est vérifiée par n'importe quel opérateur si l'on utilise une relation trop faible. Considérons le cas extrême où l'espace est ordonné par la relation identité : l'idéalité indique que tout opérateur est complet puisque pour tout couple d'hypothèses (H, H') en relation (ici, on a $H = H'$), H et H' peuvent être jointes par 0 application de l'opérateur ! Pourtant, nous n'avons aucune garantie sur les clauses atteignables par l'opérateur.

Ce défaut apparaît donc lorsque la relation devient faible ; un autre défaut survient lorsque la relation est plus importante : dans ce cas, il y a plusieurs chemins pour atteindre une même hypothèse et l'idéalité impose que tous ces chemins doivent

être utilisés. L'opérateur va ainsi, par des voies différentes, produire sans cesse les mêmes hypothèses. Les opérateurs idéaux semblent donc inefficaces.

En dernier lieu, si l'idéalité pose que n'importe quelles hypothèses comparables doivent pouvoir être reliées par applications successives de l'opérateur, elle n'indique pas où doit débuter la recherche.

Venons en aux relations de généralité que devront respecter nos opérateurs à savoir les relations naturelles. Les relations naturelles dont nous avons discuté jusqu'à présent faisaient intervenir au maximum deux propriétés, dont l'une était nécessairement la complétude par rapport aux exemples positifs A^+ et l'autre un biais de langage.

Nous allons maintenant relâcher ces contraintes. Une propriété élémentaire étant de la forme $f(H)\ \mathcal{R}\ k$ (comme posé à la section 3.1), nous allons considérer un ensemble fini $\mathcal{P}$ de propriétés élémentaires avec comme seule contrainte qu'au moins l'une de ces propriétés devrait concerner les exemples (COUVRE$_e$ ou ÉCARTE$_e$).

Un tel ensemble de propriétés dénotant une conjonction

$$\mathcal{P}(H) \Leftrightarrow P_1(H) \wedge \ldots \wedge P_n(H)$$

la relation naturelle de $\mathcal{P}$ est alors définie par

$$\succeq_{\mathcal{P}} = \succeq_{P_1} \cap \ldots \cap \succeq_{P_n}$$

d'après la proposition 4.4 (page 66).

On notera $\langle \mathcal{L}_h, \mathcal{P} \rangle$ l'espace de recherche $\mathcal{L}_h$ ordonné par la relation naturelle de $\mathcal{P}$, $\succeq_{\mathcal{P}}$. Enfin, $\mathcal{L}_h^{/\mathcal{P}}$ représentera l'ensemble des clauses de $\mathcal{L}_h$ qui vérifient la propriété $\mathcal{P}$.

L'idée que nous allons exploiter est qu'il faut éviter au maximum de considérer des hypothèses qui ne satisfont pas une propriété connue de la solution.

Cette idée n'est pas présente dans les opérateurs de raffinement classiques : leurs notions de complétude imposent de parcourir tout $\mathcal{L}_h$ indépendamment des propriétés connues de la solution.

Avec cette volonté d'utiliser intensivement les propriétés, nous allons rapidement nous heurter à de nouveaux problèmes : l'incompatiblité de certaines et l'appauvrissement de la relation naturelle qui l'accompagne.

En combinant les propriétés, on fait la conjonction de leurs relations naturelles. La relation utilisée pour ordonner l'espace de recherche s'appauvrit de plus en plus et l'on peut craindre d'obtenir une relation trop faible pour nous permettre de parcourir l'espace de recherche.

Exemple 5.1 (bornes sur la taille)
Ainsi, si l'on considère $\mathcal{P}$ composé des propriétés TAILLEMIN$_2$ *et* TAILLEMAX$_4$ *:*

$$\begin{array}{rcl} \text{TAILLEMIN}_2(H) & \Leftrightarrow & |H| \geq 2 \\ \text{TAILLEMAX}_4(H) & \Leftrightarrow & |H| \leq 4 \end{array}$$

la relation naturelle de leur conjonction est définie par

$$C \geq_{\mathcal{P}} D \quad \Leftrightarrow \quad |C| = |D|$$

ce qui signifie que l'on ne pourra passer d'une clause à l'autre que si elles sont de même taille.

Ce comportement semble tellement limité que nous ne sommes pas sûrs de vouloir l'adopter. Cela est d'autant plus grave que cette contradiction apparaît également sur des propriétés faisant intervenir des exemples.

Exemple 5.2
Cette fois-ci, on allie TAILLEMIN$_2$ *à* COUVRE$_{e^+}$ *:*

$$\begin{array}{rcl} \textsc{Couvre}_{e^+}(H) & \Leftrightarrow & H \models e^+ \\ \textsc{TailleMin}_2(H) & \Leftrightarrow & |H| \geq 2 \end{array}$$

la relation naturelle de leur conjonction $\mathcal{P}$ est définie par

$$C \geq_{\mathcal{P}} D \quad \Leftrightarrow \quad (C \models D) \wedge (|C| \geq |D|)$$

$C \models D$ indique que la recherche ira en spécialisant en ajoutant donc des littéraux entre autre. À l'inverse, $|C| \geq |D|$ contraint la taille des clauses à diminuer au cours de la recherche.

Ces exemples suggèrent qu'une relation naturelle peut devenir si pauvre qu'elle ne nous permettra plus d'atteindre toutes les clauses de l'espace de recherche. Par suite, la première question à laquelle nous devons répondre est : étant donné un ensemble de propriétés caractérisant la solution, *peut-il exister* un opérateur utilisant au mieux ces propriétés pour permettre une recherche efficace ? Nous avons d'ores et déjà l'intuition que la réponse passera par une définition pertinente de la notion de *compatibilité* entre propriétés, elle-même liée à une idée de *complétude* par rapport au sous-ensemble de $\mathcal{L}_h$ dont les hypothèses satisfont $\mathcal{P}$: grossièrement, nous dirons que des propriétés sont incompatibles si leur relation naturelle ne permet pas de parcourir l'espace de recherche. Cependant, cette notion d'incompatibilité ne sera pas absolue : elle sera relative à l'ensemble des hypothèses qui servent de points de départ à la recherche.

5.2 Gestion des propriétés

5.2.1 Parcourabilité et compatibilité

Nous devons définir notre propre notion de complétude. Cela est possible à partir de deux observations.

— Tout d'abord, dans l'espace de recherche $\mathcal{L}_h$, nous ne tenons pas à conserver toutes les clauses : seules celles qui vérifient les propriétés désirées doivent impérativement être accessibles. Autrement dit, peu nous importe d'ignorer certaines hypothèses de $\mathcal{L}_h$, seules celles de $\mathcal{L}_h^{/\mathcal{P}}$ doivent impérativement être accessibles.

— Ensuite, un point important est d'avoir des points de départ pour lancer la recherche du générer-et-tester. Dans l'approche descendante des opérateurs optimaux, la clause vide jouait ce rôle mais, en acceptant de perdre des clauses de $\mathcal{L}_h$, il n'est plus certain que nous disposions encore de cette unique hypothèse plus générale.

Ces deux notions sont intimement liées et c'est pourquoi nous les définirons l'une par rapport à l'autre.

Définition 5.1 (parcourabilité et bases)
$\langle \mathcal{L}_h, \mathcal{P} \rangle$ *est dit* parcourable *s'il existe* $\mathcal{B}$*, un sous-ensemble de* $\mathcal{L}_h^{/\mathcal{P}}$*, qui soit calculable et complet, c'est-à-dire*

$$\forall A \in \mathcal{L}_h^{/\mathcal{P}}, \exists B \in \mathcal{B} : B \succeq_{\mathcal{P}} A \ .$$

Nous dirons que $\mathcal{B}$ *est une* base *de* $\langle \mathcal{L}_h, \mathcal{P} \rangle$.

Intuitivement, nous voulons capturer dans la base, les hypothèses les plus générales de $\mathcal{L}_h^{\mathcal{P}}$ pour $\succeq_{\mathcal{P}}$ et nous en prenons autant qu'il faut pour que toute hypothèse de $\mathcal{L}_h^{\mathcal{P}}$ soit en relation par $\succeq_{\mathcal{P}}$ avec une des hypothèses de la base.

Exemple 5.3
Si l'on veut que la solution ne dépasse pas une certaine taille et qu'elle couvre un exemple positif :

$$\begin{array}{rcl} |H| & \leq & n \\ H & \models & e^+ \ . \end{array}$$

L'espace est ordonné par la relation naturelle qui nous a permis de découvrir un opérateur idéal au chapitre précédent :

$$(C \models D) \wedge (|C| \leq |D|)$$

Cet espace est parcourable à partir de la base $\mathcal{B} = \{\top\}$*. En effet, toutes les clauses satisfaisant les propriétés sont effectivement plus grandes en taille et plus spécifiques au sens de la* θ*-subsomption que* $\top$*.*

C'est la solution classique. Cependant, nous pourrions tout aussi bien utiliser une base $\mathcal{B}'$ *contenant les clauses maximalement générales pour chaque taille autorisée (c'est-à-dire entre 0 et* n*). Nous verrons un peu plus loin que cet ensemble est calculable.*

La parcourabilité va nous permettre de cerner la notion de compatibilité entre propriétés, mais auparavant nous devons discuter le fait que plusieurs bases soient possibles pour un ensemble de propriétés donné. Cela signifie simplement qu'il y a parfois plusieurs façons de construire une base, et cela est directement lié à la manière dont on veut utiliser chaque propriété en cours de recherche.

Dans l'exemple précédent, l'unique clause qui compose $\mathcal{B}$ est un élément maximal pour les deux relations naturelles (c'est la clause la plus générale de l'espace de recherche et aussi la plus petite). La recherche se fera à la fois en spécialisant et en augmentant la taille des clauses. Il y aura donc un risque de trouver des hypothèses trop spécifiques ou de taille trop importante : l'élagage devra se produire par rapport à ces deux propriétés. Les deux propriétés sont dans ce cas dites *dynamiques*.

Dans $\mathcal{B}'$, nous avons au moins un représentant par taille et ces représentants sont des éléments maximaux par rapport à l'ordre de généralité. La recherche se fera toujours en spécialisant mais sans faire varier la taille et, par conséquent, toutes les clauses rencontrées vérifieront la contrainte sur la taille. Cette propriété est cette fois-ci dite *statique*.

Ces deux possibilités pour l'utilisation d'une propriété, *dynamique* si la propriété nécessite un élagage dynamique, *statique* si toutes les hypothèses rencontrées vérifient la propriété, vont nous permettre de caractériser la compatibilité entre deux propriétés.

Définition 5.2 (compatibilités)
Soient P_1 et P_2 deux propriétés. Considérons les ensembles de propriétés $\mathcal{P}$ contenant P_1 et P_2.

- *Si, quel que soit $\mathcal{P}$, $\langle \mathcal{L}_h, \mathcal{P} \rangle$ n'est pas parcourable alors P_1 et P_2 sont dites* incompatibles.
- *Sinon, si pour les $\mathcal{P}$ tels que $\langle \mathcal{L}_h, \mathcal{P} \rangle$ est parcourable, on ne peut pas construire de base pour laquelle P_1 et P_2 soient dynamiques à la fois, ces propriétés sont dites* faiblement compatibles.
- *Sinon, c'est-à-dire s'il existe $\mathcal{P}$ et $\mathcal{B}$ pour lesquels P_1 et P_2 sont dynamiques, ces propriétés sont dites* fortement compatibles.

Exemple 5.4
Considérons les propriétés TailleMin$_n$ *et* Écarte$_{e^-}$ *(ce sont les duales de l'exemple précédent) :*

$$\begin{array}{rcl} |H| & \geq & l_{min} \\ H & \not\models & e^- \ . \end{array}$$

L'espace est ordonné par la relation $(|C| \geq |D|) \wedge (D \models C)$. Elles ne sont pas fortement compatibles car on ne peut pas trouver d'élément maximal pour la relation naturelle, c'est-à-dire un élément qui soit à la fois plus spécifique et de plus grande taille. En revanche, elles sont faiblement compatibles ; en effet, si nous ajoutons une borne maximale sur la taille et si la profondeur est elle aussi bornée, le calcul des clauses les plus spécifiques devient possible.

L'ensemble des propriétés $\mathcal{P}$ à satisfaire peut être partitionné comme suit :

- $\mathcal{P}_d$, l'ensemble des propriétés que l'on veut dynamiques ;
- $\mathcal{P}_s$, celles que l'on veut statiques ;
- $\mathcal{P}_a$, les propriétés actives :

$$\mathcal{P}_a = \mathcal{P}_d \cup \mathcal{P}_s$$

- $\mathcal{P}_i$, les inactives :

$$\mathcal{P}_i = \mathcal{P} - \mathcal{P}_a$$

À partir de maintenant, une entrée de l'algorithme générer-et-tester est constituée par $\langle \mathcal{L}_h, \mathcal{B}, \mathcal{P}_d, \mathcal{P}_s, \mathcal{P}_i \rangle$ où $\mathcal{B}$ est caractérisée de la manière suivante :

> Pour chaque classe d'équivalence induite par les propriétés statiques $\mathcal{P}_s$, une base doit contenir les éléments maximaux de cette classe par rapport à la relation naturelle des propriétés dynamiques $\geq_{\mathcal{P}_d}$.

Les classes d'équivalences sont définies comme suit. L'ensemble des propriétés statiques $\mathcal{P}_s$ étant constitué de propriétés de la forme

$$f_i(H)\, \mathcal{R}_i\, k_i$$

la classe d'équivalence d'une hypothèse H est définie comme l'ensemble des hypothèses H' telles que, pout tout i, les valeurs de $f_i(H)$ et $f_i(H')$ sont équivalentes pour $\mathcal{R}_i$.

Les propriétés Couvre_{e^+} et Écarte_{e^-} sont incompatibles. Cela n'a rien d'étonnant : le contraire aurait signifié que l'on est capable de résoudre le problème d'apprentissage sans recherche effective, simplement par calcul de la base. Dans la suite, nous admettrons donc que les propriétés de couverture des exemples positifs sont dans $\mathcal{P}_d$ tandis que la non-couverture des négatifs est dans $\mathcal{P}_i$, ou l'inverse. En particulier, les compatibilités dont nous allons discuter n'ont de sens qu'en présence de l'une de ces propriétés, Couvre_e et Écarte_e.

Nous avons vu, à l'exemple 5.1, que les propriétés duales semblaient de bonnes candidates à l'incompatibilité. Considérons, pour simplifier le propos, des propriétés duales à valeurs entières :

$$k_1 \leq f(H) \leq k_2$$

Elles ne peuvent pas être fortement compatibles ; sont elles faiblement compatibles ? D'après la méthodologie introduite ci-dessus, nous devons trouver des éléments maximaux pour l'ordre de généralité (soit les hypothèses les plus spécifiques, soit les plus générales), et cela pour chaque valeur de f comprise entre k_1 et k_2. La question est finalement :

> f étant fixé, l'ensemble des éléments maximaux pour l'ordre de généralité est-il calculable ou non ?

Si oui, au besoin en imposant d'autres propriétés, alors les propriétés duales considérées seront faiblement compatibles. Les tables 5.1 et 5.2 montrent que c'est bien le cas : toutes les propriétés duales sont toutes faiblement compatibles.

Finalement, toutes les propriétés sont faiblement compatibles sauf celles liées aux exemples. La distinction traditionnelle entre approches descendantes et ascendantes était justifiée par l'incompatibilité entre correction et complétude. Cependant, cette séparation n'a pas de sens pour nos autres propriétés.

5.2.2 Calcul des bases de départ

Nous allons montrer comment calculer des bases pour le problème `grand-père`, avec quelques propriétés simples. Commençons par une approche descendante : considèrons les propriétés TailleMin_1 et TailleMax_2 alliées à la couverture des

Table 5.1 – Compatibilité faible de propriétés duales en présence de Couvre_e

Propriétés duales	Propriétés nécessaires pour la compatibilité faible
TailleMin_n et $\text{TailleMax}_{n'}$	$\{-\}$
NbVarsMin_n et $\text{NbVarsMax}_{n'}$	$\{-\}$
ProfMin_n et $\text{ProfMax}_{n'}$	$\{-\}$

TABLE 5.2 – Compatibilité faible de propriétés duales en présence de ÉCARTE$_e$

Propriétés duales	Propriétés nécessaires pour la compatibilité faible
TAILLEMIN$_n$ et TAILLEMAX$_{n'}$	{PROFMAX$_k$}
NBVARSMIN$_n$ et NBVARSMAX$_{n'}$	{PROFMAX$_k$}
PROFMIN$_n$ et PROFMAX$_{n'}$	{TAILLEMIN$_k$; TAILLEMAX$_{k'}$}

exemples de A^+ :

$$\begin{array}{rcl} P_1(H) & \Leftrightarrow & |H| \geq 1 \\ P_2(H) & \Leftrightarrow & |H| \leq 2 \\ P_{i+2}(H) & \Leftrightarrow & H \models e_i^+ \end{array}$$

Pour chaque taille possible de clauses, nous construisons les clauses les plus générales. Par exemple, pour les clauses de taille 2, nous prenons toutes les combinaisons de taille 2 utilisant des symboles de prédicats du problème avec comme arguments que des variables différentes deux à deux. Nous obtenons les clauses suivantes :

grand-père(A,B) ← père(C,D).
grand-père(A,B) ← mère(C,D).
grand-père(A,B) ← parent(C,D).
grand-père(A,B) ← parent(C,D),parent(E,F).
grand-père(A,B) ← père(C,D),père(E,F).
grand-père(A,B) ← mère(C,D),mère(E,F).
grand-père(A,B) ← père(C,D),parent(E,F).
grand-père(A,B) ← père(C,D),mère(E,F).
grand-père(A,B) ← parent(C,D),mère(E,F).

Ensuite, nous ne conservons que les clauses couvrant les exemples positifs : ici, il suffit d'écarter les clauses faisant apparaître le prédicat « mère » (car celui-ci n'intervient pas dans tous les exemples positifs).

Finalement, la base suivante permet bien de parcourir l'espace de recherche du problème *grand-père* (avec un opérateur qui ne s'autorise qu'à unifier des variables).

grand-père(A,B) ← père(C,D).
grand-père(A,B) ← parent(C,D).
grand-père(A,B) ← parent(C,D),parent(E,F).
grand-père(A,B) ← père(C,D),père(E,F).
grand-père(A,B) ← père(C,D),parent(E,F).

Évoquons, pour terminer, les approches ascendantes. Dans notre formalisme, cela signifie que des propriétés du type ÉCARTE$_e$ sont utilisées en dynamique. La base doit donc contenir les clauses les plus spécifiques ne couvrant pas les e^-. En toute généralité, cet ensemble est infini et donc incalculable.

Une solution classique consiste à démarrer la recherche à partir d'un exemple positif, ou d'un exemple positif modifié [Rouveirol, 1992, Muggleton, 1995]. Cela

n'est, à notre sens, pas satisfaisant : la couverture des exemples positifs fait partie des propriétés inactives et n'a donc pas à intervenir dans la recherche. Cela dit, il faut bien convenir que cette façon de faire ne fait pas courir le risque de perdre une solution. Cependant, nous avons pu constater au cours de nos expérimentations que cela était particulièrement inefficace (section 5.4) ce qui, cette fois-ci, est un inconvénient majeur.

Pour notre part, nous conservons donc l'idée de calculer les clauses les plus spécifiques. En bornant taille et profondeur, il devient possible de calculer les clauses les plus spécifiques ne couvrant pas les exemples négatifs. Dans le cas du problème *grand-père*, nous avons déjà fixé la taille des clauses à 1 ou 2. De plus, la profondeur est bornée dans le sens où ce problème ne dispose pas de symboles fonctionnels. Nous allons construire les mêmes combinaisons de symboles de prédicat que dans le cas descendant mais cette fois-ci les arguments ne seront plus des variables distinctes mais une même variable. Nous trouvons cette fois-ci :

grand-père(A,A) ← père(A,A).
grand-père(A,A) ← mère(A,A).
grand-père(A,A) ← parent(A,A).
grand-père(A,A) ← parent(A,A),parent(A,A).
grand-père(A,A) ← père(A,A),père(A,A).
grand-père(A,A) ← mère(A,A),mère(A,A).
grand-père(A,A) ← père(A,A),parent(A,A).
grand-père(A,A) ← père(A,A),mère(A,A).
grand-père(A,A) ← parent(A,A),mère(A,A).

Nous ne conservons ensuite que celles qui ne couvrent pas d'exemples négatifs : a priori, toutes les clauses sont cette fois-ci conservées. Il restera à parcourir l'espace de recherche à partir de cette base par *désunification* des variables.

5.2.3 Dynamiques vs Statiques

N'avoir que des propriétés dynamiques pose le problème de calcul de la base : si nous n'utilisons pas explicitement les bornes fixées par certaines propriétés, cette base ne sera pas calculable. N'avoir que des propriétés statiques signifie que la recherche n'est pas nécessaire : les éléments de la base sont des solutions. La solution semble donc être un compromis entre ces deux extrêmes, les combinaisons pertinentes devant être identifiées empiriquement.

Les dynamiques devront être testées après chaque application de l'opérateur. Les statiques n'ont pas besoin d'être testées mais doivent être « codées » dans la base : nous aurons un représentant par classe d'équivalence, le nombre de ces classes augmentant avec le nombre de propriétés statiques.

5.3 Opérateurs parfaits

5.3.1 Définition de la perfection

Munis des notions de base et de parcourabilité, nous pouvons maintenant donner notre définition de complétude.

Définition 5.3 (opérateur complet)
Un opérateur $\mathcal{O}$ est dit complet *pour un espace $\langle \mathcal{L}_h, \mathcal{P} \rangle$ et une base $\mathcal{B}$, si l'on a :*

$$\mathcal{L}_h^{/\mathcal{P}} \subseteq \bigcup_{B \in \mathcal{B}} \mathcal{O}^*(B)$$

Notez que l'on a demandé l'inclusion de $\mathcal{L}_h^{/\mathcal{P}}$ dans $\bigcup_{B \in \mathcal{B}} \mathcal{O}^*(B)$ et non pas l'égalité, car l'opérateur peut générer des clauses ne satisfaisant pas certaines des propriétés dynamiques $\mathcal{P}_d$ (c'est le propre des propriétés dynamiques !).

En revanche, on veut qu'absolument toutes les clauses générées par l'opérateur satisfassent les propriétés statiques $\mathcal{P}_s$.

Définition 5.4 (opérateur clos)
Étant donnée une base $\mathcal{B}$, un opérateur $\mathcal{O}$ est clos *par rapport aux propriétés statiques $\mathcal{P}_s$ si et seulement si*

$$\forall A \in \mathcal{L}_h, \forall B \in \mathcal{B} : A \in \mathcal{O}^*(B) \Rightarrow \mathcal{P}_s(A)$$

Comme pour les opérateurs optimaux, nous allons demander ensuite que nos opérateurs soient tels que le graphe de raffinement est un arbre, ou plutôt une forêt puisque nos bases peuvent avoir plus d'un élément. Cela nous amène à distinguer deux cas.

— Soit les arbres de la forêt sont disjoints et l'on parle alors d'*optimalité forte*. Cette notion coïncide alors avec celle de [De Raedt et Bruynooghe, 1993] (définition 2.6).
— Soit les arbres ne sont pas disjoints : on parlera d'*optimalité faible*.

Cette notion, dans ses deux versions, est illustrée à la Figure 5.1.

Finalement, nous pouvons définir nos propres opérateurs de raffinement : les opérateurs *parfaits*.

Définition 5.5 (perfection)
Un opérateur est dit parfait *pour un problème défini par $\langle \mathcal{L}_h, \mathcal{B}, \mathcal{P}_d, \mathcal{P}_s, \mathcal{P}_i \rangle$ si :*
— *il est calculable ;*
— *il respecte $\succeq_{\mathcal{P}_d}$, la relation naturelle des propriétés dynamiques, et il est strict par rapport à cette relation ;*
— *il est complet à partir de la base $\mathcal{B}$ pour $\langle \mathcal{L}_h, \mathcal{P}_d \cup \mathcal{P}_s \rangle$;*
— *il est clos à partir de la base $\mathcal{B}$ pour $\langle \mathcal{L}_h, \mathcal{P}_s \rangle$;*
— *il est fortement optimal pour $\mathcal{L}_h$.*

Les différents points de cette définition sont illustrés à la Figure 5.2.

5.3.2 Obtention d'un opérateur parfait

Il est possible d'obtenir un opérateur parfait en fixant un ordre sur les transformations de base que nous nous autorisons, le plus souvent en ordre un, l'ajout de littéraux, l'unification de variables et la substitution. Explicitons cette démarche sur le problème `grand-père` à partir de la base de la section 5.2.2 : l'opérateur doit spécialiser cette base, sans faire varier la taille (en unifiant des variables par exemple).

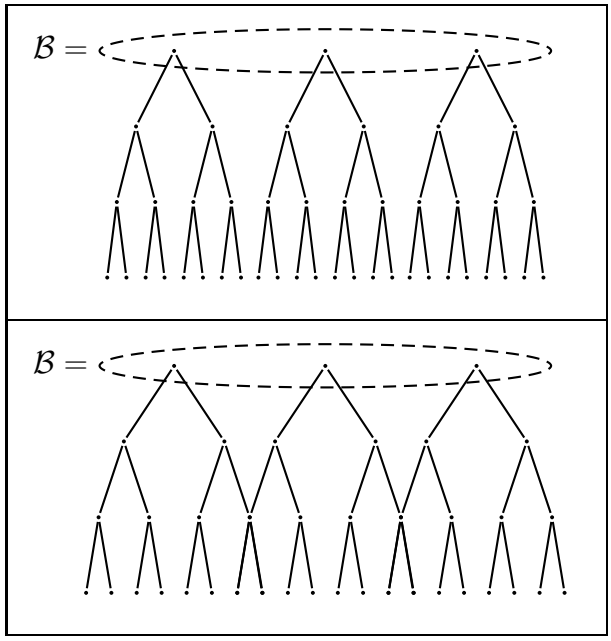


FIGURE 5.1 – Optimalités forte et faible.

De plus, ce problème ne dispose pas de symboles fonctionnels, si bien que l'opérateur ne peut qu'effectuer des unifications de variables.

Pratiquement, on associe à chaque clause C, la liste des unifications de deux variables qu'il est possible de lui appliquer. Si v est le nombre de variables de C, nous avons n unifications possibles avec $n = C_v^2$. L'opérateur utilise cette donnée mais doit aussi la mettre à jour. L'opérateur appliqué à la clause C pour laquelle les unifications possibles sont $\{u_1, \ldots, u_i, \ldots, u_n\}$ fournit n nouvelles clauses $\{D_1, \ldots, D_i, \ldots, D_n\}$, chaque D_i étant obtenue par application de u_i sur C (si cette unification ne réduit pas la taille), et l'ensemble des unifications applicables à D_i est $\{u_{i+1}, \ldots, u_n\}$.

Cet opérateur est optimal et complet. L'intuition de la preuve est la suivante. Pour une clause A de taille autorisée, on considère la clause B de la base de même taille et faisant intervenir les mêmes symboles de prédicat que A. On a alors B qui θ-subsume A : il existe θ telle que $B\theta = A$ et θ ne contient que des unifications applicables à B. Il ne reste qu'à les appliquer dans le bon ordre (cet ordre est unique) pour obtenir A.

5.4 Expérimentations

Nous décrivons maintenant l'application de nos méthodes au problème des trains. Ce problème a été introduit par Michalski puis repris pour une compétition entre programmes d'apprentissage [Michie et al., 1994]. Nous ne donnons ici qu'une rapide description de ce problème.

— Il s'agit de distinguer les trains qui vont vers l'est de ceux qui vont vers l'ouest.
— Un train peut avoir jusqu'à quatre wagons. Les différentes caractéristiques de chaque wagon (ainsi que leurs valeurs possibles) sont les suivantes : la forme

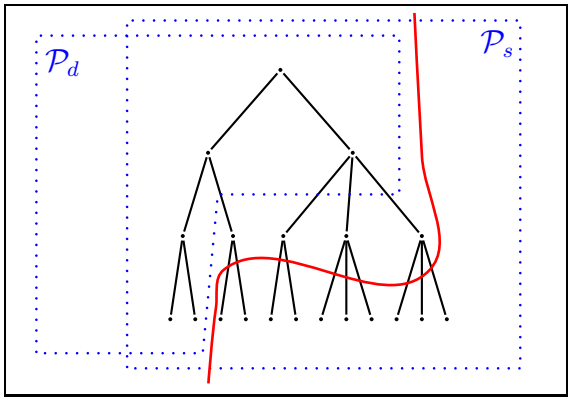


FIGURE 5.2 – Rôles des différents types de propriétés. L'opérateur reste dans l'espace défini par les propriétés statiques $\mathcal{P}_s$. Il peut en revanche rencontrer des propriétés ne satisfaisant pas les propriétés dynamiques de $\mathcal{P}_d$ mais, dans ce cas, l'élagage dynamique est possible. $\mathcal{P}_i$, l'ensemble des propriétés inactives, ne peut pas être utilisé en cours de recherche.

(5), la taille (2), les parois (2), le toit (5), le nombre de roues (2), la forme des charges (6) et le nombre de charges (4).

— Ce langage autorise 4800 wagons différents et donc 2.10^{14} trains possibles. Il ne s'agit pas là de la taille de notre espace de recherche mais du nombre de clauses de cet espace qui ne contiennent aucune variable et qui donnent une valeur à chaque attribut cité ci-dessus.

— Nous avons travaillé avec une base de 100 trains. La définition à découvrir partitionne à chaque fois la base d'exemples en environ 50 positifs et 50 négatifs. Ces définitions sont fixées à l'avance (notre propos est d'illustrer simplement l'efficacité de nos méthodes, pas de résoudre un problème réel).

Commençons en cherchant la définition suivante pour les trains allant vers l'est :

$$\begin{aligned} \text{eastbound}(A) \leftarrow \quad & \text{member}(B, A), \\ & \text{load}(B, \text{l\_triangle}) \end{aligned}$$

Nous posons a priori que la taille des clauses ne doit pas dépasser quatre littéraux, que ces clauses ne doivent pas posséder plus de deux variables existentielles, et enfin qu'elles doivent être connectées. Toutes ces propriétés vont être utilisées en dynamique. On part de la clause

$$\text{eastbound}(A) \leftarrow$$

et l'opérateur parfait ajoute des littéraux, unifie des variables ou substitue des termes aux variables.

Les résultats sont encourageants :

Hypothèses générées :	1 127
Hypothèses élaguées :	658
Temps CPU : 70 secondes.	

Avec la même configuration, on cherche à apprendre la définition disjonctive suivante :

$$\begin{aligned}
\text{eastbound}(A) \leftarrow\ & \text{member}(B, A), \text{position}(B, \text{p_three}),\\
& \text{load}(B, \text{l_triangle})\\
\\
\text{eastbound}(A) \leftarrow\ & \text{member}(B, A), \text{position}(B, \text{p_three}),\\
& \text{load}(B, \text{l_hexagon})\\
\\
\text{eastbound}(A) \leftarrow\ & \text{member}(B, A), \text{position}(B, \text{p_three}),\\
& \text{not_double}(B), \text{load}(B, \text{l_circle})
\end{aligned}$$

La différence par rapport au cas précédent est qu'une hypothèse n'est pas contrainte de couvrir tous les exemples positifs : même si elle n'en couvre plus qu'un, nous devons toujours considérer qu'elle peut faire partie de la solution. Cela affaiblit le pouvoir élagant de la couverture et, effectivement, les résultats sont cette fois-ci catastrophiques.

Hypothèses générées :	1 799 173
Hypothèses élaguées :	796 814
Temps CPU : 7 heures 30 minutes.	

La conclusion est simple : une propriété qui porte sur la disjonction et non pas sur chaque hypothèse de la disjonction ne doit pas être utilisée en dynamique. Nous allons donc privilégier les propriétés liées aux exemples négatifs, plutôt qu'aux positifs ; en effet, aucune clause de la disjonction ne doit couvrir un exemple négatif. On place donc la couverture des exemples positifs dans les propriétés inactives et l'on utilise en dynamique le rejet des négatifs.

Comme nous l'avons déjà évoqué, le problème du calcul de la base dans les approches ascendantes est souvent résolu en démarrant la recherche avec un exemple positif. Ici, un exemple (un train) est décrit par une quarantaine de littéraux et nous pouvons légitimement penser qu'une définition suffisamment générale pour couvrir tous les exemples positifs contient beaucoup moins de littéraux. Or, l'espace à franchir pour passer des exemples positifs à la « zone » des hypothèses ne contenant que quelques littéraux est intraitable. À titre indicatif, le tableau suivant donne, pour chaque taille, le nombre d'hypothèses possibles.

1	2	3	4	5	...	30	...
10^1	10^1	10^2	10^4	10^6	...	?!?	...

Nous démarrons la recherche avec les clauses les plus spécifiques ne couvrant pas les négatifs, de taille comprise entre 0 et 4. Les résultats sont alors les suivants.

Hypothèses générées :	3 988
Hypothèses élaguées :	3 988
Temps CPU : 406 secondes ($<$ 7 minutes).	

Il apparaît que toutes les clauses générées ont été élaguées. Cela signifie que les solutions étaient présentes dans la base de départ. Cela est lié au problème et non

pas à notre approche : les définitions de train sont nécessairement maximalement spécifiques. Par exemple, considérons le prédicat `position` dont le deuxième argument indique la position du wagon dans le train : si cet argument n'est pas une constante, l'information amenée par le littéral ainsi formé est absolument nulle et il n'a donc pas de raison d'apparaître dans une définition.

5.5 Bilan

Nous avons défini, dans ce chapitre, les notions de parcourabilité et de complétude : il s'agit de pouvoir atteindre toutes les clauses de l'espace de recherche vérifiant les propriétés, et cela à partir d'une base regroupant les points de départ de la recherche.

Nous avons vu que les propriétés possédaient deux moyens d'action : dynamique ou statique selon qu'elles vont permettre d'élaguer l'espace de recherche pendant le parcours ou qu'elles sont codées dans la base.

Finalement, à l'aide d'une reformulation de l'optimalité [De Raedt et Bruynooghe, 1993] dans notre cadre de travail, nous avons abouti à la définition de nos propres opérateurs, les *opérateurs parfaits*.

Au cours de cette quête d'un opérateur parfait, nous avons montré que toutes les propriétés étaient compatibles dans le sens où elles peuvent toutes participer efficacement à la recherche. En vérité, il faut faire exception des propriétés liées aux exemples : couverture des positifs et rejet des négatifs sont bien incompatibles. Cela justifie la distinction traditionnelle entre approches descendante et ascendante. Cela dit, nous avons montré que cette distinction n'a pas de sens pour les autres propriétés car quelles qu'elles soient, nous pouvons les rendre compatibles. Pour cela, il peut être nécessaire d'utiliser d'autres propriétés.

Cela constitue le point faible de notre méthode : comment justifier l'utilisation de propriétés que l'expert n'a pas désignées comme pertinentes pour son problème ? Il y a deux réponses.

— Premièrement, ces propriétés peuvent être contenues implicitement dans les connaissances fournies par l'utilisateur : ainsi, si l'on a besoin d'une borne pour la profondeur, on peut utiliser la profondeur maximale des exemples positifs.

— Deuxièmement, on peut utiliser de nouvelles propriétés au cours d'un *iterative deepening* à la manière des langages imbriqués de CLINT [De Raedt, 1992] : on effectue une recherche avec des propriétés très fortes (profondeur et taille nulle, pas de variables existentielles, etc.) ; puis, si la recherche n'aboutit pas, on relâche progressivement ces propriétés. Ainsi, nous pouvons nous donner des propriétés sans prendre le risque de perdre une solution et nos méthodes sont applicables indépendamment des propriétés fournies par l'utilisateur.

L'exemple décrit à la section 5.2.2 montre qu'en utilisant des propriétés statiques, la base va regrouper des « patrons » de la solution, c'est-à-dire des clauses où seuls les arguments des littéraux restent à déterminer. Cela n'est pas sans rappeler les formalismes que nous avons décrits à la section 3.3 : les *clause sets* de [Bergadano et Gunetti, 1995], les schémas de [Kietz et Wrobel, 1992] ou encore les grammaires

de [Cohen, 1994]. La différence essentielle est que nous ne demandons pas à l'utilisateur de fournir de tels patrons, ils sont induits par les propriétés à satisfaire et notre méthode permet de les exploiter explicitement. Nous estimons qu'il est plus naturel pour un expert d'exprimer les propriétés des solutions que de fournir une représentation de l'espace de recherche dans un formalisme qui n'a pas de sens dans son domaine.

Au cours de nos expérimentations, nous avons pu constater que l'algorithme générer-et-tester était mis à mal par l'apprentissage disjonctif. Dans le cas des trains, nous avons dépassé cette difficulté à travers une approche ascendante mais nous avons montré que cette victoire était imputable aux particularités du problème. Dans l'absolu, l'approche ascendante n'est pas plus efficace que la descendante, même dans le cas disjonctif. Cela parce que les négatifs élaguent moins que les positifs : intuitivement, il y a beaucoup moins d'hypothèses qui couvrent un exemple que d'hypothèses qui le rejettent.

Dans la première partie de cette thèse, nous avons amélioré l'algorithme générer-et-tester et obtenu des résultats satisfaisants. Cependant, ces gains d'efficacité, appréciables en conjonctif, sont dérisoires lorsque la solution est disjonctive et on voit mal comment le générer-et-tester pourrait appréhender une telle difficulté. La seconde partie de cette thèse est justement consacrée à l'étude de l'apprentissage disjonctif, indépendamment de la Programmation Logique Inductive et de l'algorithme générer-et-tester.

Bibliographie

[Bergadano et Gunetti, 1995] Bergadano, F. et Gunetti, D. (1995). Learning clauses by tracing derivations. In *ILP : from ML to Software Engineering*. MIT Press.

[Cohen, 1994] Cohen, W. W. (1994). Grammatically biased learning : Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68 :303–366.

[De Raedt, 1992] De Raedt, L. (1992). *Interactive Theory Revision : An Inductive Logic Programming Approach*. Academic Press.

[De Raedt et Bruynooghe, 1993] De Raedt, L. et Bruynooghe, M. (1993). A theory of clausal discovery. In Bajcsy, R., éditeur, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann.

[Kietz et Wrobel, 1992] Kietz, J.-U. et Wrobel, S. (1992). Controlling the complexity of learning in logic through syntactic and task-oriented models. In Muggleton, S., éditeur, *Inductive Logic Programming*, pages 335–359. Academic Press.

[Michie et al., 1994] Michie, D., Muggleton, S., Page, D., et Srinivasan, A. (1994). To the international computing community : A new East-West challenge. Rapport interne, Oxford University Computing laboratory, Oxford,UK.

[Muggleton, 1995] Muggleton, S. (1995). Inverse entailment and PROGOL. *New Generation Computing Journal*, 13 :245–286.

[Rouveirol, 1992] Rouveirol, C. (1992). Extensions of inversion resolution applied to theory completion. In Muggleton, S., éditeur, *Inductive Logic Programming*, chapitre 3. Academic Press, London.

[van der Laag et Nienhuys-Cheng, 1994] van der Laag, P. R. J. et Nienhuys-Cheng, S. (1994). Existence and nonexistence of complete refinement operators. In Bergadano, F. et de Raedt, L., éditeurs, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag.

CHAPITRE 6

Introduction à l'apprentissage disjonctif

Ce chapitre présente l'apprentissage disjonctif, problème pour lequel nous proposerons deux solutions aux chapitres suivants.

Nous décrirons le problème et passerons en revue deux grandes familles de méthodes permettant d'apprendre des définitions disjonctives :

— les approches *par couverture* dont l'algorithme générer-et-tester fait partie ;
— les approches *diviser-pour-régner* qui regroupent, entre autres, les méthodes construisant des arbres de décision ;

Nous discuterons les faiblesses de ces méthodes, maintenant bien identifiées par la communauté d'apprentissage automatique, puis, nous présenterons un nouveau type d'apprentissage baptisé *approche globale*.

Cette dernière approche mettra en évidence la problématique commune de l'apprentissage supervisé disjonctif et de l'apprentissage non supervisé et nous en proposerons une présentation unifiée.

6.1 Le problème

Dans la première partie de cette thèse, nous nous sommes attachés à améliorer la recherche d'une solution conjonctive. Nous nous intéressons maintenant aux problèmes qui n'admettent pas de solution conjonctive, et pour lesquels il faut donc recourir à une solution disjonctive.

Un problème n'est pas intrinsèquement disjonctif. Comme le montre la condition sur la correction du moindre généralisé (corollaire 1.1, page 8), cela dépend du langage $\mathcal{L}_h$ choisi pour représenter les hypothèses du problème. Pour tout problème, il existe un langage $\mathcal{L}_h$ qui contient une caractérisation conjonctive du concept-cible.

Le choix consiste alors

— soit à construire de nouveaux attributs, ou de nouveaux prédicats selon le langage, bref à modifier le langage $\mathcal{L}_h$ pour qu'il contienne une solution conjonctive [Utgoff, 1986] ;

— soit se résoudre à chercher une solution disjonctive.

Pour notre part, nous allons nous consacrer à cette dernière possibilité, en commençant par justifier ce choix sur le problème du morpion.

6.1.1 Langage des hypothèses et morpion

Nous avons déjà évoqué ce qu'est un problème disjonctif à la section 1.4.2 (page 9), voyons maintenant un exemple de problème disjonctif. Il s'agit d'un problème disponible sur la base de l'UCI [Merz et Murphy, 1996] sous le nom de *tic-tac-toe endgame*, autrement dit, la *fin de partie au jeu du morpion*.

Il s'agit en effet de caractériser les situations de victoire dans ce jeu. Dans ce but, nous disposons de toutes les fins de partie voyant la victoire des croix comme exemples positifs, toutes les autres fins de partie constituant les exemples négatifs.

Ce problème présente un intérêt si l'on ne connaît pas le concept de ligne. Dans ce cas, un exemple est simplement le contenu des neuf cases du jeu que nous repérons comme suit :

A	B	C
D	E	F
G	H	I

Chacune de ces cases peut avoir comme valeur x pour une croix, o pour un rond et b si la case est vide. Ainsi, les exemples positif et négatif de la Figure 6.1 seront respectivement codés par

$$A = x \ \wedge \ B = b \ \wedge \ C = o \ \wedge \ D = x \ \wedge \ E = x \ \wedge \ F = b \ \wedge \ G = x \ \wedge \ H = o \ \wedge \ I = o$$

et

$$A = x \ \wedge \ B = x \ \wedge \ C = o \ \wedge \ D = o \ \wedge \ E = x \ \wedge \ F = x \ \wedge \ G = x \ \wedge \ H = o \ \wedge \ I = o$$

Pour une meilleure compréhension, nous n'utiliserons dans la suite que la représentation habituelle du jeu. Sous cette forme, la solution attendue est montrée à la Figure 6.2.

FIGURE 6.1 – Exemples positif et négatif du morpion

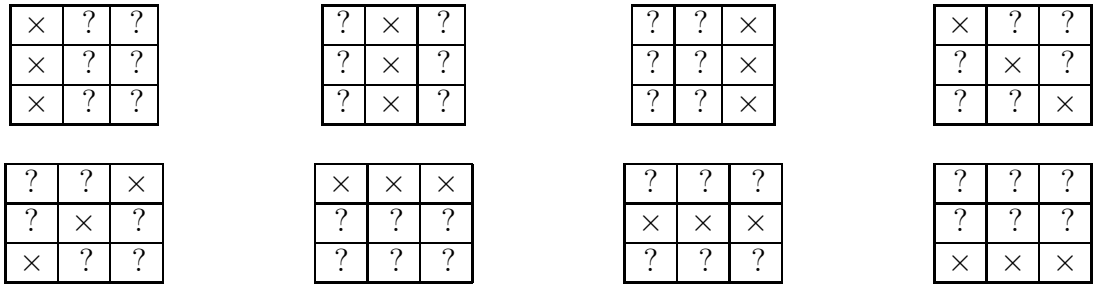
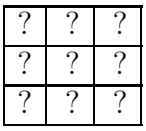


FIGURE 6.2 – Solution du morpion

Le corollaire 1.1 (page 8) nous a déjà fourni le moyen de déterminer si un problème nécessite ou non une définition disjonctive : il suffit de tester la correction par rapport à A^- du moindre généralisé de l'ensemble A^+. Dans la cas du morpion, le moindre généralisé de A^+ fournit l'hypothèse la plus générale :



qui est bien sûr incorrecte puisqu'elle couvre tous les exemples négatifs.

Ce problème nécessite donc une solution disjonctive et, en effet, il s'agit d'un problème référence pour les systèmes qui affrontent les problèmes disjonctifs [Aha, 1991]. Les sous-concepts semblent très difficiles à isoler : aucun système n'est parvenu à 100 % de bonnes prédictions en utilisant pourtant 70 % des données disponibles pour apprendre (soit plusieurs centaines d'exemples). À titre indicatif, voici les précisions obtenues sur les 30% restants par différents systèmes et rapportées dans [Aha, 1991] :

default	65.3 %
NewID	84.0 %
CN2	98.1 %
MBRtalk	88.4 %

IB1	98.1 %
IB3	82.0 %
IB3-CI	99.1 %

Le problème du morpion peut accepter une solution conjonctive si le concept d'alignement est connu. Par exemple, outre le contenu de chaque case, nous pouvons ajouter deux informations, $N_\times$ et $N_\circ$: respectivement, la taille du plus grand alignement de croix et la même information pour les ronds. Ces nouveaux attributs prennent leur valeur dans $\{0; 1; 2; 3\}$. Les exemples de la Figure 6.1 sont donc cette

fois codés par

$$A = x \ \wedge \ B = b \ \wedge \ C = o \ \wedge \ D = x \ \wedge \ E = x \ \wedge \ F = b \ \wedge \ G = x \\ \wedge \ H = o \ \wedge \ I = o \ \wedge \ N_\times = 3 \ \wedge \ N_\circ = 2$$

et

$$A = x \ \wedge \ B = x \ \wedge \ C = o \ \wedge \ D = o \ \wedge \ E = x \ \wedge \ F = x \ \wedge \ G = x \\ \wedge \ H = o \ \wedge \ I = o \ \wedge \ N_\times = 2 \ \wedge \ N_\circ = 2$$

Dans ce langage, le concept de victoire au morpion par les croix peut alors s'exprimer par la solution conjonctive suivante :

$$N_\times = 3$$

L'enrichissement de $\mathcal{L}_h$ par les attributs $N_\times$ et $N_\circ$ peut se justifier : en cours de partie, ils peuvent donner une idée sur le joueur qui a l'avantage. Voici, en revanche, une représentation plus discutable qui rend pourtant le problème du morpion conjonctif. Considérons le carré magique suivant :

2	9	4
7	5	3
6	1	8

Toutes les lignes, horizontales, verticales ou diagonales de ce carré ont 15 pour somme de leurs éléments. Ce carré a une autre propriété remarquable : si trois entiers distincts de ce carré ont pour somme 15, alors ces trois entiers sont nécessairement alignés dans le carré.

Un exemple est un ensemble d'entiers, inclus dans $\Omega = \{1; 2; \ldots; 15\}$, regroupant les valeurs des cases du carré magique qui contiennent une croix dans l'exemple. Autrement dit, nous repérons la position des croix à l'aide du carré magique. Ainsi, le premier exemple de la Figure 6.1 sera représenté par la formule suivante :

$$(2, 5, 6, 7)$$

et le second par

$$(2, 3, 5, 6, 9)$$

Une hypothèse de $\mathcal{L}_h$ pourra être, comme un exemple, un ensemble de valeurs choisies dans Ω. Par exemple, l'hypothèse suivante appartient à $\mathcal{L}_h$ et représente le concept de ligne de croix, horizontale, en haut.

$$(2, 4, 9)$$

Nous autorisons aussi des hypothèses plus complexes qui, toujours sur un sous-ensemble de Ω, peuvent utiliser des quantificateurs $\forall$ et $\exists$ et les opérations de l'arithmétique. Dans ce langage, une solution conjonctive est dénotée par la formule :

$$\exists a, b, c : a + b + c = 15$$

$\mathcal{L}_h$ autorise donc bien une solution conjonctive mais, cette fois-ci, elle est beaucoup moins pertinente : ce langage n'a absolument aucun sens pour un joueur de morpion.

En conclusion, notre choix de chercher une solution disjonctive plutôt que de modifier le langage $\mathcal{L}_h$ se justifie par le fait qu'une telle modification peut occasionner une sévère perte de compréhensibilité. Nous préférons conserver un langage compréhensible pour l'expert qui en général a lui-même défini $\mathcal{L}_h$; dans ce cas, la solution peut être disjonctive, sans que cela révèle une quelconque malformation du langage des hypothèses.

6.1.2 Explosion combinatoire

Une tendance est de voir la généralisation comme un problème de recherche [Mitchell, 1982]. C'est cette approche que nous avons développée dans la première partie de cette thèse, participant ainsi à l'effort de la communauté visant à élaguer l'espace de recherche et à optimiser la recherche elle-même. Citons quelques-unes des optimisations les plus efficaces.

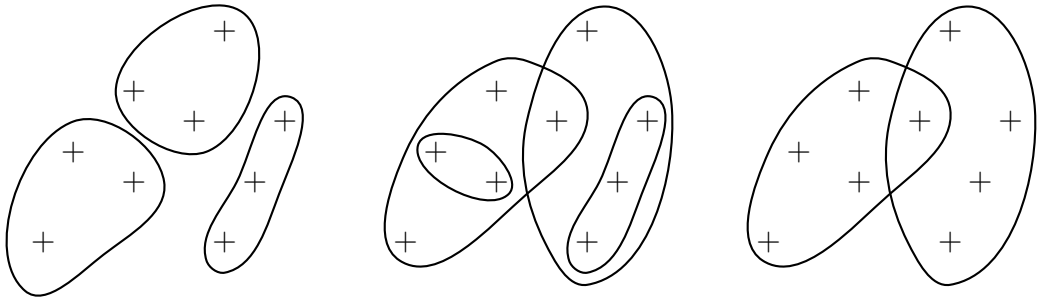
— La première optimisation est l'élagage originel proposé par Tom Mitchell, élagage par rapport à la correction et à la complétude [Mitchell, 1982].
— Nous avons nous-mêmes étendu cet élagage dynamique à toute propriété connue de la solution, à travers les *relations naturelles* décrites chapitre 4.
— Enfin, la recherche elle-même peut être optimisée. Par exemple, on peut imposer que chaque hypothèse ne soit considérée qu'une seule fois durant la recherche : ce sont les *opérateurs optimaux* [De Raedt et Bruynooghe, 1993], ainsi que le prolongement que nous en avons donné avec les *opérateurs parfaits* du chapitre 5.

Malheureusement, l'extension de ces méthodes du conjonctif au disjonctif est difficile. Comme nous l'avons déjà constaté à la section 5.4, (page 89), le pouvoir élagant des propriétés connues de la solution est considérablement affaibli lorsque ces propriétés portent sur une disjonction et non plus sur une hypothèse unique. En conjonctif, nous cherchions la solution dans $\mathcal{L}_h$, maintenant, la solution se trouve dans $2^{\mathcal{L}_h}$.

En particulier, l'élagage qui peut être considéré comme le plus puissant, par rapport à la couverture des positifs, s'effondre. En conjonctif, nous pouvions nous débarrasser d'une hypothèse dès qu'elle ne couvrait plus un exemple positif. En disjonctif, la complétude doit être assurée par la disjonction entière et il faut donc considérer qu'une hypothèse qui ne couvre plus qu'un unique exemple positif peut encore faire partie de la solution.

Quand bien même ces optimisations garderaient la même efficacité, elles seraient tout de même trop faibles face à l'explosion combinatoire produite par notre nouveau problème. Pour prendre conscience de ce phénomène, considérons l'ensemble des solutions possibles et, en particulier, les couvertures qu'elles induisent sur A^+.

Si en apprentissage conjonctif toutes les solutions sont équivalentes par rapport à A^+ et A^- (elles sont toutes correctes et complètes), il en va différemment lorsque le problème est disjonctif : les solutions, cette fois des disjonctions, sont toujours toutes correctes et complètes mais chacune d'elles induit une couverture différente de A^+ comme cela est illustré en Figure 6.3. La seule condition est que chaque hypothèse de la disjonction induise un paquet correct. Précisons immédiatement cette notion.



(a) (b) (c)

FIGURE 6.3 – Différentes couvertures de A^+ induites par disjonctions différentes, toutes correctes et complètes.

Définition 6.1 (paquet correct)
Un paquet correct est un sous-ensemble de A^+ dont le moindre généralisé ne couvre aucun exemple de A^-.

Dans la suite, nous parlerons indifféremment de paquet, de sous-ensemble de A^+ ou encore de sous-concept.

L'astuce de représentation unique autorise l'apprentissage par cœur ; autrement dit, la disjonction des exemples de A^+ est une solution. Cette solution n'est pas difficile à écarter. Elle révèle cependant toute la difficulté de ce problème : la solution par cœur est la plus spécifique mais combien y en a t-il de plus générales ?

Revenons au problème du morpion. Dans le cas où l'on utilise 50 exemples positifs pour A^+ et *tous* les exemples négatifs de la base pour A^-, il y a en moyenne plus de 5 000 sous-ensembles de A^+ qui sont corrects par rapport à A^- entier. Cette explosion est illustrée Figure 6.4, pour un nombre d'exemples allant de 1 à 55. Précisons que la base contient au total 626 exemples positifs et que les systèmes qui utilisent 70% des données pour apprendre ont donc plus de 400 exemples positifs à leur disposition.

Ainsi, il y a beaucoup de paquets corrects. Il y a encore plus de couvertures correctes et complètes, c'est-à-dire de manières de combiner ces paquets corrects pour obtenir une couverture de A^+. Et encore plus de solutions possibles : pour chaque couverture, il peut y avoir plusieurs solutions possibles selon les hypothèses de $\mathcal{L}_h$ qui sont choisies pour caractériser chaque paquet de la couverture.

Nous allons maintenant étudier comment les principales méthodes de construction d'une solution disjonctive opèrent leur choix d'une solution parmi toutes les solutions possibles.

6.2 Méthodes

Plusieurs types de méthodes peuvent être distingués. [Quinlan, 1990, Boström, 1995] en opposent deux : les *diviser pour régner* et les *approches par couverture*.

Dans ces deux approches, des choix heuristiques sont effectués, ces choix portant sur des décisions locales. Par suite, l'optimalité globale de la solution apprise pour le critère à optimiser ne peut être garantie : c'est pourquoi nous parlerons de méthodes

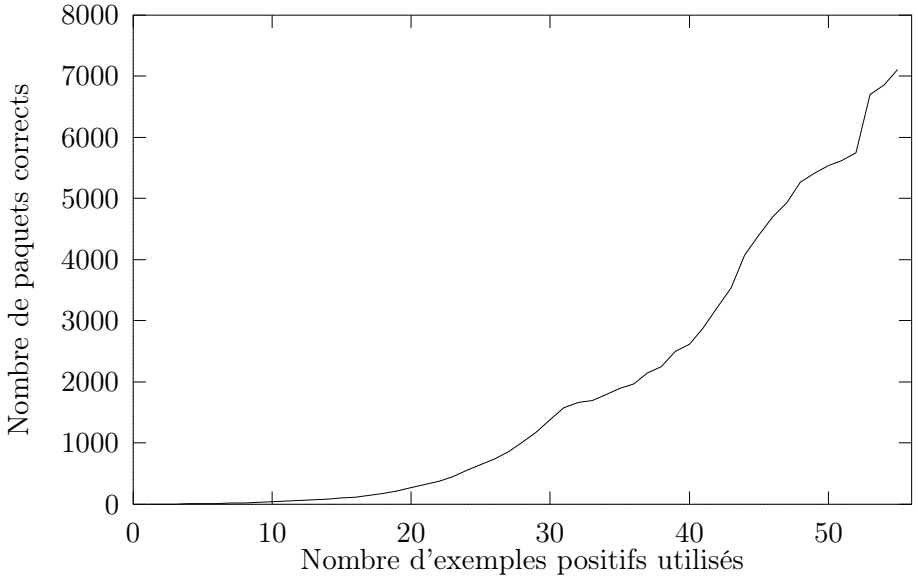


FIGURE 6.4 – Paquets corrects pour le morpion. Le nombre d'exemples positifs disponibles varie de 1 à 55, tous les exemples négatifs présents, nombres de paquets corrects moyens obtenus sur 100 tirages aléatoires.

locales.

6.2.1 Approches diviser pour régner

Les représentants les plus connus de cette famille d'algorithmes sont ID3 (*Iterative Dichotomizer version 3*) [Quinlan, 1986] et C4.5 [Quinlan, 1993]. Le schéma général est donné à l'Algorithme 6.1.

Algorithme 6.1 (diviser pour régner)
Entrées : des ensembles d'exemples A^+ et A^-, et une hypothèse H de $\mathcal{L}_h$ qui couvre tous les exemples de A^+ et A^- (H est donc complète et incorrecte si $A^- \neq \emptyset$).

1. *Si $A^- = \emptyset$ alors retourner H sinon spécialiser H en plusieurs nouvelles hypothèses notées H_i.*
2. *Pour chaque H_i, on construit A_i^+ et A_i^- qui sont respectivement les exemples de A^+ couverts par H_i et les exemples de A^- couverts par H_i.*
3. *L'algorithme est rappelé sur (A_i^+, A_i^-, H_i).*

Les méthodes de cette famille sont donc toutes descendantes. Dans le cas de ID3, c'est-à-dire dans un langage attribut-valeur, cet algorithme démarre avec A^+, A^- et l'hypothèse la plus générale. La spécialisation d'une hypothèse H opère de la manière suivante :

— un attribut A est sélectionné à l'aide d'un critère d'entropie ; c'est sur cette heuristique que l'on compte pour obtenir, au final, la solution la plus simple possible.

— pour chaque valeur possible v_i de l'attribut A, une nouvelle hypothèse H_i est obtenue en ajoutant la condition $(A = v_i)$ à l'hypothèse H.

À l'instar de ID3, les méthodes diviser-pour-régner construisent toutes les hypothèses de la disjonction en parallèle. Cependant, au fur et à mesure que l'on spécialise les hypothèses, chacune d'entre elles ne couvre plus que très peu d'exemples de l'ensemble d'apprentissage. Outre une explosion du nombre d'hypothèses dans la disjonction survient une perte de support statistique au fil des appels récursifs : les hypothèses finalement dans la solution sont difficilement justifiables car le critère entropique suppose un nombre suffisant d'exemples, hypothèse qui n'est plus vérifiée lorsque l'on arrive aux feuilles de l'arbre.

Par ailleurs, le critère se fonde sur des décisions locales : à chaque étape, il désigne l'attribut le plus discriminant, mais rien ne garantit que ce critère soit optimal pour l'arbre final obtenu. Preuve en est qu'il est courant d'élaguer a posteriori l'arbre obtenu, c'est le cas par exemple de C4.5. Nous verrons d'ailleurs à la section 6.3 que cette simplification post-apprentissage de l'arbre peut poser de sérieux problèmes.

6.2.2 Approches par couverture

Dans les approches par couverture, nous trouvons des méthodes ascendantes, aussi bien que descendantes, au contraire des approches diviser pour régner qui sont toutes descendantes. FOIL [Quinlan, 1990] et la famille des algorithmes AQ [Michalski et al., 1986] font partie des approches par couverture. L'algorithme générer-et-tester, décrit à la section 2.2, en est aussi un représentant. D'ailleurs, l'algorithme 6.2, représentant les approches par couverture, est une reformulation de la boucle externe du générer-et-tester (algorithme 2.2).

Algorithme 6.2 (par couverture)
Étant donnés A^+ et A^-.

1. *La disjonction solution est initialisée à $S = \emptyset$.*
2. *Tant que $A^+ \neq \emptyset$*
   (a) *Trouver H dans $\mathcal{L}_h$ telle que H ne couvre aucun exemple de A^-.*
   (b) *Écarter les exemples de A^+ couverts par H :*
   $$A^+ = A^+ - \{e \in A^+ : H \succeq e\}$$
   (c) *Ajouter H à la disjonction : $S = S \cup \{H\}$.*
3. *Renvoyer S.*

Contrairement aux approches diviser-pour-régner, les hypothèses de la disjonction sont ici construites les unes après les autres : une hypothèse H correcte par rapport à A^- est découverte, par exemple en utilisant un générer-et-tester, les exemples de A^+ couverts par H sont écartés, une nouvelle hypothèse est recherchée pour couvrir les exemples restants dans A^+. Précisons tout de même qu'il existe un paramètre de AQ, fort peu usité, permettant de ne pas effectuer l'étape (2b), ce qui conduit à considérer tous les exemples positifs comme graine.

Le choix des biais de recherche dans ces méthodes influe très fortement sur le résultat. En particulier, le choix de la première hypothèse appartenant à la solution est déterminant pour la suite de la solution et il n'y a pas de retour arrière.

À nouveau, c'est un critère local qui peut faire échouer la construction d'une solution satisfaisante. Cela nous amènera à considérer à la section 6.4 des méthodes que nous qualifierons de *globales*.

6.3 Difficultés identifiées

Nous présentons les principaux problèmes rencontrés par les méthodes que nous venons d'évoquer. Les deux premiers, le problème de la réplication et celui des généralisations hâtives, sont propres aux approches diviser-pour-régner. En revanche, les deux problèmes suivants, le problème des petits paquets et celui de la taille de la disjonction, sont partagés par les deux approches.

6.3.1 Problème de la réplication

Le *replication problem* a été précisément identifié par [Pagallo et Haussler, 1990] pour un langage n'utilisant que des attributs booléens. Les auteurs de cet article s'interrogent sur la taille de l'arbre de décision minimal représentant un concept donné. Reprenons leur exemple, considérons le concept défini par la formule

$$(x_1 \wedge x_2) \vee (x_3 \wedge \overline{x_4} \wedge x_5)$$

La Figure 6.5 montre l'arbre de décision minimal représentant ce concept.

La formule dont nous sommes partis est une disjonction de taille deux, l'une des conjonctions fait intervenir deux variables, l'autre trois. L'arbre lui contient neuf feuilles dont six concluent à faux : pour découvrir deux sous-concepts, une éventuelle base d'apprentissage aurait été partitionnée en neuf sous-ensembles.

En observant l'arbre, il est aisé de comprendre la raison de ce fractionnement : un sous-arbre a été dupliqué. Malheureusement, cette réplication est systématique : à chaque fois que le test d'une branche provoque l'échec d'une conjonction, il faut coder la conjonction suivante sur cette branche.

[Pagallo et Haussler, 1990] indique que le nombre de feuilles concluant à faux dans l'arbre de décision minimal est supérieur au produit de la taille des conjonctions dans la plus petite formule équivalente.

Outre le fractionnement injustifié de la base d'exemples, il apparaît donc que l'arbre de décision minimal est beaucoup plus complexe que l'ensemble de règles équivalent. C'est la raison pour laquelle, il est très courant que les systèmes apprenant des arbres de décison se livrent à ce que nous pourrions appeler une optimisation globale post-induction : l'arbre est transformé en ensemble de règles (chaque chemin entre la racine et une feuille formant une règle), puis on cherche à généraliser et à recouper ces règles, pour en obtenir un nombre minimal.

Nous avons vu que le problème de la réplication contraignait à ce type d'optimisation mais, malheureusement cette solution va introduire un nouveau problème, celui des *généralisations hâtives* identifié par [Frank et Witten, 1998].

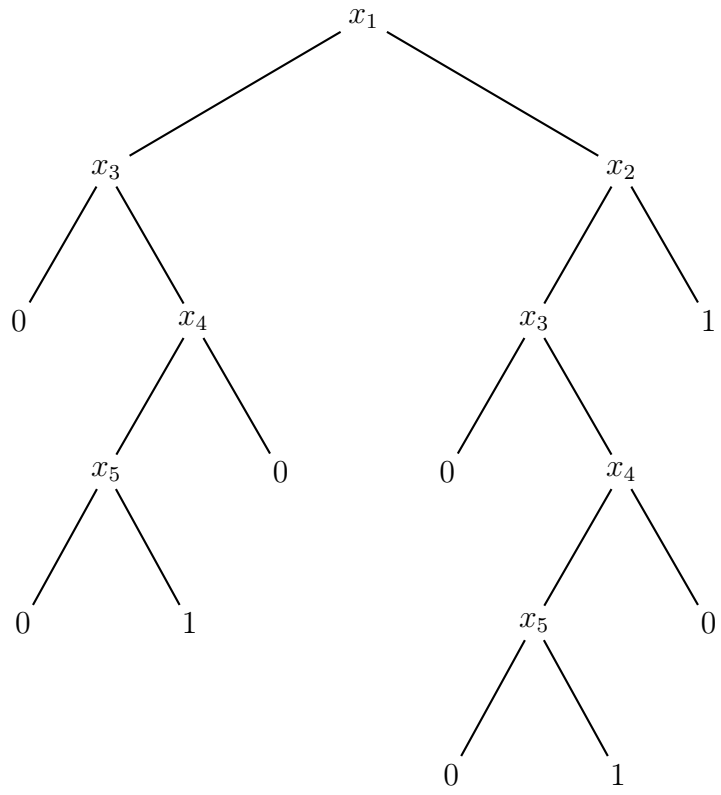


FIGURE 6.5 – Arbre de décision minimal pour $(x_1 \wedge x_2) \vee (x_3 \wedge \overline{x_4} \wedge x_5)$. Le sous-arbre ayant x_3 pour racine est dupliqué.

6.3.2 Problème des petits paquets

Le problème des *small disjuncts* repéré par [Holte et al., 1989] n'est pas lié à une méthode d'apprentissage particulière mais à certains problèmes connus comme délicats. À nouveau, il s'agit de justifier une règle qui ne couvre que très peu d'exemples mais cette fois-ci le fractionnement n'est pas en cause. Le problème est que certains sous-concepts sont, par nature, peu représentés. Typiquement, un tel sous-concept représente les cas particuliers du concept.

Or, [Holte et al., 1989] montre expérimentalement que ce sont les hypothèses construites pour couvrir ces petits paquets qui ont une très mauvaise précision.

6.3.3 Taille de la disjonction

Les approches que nous avons considérées à la section précédente ne garantissent pas de trouver une solution de taille minimale (en terme de nombre de paquets dans la couverture), ni même d'obtenir une solution de taille raisonnable. Plus particulièrement, ces méthodes peuvent fournir une solution disjonctive (plusieurs paquets) alors qu'une solution conjonctive (un seul paquet reprenant tous les exemples positifs) existe. Finalement, ces méthodes sont sujettes à l'éclatement injustifié de A^+ et à l'explosion de la taille de la disjonction découverte. Cela occasionne en particulier une perte de compréhensibilité de la solution.

Nous l'avons vu avec le morpion et la Figure 6.4 : les candidats sont très nombreux et nous voyons mal comment un critère local pourrait trouver une combinaison de plus petite taille parmi ceux-ci. C'est pourquoi nous allons nous orienter vers des méthodes n'utilisant pas de critère local.

6.4 Objectifs

6.4.1 Approches globales

Beaucoup de méthodes existantes appartiennent à l'une des deux familles que nous avons décrites section 6.2. Celles-ci parcourent le langage des hypothèses $\mathcal{L}_h$ pour trouver les éléments de la disjonction, guidées par un critère local.

Nous avons évoqué les problèmes créés par l'utilisation d'un critère local et c'est pourquoi nous définissons les approches globales par différenciation : une méthode globale doit optimiser un critère global sans utiliser l'approximation qui consiste à tester ce critère localement.

Typiquement, un tel critère traduit une notion de simplicité, le but étant de trouver la solution la plus simple. Pour notre part, nous choisissons de confondre la simplicité avec le nombre minimal de sous-concepts : autrement dit, la couverture de A^+ doit se faire par un nombre minimal de paquets. Il s'agit bien d'un critère global : il porte sur l'ensemble de la disjonction des paquets.

Ce choix nous conduit à considérer des algorithmes qui se distinguent en utilisant l'ambivalence d'une hypothèse : sa définition à l'aide d'un mot du langage $\mathcal{L}_h$ et les exemples de $\mathcal{L}_e^+$ qu'elle couvre (et plus particulièrement ceux de A^+). L'idée est la suivante : construire d'abord des paquets d'exemples positifs, découvrir un nombre

minimal de ces paquets permettant de couvrir A^+ et seulement ensuite trouver une solution conjonctive dans $\mathcal{L}_h$ pour chacun de ces paquets.

Le meilleur représentant de ce type de méthodes est sans doute celui de [Erdem et Flener, 1997], dans le contexte de la synthèse de programme. Un graphe est construit dont les sommets sont les exemples positifs et dont les arêtes lient les exemples positifs pour lesquels la généralisation respecte certaines contraintes syntaxiques. À partir de ce graphe, nous cherchons une couverture minimale par un ensemble de cliques maximales (une clique est un ensemble de sommets qui sont tous liés deux à deux dans le graphe). Enfin, il reste à caractériser chaque paquet en prenant, par exemple, son moindre généralisé.

Nous ne disposons pas de l'équivalent de ces contraintes syntaxiques et il est de toute façon contrariant que les exemples négatifs n'interviennent pas dans la construction des paquets corrects. C'est pourquoi nous proposons maintenant une adaptation de cette méthode à nos préoccupations, en remplaçant les critères syntaxiques par la correction vis-à-vis de A^-.

6.4.2 Couverture par cliques maximales

La méthode d'apprentissage que nous présentons ici constitue une première étape dans la démarche qui nous a finalement conduit au système GloBo qui sera décrit au chapitre suivant. Cette méthode utilise la notion de *graphe de correction* des exemples que nous définissons immédiatement.

Définition 6.2 (graphe de correction)
Le graphe de correction $\mathcal{G}$ représentant (A^+, A^-) est défini par :
- *l'ensemble des sommets de $\mathcal{G}$ est A^+ ;*
- *il y a un lien dans $\mathcal{G}$ entre e_1^+ et e_2^+ si et seulement si le moindre généralisé de $\{e_1^+; e_2^+\}$ est correct par rapport à A^-.*

À partir de cette représentation du problème, l'apprentissage consiste à rechercher une couverture minimale de $\mathcal{G}$ par un ensemble de cliques maximales de $\mathcal{G}$, comme cela est illustré Figure 6.6. Ensuite, il faut caractériser chaque paquet à l'aide d'une hypothèse de $\mathcal{L}_h$. À nouveau, il est possible de prendre le moindre généralisé de chaque paquet.

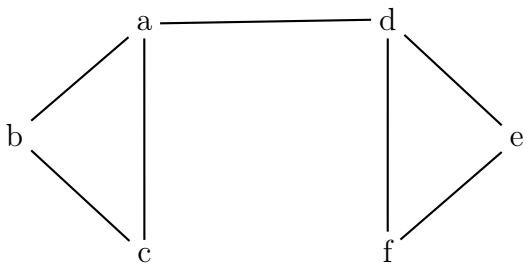


FIGURE 6.6 – Couverture minimale par des cliques maximales. Dans ce cas, l'unique solution est $\{\{a, b, c\}, \{d, e, f\}\}$

Il y a cependant un risque de ne pas obtenir une solution correcte : dans une clique, nous avons une correction des exemples deux à deux mais rien ne garantit que la clique dans son ensemble soit correcte par rapport à A^-.

Signalons que cette construction est favorable à l'utilisation d'une subsomption incomplète comme la subsomption stochastique [Sebag et Rouveirol, 1997] que nous avons évoquée section 1.5. Un tel test de subsomption indique parfois à tort qu'une hypothèse ne subsume pas un exemple. Dans notre cadre, cela signifie qu'un lien est construit entre deux exemples dans le graphe de correction alors que leur moindre généralisé couvre un exemple négatif. Or, comme cela est illustré Figure 6.7, la recherche de cliques maximale est robuste : il faudrait ajouter beaucoup de mauvais liens pour que les cliques découvertes soient différentes.

La phase de construction des paquets n'est pas réellement dissociée du langage de représentation des hypothèses puisque la définition du graphe de correction utilise un calcul de moindre généralisé et donc $\mathcal{L}_h$.

Cependant, nous nous sommes affranchis du parcours de $\mathcal{L}_h$ et des biais de recherche qui y sont liés : par exemple, nous n'avons pas à spécifier où commencer la recherche. Autrement dit, comme nous le voulions, la méthode est globale, il n'y a pas de critère local comme dans les cas précédents.

Du point de vue de la complexité, la construction du graphe de correction nécessite $|A^+|^2$ calculs de moindres généralisés et, dans le pire des cas, $|A^+|^2 \times |A^-|$ tests de subsomption. En revanche, trouver une couverture du graphe de correction par des cliques maximales est une tâche NP-difficile : trouver la plus grande clique d'un graphe et trouver la couverture minimale d'un ensemble sont, à eux seuls, des problèmes NP-difficiles [Garey et Johnson, 1979]. Même si nous utilisons une approximation de cette couverture, le prix à payer nous a semblé trop élevé pour finalement n'obtenir qu'une disjonction qui n'est peut-être pas correcte.

Comme nous l'avons signalé, la méthode que nous venons de décrire est un ancêtre du système GloBo que nous présenterons au chapitre suivant : celui-ci construit des paquets d'exemples positifs parfaitement corrects vis-à-vis de A^-, et cela, avec la même complexité cubique que pour la construction du graphe de correction.

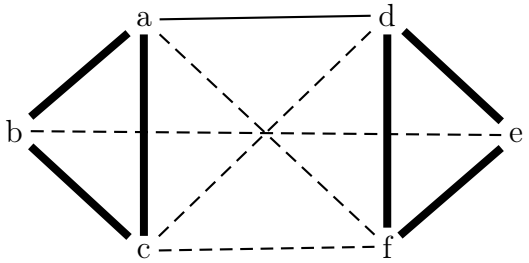


FIGURE 6.7 – Résistance de la couverture minimale. Il est possible de rajouter de nombreux faux liens (ici en pointillés) sans pour autant perdre la solution.

6.4.3 Apprentissages supervisé et non supervisé

Former des paquets pertinents avec un critère global... cette problématique n'est pas sans rappeler celle de l'apprentissage non supervisé. Étant données une distance d sur les exemples et une borne d_{max} sur cette distance, il s'agit de trouver un nombre minimal de paquets, chacun de ces paquets C devant vérifier

$$\text{satisfaisant}(C) \Leftrightarrow \forall (a,b) \in C, d(a,b) \leq d_{max}$$

Mis à part le fait que cette propriété se substitue à la correction par rapport à A^- en supervisé, les deux problèmes sont identiques.

Le lien entre les deux types d'apprentissage est bien connu [Langley et al., 1987] et va même au-delà de la ressemblance que nous venons d'exhiber : il a été souligné que les méthodes d'apprentissage supervisé pouvaient réaliser des tâches non supervisées, que les êtres humains apprenaient sans étiquetages explicites des exemples, ou encore que l'apprentissage non supervisé pouvait être utilisé sur des données non étiquetées en vue d'enrichir un corpus qui lui, est étiqueté [Caruana et al., 1998].

Considérons donc l'apprentissage non supervisé. Plus généralement, son but est, étant donné un ensemble d'instances, de construire un ensemble de concepts qui classifient ces instances [Gennari et al., 1989].

D'une part, nombre de méthodes sont fournies par l'Analyse de Données. Par exemple, la *classification ascendante hiérarchique* [Diday et al., 1982] réalise un apprentissage non supervisé de la manière suivante : on démarre avec une instance par classe, puis deux classes (choisies selon un indice de similarité) sont fusionnées, et l'on recommence jusqu'à ce que toutes les instances soient dans la même classe. Ainsi, à chaque étape, une partition différente des instances est découverte ; ces partitions permettent finalement de construire une hiérarchie de concepts. Parmi les algorithmes de classification par partitions, évoquons rapidement les *nuées dynamiques* de [Diday et al., 1982]. k noyaux sont choisis aléatoirement, puis chaque point (instance) est associé au noyau le plus proche, obtenant ainsi k paquets ; ensuite, chaque noyau est remplacé par le centre de gravité de son paquet, et l'on recommence à partir de ces nouveaux noyaux. Les nuées dynamiques sont capables d'optimiser un critère donné sur les paquets à former. En revanche, cette méthode ne permet de trouver un nombre minimal de paquets, puisque la valeur de k doit être fixée à l'avance.

D'autre part, selon [Fisher, 1987] (COBWEB) et [Gennari et al., 1989] (CLASSIT), on peut voir l'apprentissage non supervisé comme une recherche à travers l'espace des hiérarchies. À nouveau, nous trouvons les problèmes soulevés par l'apprentissage supervisé : efficacité de la recherche, optimalité du parcours de l'espace de recherche, définition d'opérateurs, absence de contrôle sur la taille des solutions, etc.

Dans toute la suite de cette thèse, nous discuterons essentiellement d'apprentissage supervisé. Cependant, il est bien clair maintenant que l'on peut passer du supervisé au non supervisé en remplaçant le critère de correction d'un paquet par une propriété P . Une différence est tout de même à signaler : dans le cas du non supervisé, on a :

$$P(C) \Leftrightarrow \forall a, b \in C, P(\{a,b\})$$

En supervisé, on a seulement :

$$P(C) \Rightarrow \forall a, b \in C, P(\{a, b\})$$

En apprentissage non supervisé, la satisfaction de P par un ensemble d'exemples est impliquée par la satisfaction de P par ces exemples pris deux à deux, ce qui n'est pas le cas en apprentissage supervisé comme nous l'avons vu en discutant les propriétés du graphe de correction. Il nous semble donc que, de ce point de vue, l'apprentissage non supervisé est plus facile que l'apprentissage supervisé.

Cependant, ces deux propriétés P présentent une caractéristique intéressante qui sera la base des méthodes décrites dans les chapitres suivants : si un ensemble E vérifie P alors chaque sous-ensemble de E vérifie aussi P, c'est-à-dire :

$$P(A \cup B) \Rightarrow P(A) \wedge P(B) \tag{6.1}$$

6.4.4 Définition du problème

Nous avons vu que l'apprentissage disjonctif passait, explicitement ou non, par la construction de paquets corrects. Nous commençons par exiger que nos paquets soient *maximalement* corrects pour l'inclusion.

Définition 6.3 (maximalement correct)
Un sous-ensemble E de A^+ est maximalement correct si et seulement si
- *E est correct par rapport à A^-, c'est-à-dire que le moindre généralisé de E ne couvre aucun exemple négatif;*
- *l'ajout dans E d'un exemple de A^+, non déjà présent de E, provoque la perte de la correction de E.*

Ce choix se justifie simplement en disant que la seule raison qui empêche de réunir des exemples positifs est que leur moindre généralisé couvre des exemples négatifs. Si de tels exemples négatifs n'existent pas, il n'y a aucune raison de séparer les exemples positifs considérés.

De plus, cela réduit grandement le nombre de candidats comme l'illustre la Figure 6.8. Cependant, moins de candidats n'entraîne pas nécessairement un gain d'efficacité : pour trouver les paquets maximalement corrects, il faut parcourir tous les paquets corrects.

Nous allons de plus réclamer que la couverture trouvée soit minimale puisqu'en définitive nous voudrons un nombre minimal de paquets maximalement corrects dont l'union couvre l'ensemble A^+. En particulier, ce choix de la taille minimale impose que, si une solution conjonctive existe, c'est elle qui doit être découverte.

Nous ajoutons donc nos nouvelles conditions à la définition 1.5

Définition 6.4 (apprentissage supervisé disjonctif)
Étant donnés $(\mathcal{L}_e, \mathcal{L}_h, A^+, A^-, \succeq)$, trouver un ensemble de taille minimale $\mathcal{H} \subseteq 2^{\mathcal{L}_h}$ vérifiant

$$\forall e \in A^+, \exists h \in \mathcal{H} : e \in h$$
$$\forall h \in \mathcal{H} : \ h \text{ est maximalement correct par rapport à } A^-$$

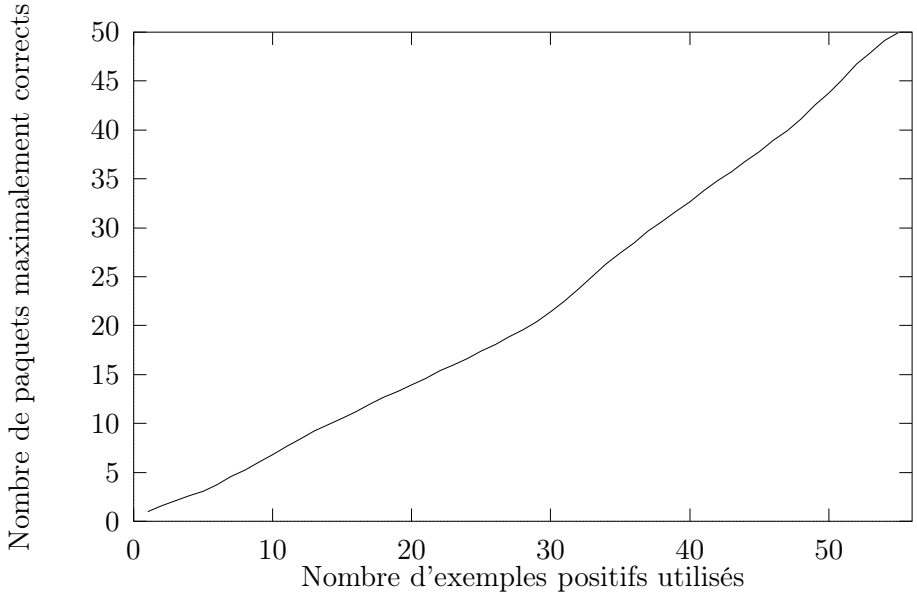


FIGURE 6.8 – Paquets maximalement corrects pour le morpion

Notons que cette définition n'exige pas que les paquets soient disjoints : le recouvrement des paquets importe peu, seule la minimalité de la couverture nous intéresse.

6.5 Bilan

Dans ce chapitre, nous avons présenté le problème de l'apprentissage disjonctif et plus particulièrement nous avons montré l'explosion combinatoire qui survenait en passant de l'apprentissage conjonctif à la recherche d'une solution disjonctive.

Puis nous avons donné le principe des principales familles de systèmes apprenant des définitions disjonctives et les difficultés qu'elles rencontraient.

Par opposition avec ces méthodes et en espérant contourner certaines de leurs difficultés, nous avons défini les appproches globales. Nous avons vu que cette approche permettait de réaliser un apprentissage supervisé aussi bien que non supervisé.

Puis nous avons défini le problème auquel nous allons nous consacrer dans la suite de cette thèse : parmi les trois approches que nous avons exposées, nous faisons le choix des approches globales en choisissant la taille de la solution comme critère global à optimiser ; de plus, nous posons que les paquets d'exemples positifs induits par la solution doivent être maximalement corrects.

Dans ce cadre, nous présenterons deux méthodes :

— au chapitre 7, nous présenterons GloBo, une version stochastique de cette approche ;
— au chapitre 8, nous décrirons les Vraizamis, un système d'éco-résolution qui convenablement instancié résout des problèmes d'apprentissage disjonctif.

Bibliographie

[Aha, 1991] Aha, D. W. (1991). Incremental constructive induction : An instance-based approach. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 117–121. Morgan Kaufmann.

[Boström, 1995] Boström, H. (1995). Covering vs. divide-and-conquer for top-down induction of logic programs. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1194–1200.

[Caruana et al., 1998] Caruana, R., de Sa, V., Kearns, M., et McCallum, A., éditeurs (1998). *NIPS*98 Workshop : Integrating Supervised and Unsupervised Learning.*

[De Raedt et Bruynooghe, 1993] De Raedt, L. et Bruynooghe, M. (1993). A theory of clausal discovery. In Bajcsy, R., éditeur, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann.

[Diday et al., 1982] Diday, E., Lemaire, J., Pouget, J., et Testu, F. (1982). *Éléments d'analyse de données*. Dunod.

[Erdem et Flener, 1997] Erdem, E. et Flener, P. (1997). A redefinition of least generalizations and its application to inductive logic program synthesis. Rapport interne. non publié.

[Fisher, 1987] Fisher, D. H. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2(2) :139,172.

[Frank et Witten, 1998] Frank, E. et Witten, I. H. (1998). Generating accurate rule sets without global optimization. In Shavlik, J., éditeur, *Proceedings 15th International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann.

[Garey et Johnson, 1979] Garey, M. et Johnson, D. (1979). *Computers and Intractability - A Guide to Theory of NP-Completeness*. Freeman, W.H. and Co., New York.

[Gennari et al., 1989] Gennari, J. H., Langley, P., et Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40 :11–61.

[Holte et al., 1989] Holte, R. C., Acker, L. E., et Porter, B. W. (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 813–818.

[Langley et al., 1987] Langley, P., Simon, H. A., Bradshaw, G. L., et M., Z. J. (1987). *Scientific Discovery : Computational Explorations of the Creative Process*. Machine Intelligence, eds : Meltzer, and Michie, vars. PublishersT Press. ACM CR 8807-0489.

[Merz et Murphy, 1996] Merz, C. J. et Murphy, P. M. (1996). UCI repository of machine learning databases.

[Michalski et al., 1986] Michalski, R., Mozetic, I., Hong, J., et Lavrac, N. (1986). The AQ15 inductive learning system : an overview and experiments. In *Proceedings of IMAL 1986*, Orsay. Université de Paris-Sud.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18 :203–226.

[Pagallo et Haussler, 1990] Pagallo, G. et Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1) :71–99.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1) :81–106.

[Quinlan, 1990] Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3) :239–266.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

[Sebag et Rouveirol, 1997] Sebag, M. et Rouveirol, C. (1997). Tractable induction and classification in FOL via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–892.

[Utgoff, 1986] Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In Michalski, R. S., Carbonell, J. G., et Mitchell, T. M., éditeurs, *Machine Learning : An Artificial Intelligence Approach*, volume II, pages 107–148. Morgan Kaufmann.

CHAPITRE 7

GloBo

Une erreur peut devenir exacte
selon que celui qui l'a commise
s'est trompé ou non.
Pierre Dac.

Nous proposons dans ce chapitre une approche stochastique du problème d'apprentissage supervisé disjonctif : le système GloBo [Torre, 1999a].

Nous décrirons les algorithmes de GloBo : fondés sur des tirages aléatoires, ils sont de complexité cubique dans le nombre d'exemples disponibles. L'aspect stochastique de la méthode conduit à évaluer la confiance que l'on peut accorder à l'apprentissage ou, autrement dit, à mesurer l'adéquation entre les ensembles d'apprentissage, la solution découverte et le système GloBo.

Nous validerons cette méthode à travers plusieurs expérimentations ; en particulier, nous présenterons les résultats de GloBo confronté au *PTE Challenge* que nous avons évoqué section 1.8 (page 19).

Nous terminerons avec une version générique de notre algorithme et quelques instanciations possibles : une instanciation de complexité linéaire, une pour le non supervisé et une pour traiter les données bruitées.

7.1 Introduction

Puisque les préoccupations des informaticiens, notamment en Intelligence Artificielle, se ramènent souvent à des problèmes NP-difficiles, il a fallu développer des méthodes permettant d'apporter des réponses à ces problèmes en temps raisonnable. Sous l'hypothèse que P est différent de NP, la nature même de ces problèmes exclut l'existence d'un algorithme exact travaillant en temps polynomial. En revanche, il est possible d'obtenir des approximations.

Une fois que nous avons accepté de ne pas obtenir une solution optimale, il est souvent facile d'obtenir une solution acceptable. Par exemple, dans le cas du voyageur de commerce, si nous n'exigeons plus le chemin le plus court, il est aisé de construire un chemin quelconque, qui pourra être le plus court comme le plus long.

Bien sûr, il n'est pas tolérable d'obtenir le chemin le plus long ; c'est pourquoi une bonne approximation est toujours exigée, c'est-à-dire une solution assez proche de la solution optimale. Parfois, il est même possible de borner le taux d'erreur de l'algorithme. Le voyageur de commerce pourra alors exiger, par exemple, un chemin dont la longueur n'excède pas de plus de 5% celle du chemin le plus court. Naturellement, la complexité de l'algorithme croît avec la qualité requise de l'approximation.

Notons que trouver une solution approchée à un problème NP-difficile peut se révéler être aussi un problème NP-difficile. Cependant, dans la plupart des cas, un algorithme polynomial existe et, paradoxalement, un tel algorithme peut trouver une solution très proche de l'optimal : l'explosion combinatoire survient dans la transformation d'une bonne solution en une solution optimale.

Tout aussi paradoxalement, du moins au premier abord, un algorithme polynomial peut fournir la solution exacte. Cela parce que les cas où le temps de résolution devient exponentiel, et qui valent au problème d'être NP-difficile, sont rares. Il s'agit de phénomènes de transition de phase [Hogg et al., 1996]. Soit le problème n'est pas contraint et il y a tellement de solution qu'il est facile d'en trouver une, soit le problème est très contraint et il est tout aussi aisé de s'apercevoir qu'il n'y pas de solution. La difficulté réelle se situant entre ces deux extrêmes dans une région très limitée.

Beaucoup de problèmes NP-difficiles, en particulier l'apprentissage disjonctif, nécessitent d'énumérer un nombre exponentiel d'éléments. Dans ce cas, une approximation polynomiale possible consiste à construire, aléatoirement, un nombre polynomial de ces éléments. Bien sûr, si les tirages aléatoires peuvent être orientés de manière à construire les éléments les plus pertinents, l'approximation n'en sera que meilleure. C'est ce type d'approches que nous allons considérer ici pour notre problème d'apprentissage disjonctif.

Mis à part les algorithmes évolutionnaires [Holland, 1975], les approches stochastiques sont peu répandues en apprentissage automatique. Nous avons tout de même déjà rencontré une approximation stochastique utilisée en apprentissage : la subsomption stochastisque [Sebag et Rouveirol, 1997] décrite à la section 1.5.

Présentons d'abord un algorithme exact, basé sur une énumération des sous-ensembles maximalement corrects de A^+, dont nous fournirons ensuite une version stochastisque et polynomiale.

Algorithme 7.1 (méthode exhaustive)

1. *Calculer tous les sous-ensembles de A^+ qui soient maximalement corrects par rapport à A^-.*
2. *Calculer la couverture minimale de A^+ par ses sous-ensembles maximalement corrects.*
3. *Caractériser chaque sous-ensemble E_i de la couverture minimale par une hypothèse H_i de $\mathcal{L}_h$.*
4. *Renvoyer la disjonction des H_i.*

Dans le pire des cas, le nombre de sous-ensembles considérés durant la première étape est $2^{|A^+|}$; quelle que soit l'optimisation envisagée, l'ordre de grandeur reste celui-ci. Pour chacun de ces sous-ensembles, il faudra calculer son moindre généralisé puis tester la correction de cette généralisation par rapport à A^- : au pire, cela nécessitera $|A^-|$ tests de subsomption à chaque fois.

L'algorithme stipule ensuite le calcul de la couverture minimale de A^+ par les ensembles repérés durant la phase précédente. Il s'agit d'un problème NP-difficile en soi qui, de surcroît, s'applique ici à un nombre d'ensembles candidats exponentiel.

Enfin, la caractérisation d'un sous-ensemble de A^+ peut se faire par le calcul de son moindre généralisé, ou par recherche conjonctive, par exemple avec un algorithme générer-et-tester aidé des méthodes décrites dans la première partie de cette thèse.

Le principe de GloBo est de générer aléatoirement des paquets d'exemples de A^+ qui soient maximalement corrects, autant de paquets qu'il y a d'exemples dans A^+. Le nombre de paquets sera donc linéaire dans la taille de l'ensemble d'apprentissage. Puis, GloBo utilisera une approximation polynomiale pour le calcul d'une couverture de A^+ par ces paquets. Nous verrons que cette procédure trouve la solution exacte en temps polynomial, sous condition que les données soient suffisamment *représentatives* de cette solution. Cette notion de représentativité sera définie à la section suivante.

7.2 Algorithmes

7.2.1 Algorithmes

Construction de paquets conjonctifs

L'algorithme principal de GloBo construit un paquet $P(s)$ à partir d'un exemple particulier s de A^+ appelé *graine*. Le paquet construit doit être maximalement correct (voir définition 6.1, 99) par rapport à A^- : ce paquet doit être correct et l'ajout dans ce paquet d'un exemple supplémentaire provoque la perte de la correction.

Pour cela, nous allons simplement parcourir la séquence des exemples positifs autres que la graine : nous ajouterons un exemple au paquet en cours de construction que si la correction de ce paquet est préservée.

Algorithme 7.2 (paquet maximalement correct)
Étant donnés une graine s et $E = \{e_1, e_2 \ldots, e_{|A^+|-1}\}$ une séquence regroupant les autres exemples de A^+.

1. *Soit $C = \{s\}$;*

2. *Pour* i *dans* $1 \ldots |E| - 1$ *faire : si* $C \cup \{e_i\}$ *est un paquet correct alors* $C = C \cup \{e_i\}$;
3. *Retourner* C.

Cet algorithme est illustré à la Figure 7.1 : nous constatons que, par construction, un paquet fourni par cet algorithme est maximalement correct par rapport à A^-. Dans les termes d'une approche AQ [Michalski, 1983], nous dirions que $P(s)$ est un élément de l'étoile de la graine s. Signalons aussi l'espace de versions disjonctives [Sebag, 1996] qui pour chaque exemple e (respectivement positif ou négatif) calcule son étoile entière, c'est-à-dire tous les sous-concepts corrects (respectivement vis-à-vis de A^+ ou A^-) incluant e.

Dans tous les cas, pour construire un paquet, il y aura $(|A^+| - 1)$ tentatives d'agrégation d'un exemple au paquet, c'est-à-dire autant de calculs de moindre généralisés et de vérification de correction : au pire, dans le cas où le concept est conjonctif, cela nécessitera $(|A^+| - 1).|A^-|$ tests de subsomption.

Explications

Évidemment, à moins qu'il y ait une solution conjonctive dans $\mathcal{L}_h$, le résultat dépend de l'ordre des exemples dans E : plusieurs paquets maximalement corrects peuvent être obtenus à partir d'une même graine s, mais en utilisant des ordres différents de E. Cependant, grâce à l'équation 6.1 (page 109), nous avons la garantie que chaque paquet maximalement correct est constructible par ajout d'un unique exemple à chaque étape, et en conservant à chaque fois la correction par rapport à A^-.

Pour compenser la dépendance vis-à-vis de l'ordre des exemples, GloBo va construire un paquet maximalement correct pour chaque exemple de A^+ : chacun des exemples positifs étant tour à tour utilisé comme graine et les autres exemples de A^+ étant présentés dans des ordres différents à chaque fois (ces listes sont mélangées aléatoirement).

Par rapport à AQ, ce point illustre la différence entre une approche par couverture et notre approche globale : AQ calcule uniquement les étoiles des exemples qui ne sont pas couverts par des étoiles déjà calculées alors que GloBo calcule un élément de l'étoile de chaque exemple positif de A^+, indépendamment les uns des autres.

graine	autres positifs	$\rightarrow$	paquet maximalement correct
p_1	$p_5\ p_8\ \not{p_2}\ p_{14} \ldots$	$\rightarrow$	$\{p_1, p_5, p_8, p_{14}, \ldots\}$
p_2	$\not{p_{14}}\ p_3\ \not{p_1}\ p_{12} \ldots$	$\rightarrow$	$\{p_2, p_3, p_{12}, \ldots\}$
p_3	$p_7\ \not{p_4}\ \not{p_{18}}\ \not{p_{12}} \ldots$	$\rightarrow$	$\{p_3, p_7, \ldots\}$
p_4	$p_{13}\ p_3\ \not{p_9}\ \not{p_{11}} \ldots$	$\rightarrow$	$\{p_4, p_{13}, p_3, \ldots\}$
p_5	$p_8\ \not{p_2}\ p_1\ p_{14} \ldots$	$\rightarrow$	$\{p_5, p_8, p_1, p_{14}, \ldots\}$
$\vdots$	$\ldots$	$\rightarrow$	$\ldots$

FIGURE 7.1 – Algorithme de formation des paquets

Puisque nous construisons un paquet pour chaque exemple positif, un sous-concept peut apparaître à chaque fois que l'on utilise un exemple de ce sous-concept comme graine. L'apprentissage échouera à caractériser le concept recherché si au moins l'un de ses sous-concepts n'apparaît pas parmi les paquets construits. La probabilité d'un tel événement sera évaluée à la section 7.2.2

Couverture minimale

Parmi tous les paquets obtenus, GloBo choisit ensuite les plus intéressants en construisant une couverture de A^+ par un nombre minimal de paquets. Comme nous l'avons déjà indiqué, ce problème est NP-difficile et c'est pourquoi nous utilisons l'heuristique qui consiste à privilégier les paquets de plus grande taille [Paschos, 1997].

Algorithme 7.3 (couverture minimale approchée)
Étant donnés E l'ensemble à couvrir et $\{C_i\}$ l'ensemble des paquets candidats.

1. *Soient noncouverts $= E$ et solution $= \emptyset$.*
2. *Tant que noncouverts $\neq \emptyset$ Faire*
   (a) *Soit C le paquet qui a la plus grande intersection avec noncouverts.*
   (b) *Les éléments de C sont effacés de noncouverts et C est ajouté à la solution.*
3. *Retourner solution.*

Le résultat de cet algorithme est montré Figure 7.2.

Comme tous les exemples de A^+ ont successivement été utilisés comme graine, il s'ensuit que la réunion des paquets C_i couvre A^+. Par suite, il existe une couverture de A^+ par les C_i et cet algorithme va terminer. Une couverture minimale approchée d'un ensemble de taille m par n ensembles, en utilisant cette méthode, a une complexité en $m.n$ [Paschos, 1997]. Ainsi, GloBo trouvera une couverture de A^+ avec une complexité au pire en $|A^+|^2$.

Algorithme principal

Algorithme 7.4 (GloBo)
Étant donné un algorithme pour calculer la caractéristation d'un paquet.

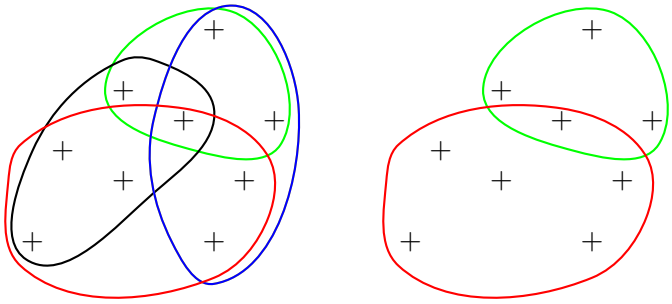


FIGURE 7.2 – Couverture minimale

1. *Pour chaque e_i de A^+ : calculer $P(e_i)$ un paquet maximalement correct par rapport à A^- avec e_i comme graine et $A^+ - \{e_i\}$ mélangé aléatoirement comme candidats (Algorithme 7.2).*
2. *Calculer une couverture minimale de A^+ par des $P(e_i)$ (Algorithme 7.3).*
3. *Calculer g_i la caractérisation de chaque $P(e_i)$ présent dans la couverture de A^+.*
4. *Retourner la disjonction des g_i.*

L'étape de construction d'un paquet (Algorithme 7.2) est itérée $|A^+|$ fois. Par conséquent, la complexité de notre algorithme est en $\Theta(|A^+|^2)$, en termes d'agrégations d'exemples à un paquet et de vérifications de correction contre A^- ; donc, la complexité au pire de GloBo est $\Theta(|A^+|^2.|A^-|)$ en tests de subsomption. À cette complexité, il faudrait ajouter le coût lié à la caractérisation d'un paquet. Notons qu'il n'y a pas de surcoût si nous utilisons le moindre généralisé d'un paquet comme son représentant car ce moindre généralisé est calculé en cours de formation des paquets.

7.2.2 Étude théorique

Discutons maintenant les conditions sous lesquelles GloBo va découvrir la solution attendue : pour simplifier le propos, nous supposerons que le concept est bien découvert si chacun de ses sous-concepts apparaît lors de la formation des paquets maximalement corrects. Nous faisons donc l'hypothèse que l'algorithme de couverture minimale sélectionnera bien ces paquets, et que le représentant choisi pour chaque paquet sera satisfaisant.

Remarquons tout d'abord que lorsque le concept à découvrir est conjonctif, GloBo fournira bien une solution conjonctive : quelle que soit la graine, l'algorithme 7.2 construit un paquet regroupant tous les exemples ; ainsi, ce paquet est le seul candidat fourni à l'algorithme 7.3.

Passons maintenant au cas plus intéressant, mais ô combien plus délicat, où le concept est réellement disjonctif. Intuitivement, pour que les sous-concepts soient découverts (naturellement, cette analyse ne peut être faite qu'a posteriori), les conditions suivantes doivent être vérifiées.

- Chacun des sous-concepts à découvrir doit être représenté par au moins (mais de préférence plus) un exemple de A^+.
- Dans le but de construire le paquet approprié à partir d'une graine, les premiers candidats à l'agrégation doivent *préserver* le paquet. Pour ce faire, il faut
  - que chacun de ces candidats appartiennent au même sous-concept que la graine s,
  - ou bien que ces candidats e_i soient fortement incompatibles avec s, c'est-à-dire que le moindre généralisé de s et d'un tel candidat e_i couvre des exemples de A^- (dans ce cas, ces exemples ne peuvent tout simplement pas entrer dans le paquet).

Les exemples e_i qui peuvent poser problème sont donc ceux qui n'appartiennent pas au sous-concept de la graine s mais qui sont pourtant compatibles avec s.

— Soit que leur moindre généralisé est trop spécifique et ne peut couvrir aucun négatif, c'est le cas des exemples suivants :

×	○	×
○	×	
○		×

×	○	×
○	×	
×		○

qui se généralise en

×	○	×
○	×	
?		?

Cette hypothèse ne peut pas couvrir une situation de victoire des ronds car il n'y a aucun moyen de former une ligne de ronds en instanciant les cases ?, ni une situation de partie nulle puisqu'il reste des cases vides. Ces deux exemples, bien qu'ils appartiennent à des sous-concepts différents, ne sont pas séparables.

— Soit qu'il peut exister un exemple négatif séparant s et e_i mais qu'un tel exemple n'est pas disponible dans A^-.

Ce risque n'intervient qu'au début de la formation d'un paquet. L'idée est que tous les paquets de petite taille sont corrects, mais au fur et à mesure que le moindre généralisé du paquet se rapproche de la définition du sous-concept, il devient de plus en plus difficile pour les exemples étrangers à ce sous-concept d'entrer dans le paquet. Lorsque l'entrée de ces exemples devient strictement impossible, nous dirons que le paquet est *verrouillé*. Si malheureusement un exemple étranger au sous-concept a pu entrer dans le paquet avant qu'il ne soit verrouillé, nous dirons que le paquet est condamné. Ces deux notions sont illustrées Figure 7.3.

Nous allons formaliser ces intuitions en estimant la probabilité de formation des sous-concepts. Pour cela, nous introduisons les notations suivantes :

— n est le nombre de sous-concepts à apprendre,

— s est la taille moyenne d'un sous-concept dans A^+, c'est-à-dire le nombre moyen de représentants d'un sous-concept.

— α la probabilité pour qu'un exemple de A^+ soit favorable à la graine (soit qu'il appartienne au même sous-concept, soit qu'il soit fortement incompatible avec elle),

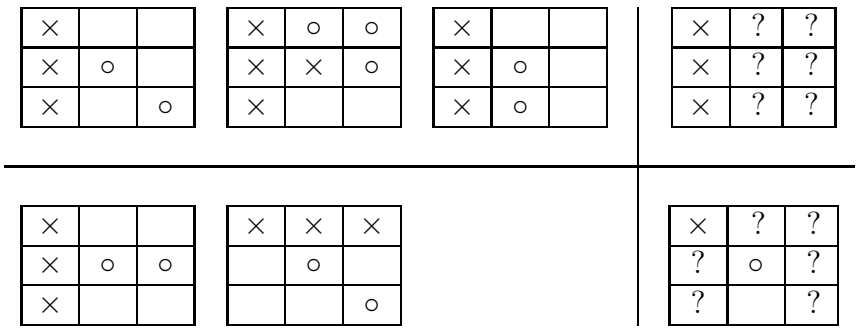


FIGURE 7.3 – Paquet verrouillé et paquet condamné

— et b la taille minimale d'un paquet interdisant l'arrivée d'exemples inappropriés (quand b éléments du sous-concept sont dans un paquet, celui-ci est considéré comme verrouillé et aucun exemple étranger au sous-concept ne peut plus entrer).

Il est maintenant possible d'évaluer la probabilité d'événements clefs.

— Commençons par l'événement *verrouiller un paquet* : la séquence des candidats à l'ajout doit commencer par b exemples favorables à la graine, d'où une probabilité de α^b.

— Même si nous échouons à verrouiller un paquet sur le véritable sous-concept de la graine, nous avons le droit à s tentatives. L'échec dans la formation d'un sous-concept ne sera vraiment consommé que si nous échouons s fois, d'où une probabilité de ne pas former un sous-concept de $\left(1-\alpha^b\right)^s$.

— Enfin la probabilité de réussir à former les n sous-concepts est de $\left[1-\left(1-\alpha^b\right)^s\right]^n$.

Finalement, nous définissons la probabilité de réussite de l'apprentissage :

$$\left[1-\left(1-\alpha^b\right)^s\right]^n \tag{7.1}$$

Les cas où cette probabilité prend des valeurs élevées caractérisent les situations où GloBo se comportera bien : quand le nombre de sous-concepts à découvrir sera petit devant le nombre d'exemples disponibles pour représenter chaque sous-concept.

Dans un cas contraire, soit il y a des sous-concepts qui sont peu ou non représentés dans A^+, soit les exemples de A^- ne permettent pas de séparer les sous-concepts. Nous dirons alors que A^+ et A^- ne sont que *faiblement représentatifs* des sous-concepts attendus et il est probable que GloBo ne parvienne pas à les découvrir.

7.3 Expérimentations

7.3.1 Problème du morpion

Nous étudierons de façon approfondie le comportement de GloBo sur la fin de jeu du morpion, problème qui nous a servi de fil conducteur au chapitre précédent, section 6.1.

GloBo a été comparé à `C4.5`, `C4.5rules` [Quinlan, 1993], et `CN2` [Clark et Niblett, 1989]. Nous avons tout d'abord utilisé le test `5x2cv` [Dietterich, 1998], décrit en annexe (section A.2), pour déterminer la confiance avec laquelle on peut affirmer que GloBo est meilleur que les autres systèmes sur cette tâche. Les résultats obtenus sont donnés à la Table 7.1 : sur le problème du morpion, GloBo doit être considéré comme le meilleur système de ceux considérés.

Ensuite, nous avons voulu tester chaque système en fonction du nombre d'exemples disponibles et déterminer le nombre d'exemples nécessaires à GloBo pour apprendre la définition attendue. En appliquant au morpion l'Équation 7.1 (page 120), la probabilité de succès de GloBo en fonction de la taille des données utilisées est donnée à la Figure 7.4. Pour tracer cette courbe, nous avons mesuré sur le jeu de données

TABLE 7.1 – Test 5×2cv (GloBo)

	statistique $\tilde{t}$	Confiance
CN2	-2.215	93.00%
C4.5rules	-4.420	99.40%
C4.5	-7.836	$> 99.99\%$

la valeur des différents paramètres définis précédemment :

$$\begin{aligned} n &= 8 \\ s &= \tfrac{1}{8}.|A^+| \\ \alpha &= \tfrac{1}{2} \\ b &= 2 \end{aligned}$$

À la Figure 7.5, nous donnons les prédictions moyennes obtenues sur 10 tirages pour des ensembles d'apprentissage dont la taille varie de 5 à 95% de la taille de l'ensemble original.

Conformément aux estimations données, GloBo obtient de très bonnes prédictions en n'utilisant que 20% de la base initiale. Dès que les exemples sont en nombre suffisant, GloBo atteint rapidement 100% de bonnes prédictions et est ensuite parfaitement stable. Précisons enfin que la solution découverte par GloBo, correspond dans tous les cas à la disjonction des huit manières de faire une ligne avec des croix au morpion.

7.3.2 Autres problèmes

Nous avons testé GloBo sur d'autres problèmes provenant de la base UCI [Merz et Murphy, 1996], en particulier sur des problèmes avec des attributs continus.

À chaque fois, GloBo obtient de bons résultats même si, selon le test `5x2cv`, GloBo n'est pas toujours significativement meilleur que les autres systèmes. Cependant, nous avons systématiquement observé que GloBo trouve des solutions plus courtes en nombre de sous-concepts que les solutions déjà connues.

La Table 7.2 donne, pour des domaines connus, les prédictions moyennes et les tailles minimales des solutions découvertes par GloBo.

Pima Indians Diabetes apparaît comme une tâche terriblement difficile, puisque GloBo ne parvient pas à diviser les 200 exemples positifs disponibles en moins de 24

TABLE 7.2 – Autres résultats de GloBo

Problème	Taille	Prédiction
Mushroom (avec négation)	3	100.0 %
Breast Cancer	4	93.0 %
Pima Indians Diabetes	24	68.8 %
Echocardiogram	3	88.2 %

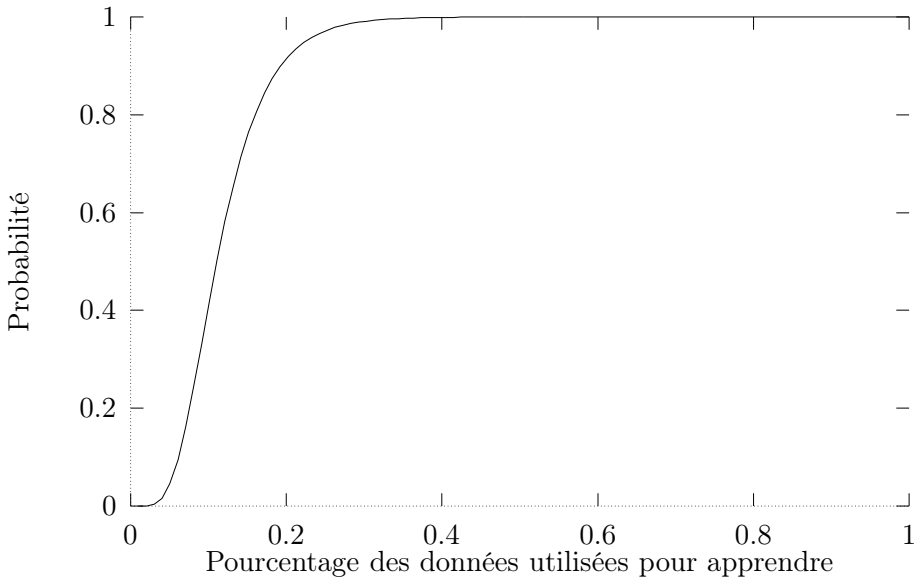


FIGURE 7.4 – Probabilité de succès pour le problème du morpion

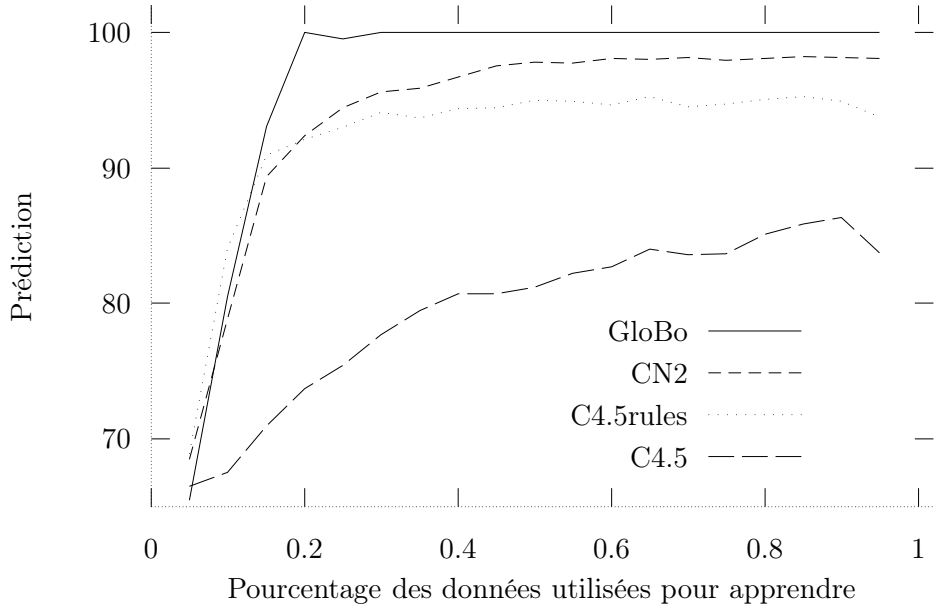


FIGURE 7.5 – Prédictions pour le problème du morpion

paquets ! Ce problème peut donc être considéré comme *hautement disjonctif*. Nous nous trouvons là dans un cas où tous les sous-concepts sont des *small disjuncts* ! Tout cela explique sans doute pourquoi aucun système ne parvient à de bons résultats sur ce problème.

En revanche, GloBo obtient une excellente prédiction sur le problème *Echocardiogram*, problème où tous les attributs sont continus.

7.3.3 Problème multi-instances

Dans un problème *multi-instances* [Dietterich et al., 1997], chaque exemple est représenté par plusieurs instances : une solution doit couvrir au moins une instance de chaque exemple positif, et aucune de chaque exemple négatif.

GloBo peut traiter le multi-instances sans modification importante : nous faisons des paquets d'*instances* d'exemples positifs, puis on cherche la couverture des *exemples* positifs. Ainsi, la solution découverte couvrira au moins une instance de chaque exemple positif.

GloBo obtient à 80% de bonnes prédictions sur le data set *musk*, ce qui n'est pas le meilleur résultat connu puisque Dietterich annonce 89%. Malgré cela, et compte tenu du fait que GloBo n'a pas été spécifiquement conçu pour le multi-instances, nous estimons que la précision obtenue par GloBo constitue un résultat très honorable.

7.3.4 PTE Challenge

Enfin, GloBo a participé au *PTE challenge* [Srinivasan et al., 1997] que nous avons présenté section 1.8 (page 19).

Il s'agit d'un problème de Programmation Logique Inductive. Cependant, nous avons travaillé dans un langage attribut-valeur, certains attributs codant la valeur, vrai ou faux, d'une clause découverte par d'autres systèmes PLI et fournit dans le cadre de la compétition.

Un exemple de règle apprise par GloBo sur ce problème est donné à la Table 7.3.

Les résultats de la compétition, présentés dans [Srinivasan et al., 1999], indiquent que GloBo obtient la meilleure précision parmi tous les systèmes produisant une théorie compréhensible, et qu'il est de ce point de vue deuxième tout type de système confondu.

Cependant, il s'agit d'un problème où il est illusoire de chercher une définition qui soit correcte et complète ; autrement dit une précision de 100 % est inaccessible. La solution par défaut, indiquant qu'aucune molécule n'est cancérigène, possède une précision de 70%. Mais, bien sûr, elle n'est pas très satisfaisante du point de vue de la santé publique.

Comme dans beaucoup de problèmes où les données ne sont pas équi-distribuées, il faut se résoudre ici à ne pas considérer la précision comme un critère absolu. La démarche consiste à reconnaître le maximum de molécules cancérigènes, même si pour cela on risque de déclarer des molécules cancérigènes alors qu'elles ne le sont pas. Autrement dit, en reprenant les termes définis à la section 1.6 (page 14), nous cherchons une bonne *sensibilité*, éventuellement au prix d'une *spécificité* médiocre.

TABLE 7.3 – Exemple de règle apprise par GloBo pour le PTE-challenge

```
AT24 is lower or equal to 2.5
AT26 ranges from 0.5 to 17.0
AT27 is 0.0
AT29 is 0.0
AT33 is 0.0
AT36 is lower or equal to 1.5
AT37 is 0.0
AT52 is 0.0
AT59 is 0.0
W7 is false
W21 is false
N11 is 0.0
N13 is lower or equal to 4.0
N20 is lower or equal to 3.0
A1 is 0.0
A22 is 0.0
A23 is 0.0
G4 is positive

Positive covered in the training set: 55 / 182
Negative covered in the training set: 0 / 155

Prediction:

100-41-4      t7      (carcinogenic)
77-09-8       t15     (carcinogenic)
125-33-7      t19     (carcinogenic)
7632-00-0     t28     (unknown class)
```

Déterminer le meilleur compromis est une tâche en général laissée à la charge des experts. Pour cela, nous utilisons les travaux de [Provost et Fawcett, 1997]. Il s'agit de fournir les différents couples *(1-spécificité,sensibilité)* obtenus en faisant varier les paramètres du système à travers une *courbe ROC* (*Receiver Operating Characteristic*) dont un exemple est donné à la Figure 7.6. Pour tracer cette courbe, il ne faut considérer que les meilleurs points : un point n'appartiendra à la courbe que si aucun autre point ne lui est supérieur ni du point de vue de la sensibilité ni du point de la spécificité. Ainsi, les points de la courbe ROC sont incomparables.

Dans le cas du *PTE challenge*, [Srinivasan et al., 1999] a tracé une telle courbe, non pas en faisant varier les paramètres d'un même système mais en représentant chaque système en compétition par un point, seuls les meilleurs systèmes apparaissant sur la courbe ROC. Or, GloBo se trouve sur cette courbe.

[Srinivasan et al., 1999] pousse l'analyse plus loin en faisant intervenir :

— la répartition des classes à travers les probabilités, $\pi(+)$ et $\pi(-)$, d'appartenir à chacune des deux classes ;

— le coût d'une erreur de classification ; $C(+|-)$ est le coût induit par l'étiquetage positif d'un exemple qui est en réalité négatif et $C(-|+)$ le coût de l'erreur inverse.

Ces informations définissent des droites sur lesquelles tous les points ont le même coût. La pente de ces droites vaut

$$m = \frac{\pi(-).C(+|-)}{\pi(+).C(-|+)}$$

Les classifieurs intéressants se trouvent sur une portion de la courbe ROC ayant pour tangente une droite de pente m : on dit qu'ils sont FAPP-optimaux (*optimal for all practical purposes*).

Lorsque les informations sur la répartition des exemples et le coût des erreurs sont imprécises, ce qui est le cas ici, il n'est plus possible de donner une valeur précise à m mais la notion de FAPP-optimalité est conservée.

Dans le cas du PTE Challenge, la répartition des classes est estimée par les probabilités suivantes :

$$\pi(+) \in [0.5; 0.7]$$
$$\pi(+) \in [0.3; 0.5]$$

et les coûts des erreurs à :

$$\frac{C(-|+)}{C(+|-)} \in [10; 20]$$

et, par suite, $m \in [0.02; 0.1]$. Sous ces hypothèses, [Srinivasan et al., 1999] conclut que GloBo est le seul des compétiteurs à être FAPP-optimal.

7.4 Schéma générique

Nous avons vu que le système GloBo obtenait des résultats dignes d'intérêt sur des problèmes délicats. Cependant, sa complexité cubique est une barrière à l'utilisation de GloBo sur des bases de très grande taille. Cette pourquoi nous proposons maintenant une version linéaire de GloBo. Les modifications possibles des algorithmes de GloBo étant nombreuses, nous serons amenés à en donner une version

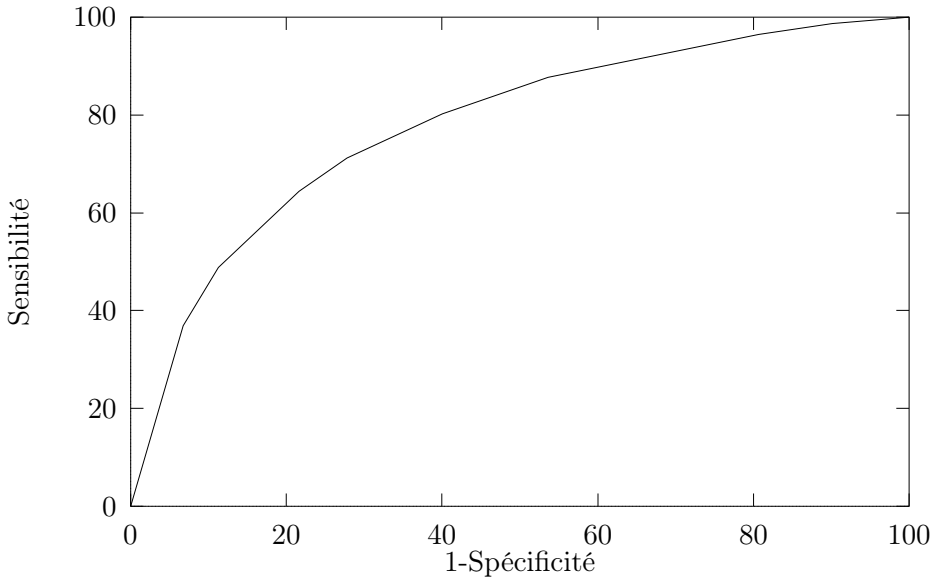


FIGURE 7.6 – Exemple de courbe ROC

générique. Nous verrons ensuite deux instanciations : l'une pour le non supervisé, l'autre pour traiter les données bruitées.

7.4.1 Version linéaire

GloBo utilise les ensembles d'exemples à trois niveaux, comme l'indique la complexité cubique de notre algorithme.

1. A^+ est l'ensemble des graines utilisées pour amorcer les paquets ;
2. pour chaque graine, nous parcourons A^+ pour sélectionner les positifs qui vont constituer le nouveau paquet ;
3. à chaque tentative d'ajout, il faut valider le nouveau paquet, en testant sa correction vis-à-vis de A^-.

À chacun de ces niveaux, nous pouvons remplacer les ensembles d'exemples considérés par un échantillonage. Reprenons un par un ces niveaux.

1. Seuls k_g exemples positifs sont utilisés comme graine ($k_g \leq |A^+|$). Nous comptons ici sur le fait qu'un sous-concept peut apparaître à partir d'autant de graines qu'il y a d'exemples dans ce sous-concept. Il n'est donc pas utile que tous les exemples de A^+ deviennent des graines.
2. Seuls k_a exemples positifs sont candidats à l'ajout dans un paquet ($k_a \leq |A^+|$). Cela se justifie par le fait qu'un paquet peut être verrouillé ou condamné bien avant que tous les exemples de A^+ n'aient été considérés.
3. Seuls k_c exemples négatifs sont considérés pour établir la correction d'un ajout ($k_c \leq |A^-|$). Nous espérons ici qu'une mauvaise association d'exemples provoque un moindre généralisé très général et couvre plusieurs exemples négatifs. Il suffit de capturer un de ces exemples parmi les k_c choisis pour empêcher une mauvaise association.

Notons que, pour chaque tentative dans la construction d'un paquet, ce sont des positifs différents qui sont candidats à l'ajout et que les négatifs, qui permettent de tester la correction des ajouts, sont eux aussi différents. Ainsi, bien que le nombre d'exemples utilisés soit borné à chaque niveau, tous les exemples disponibles peuvent intervenir au cours de l'apprentissage.

Cela nous fournit un apprentissage en temps constant : nous avons $k_g.k_a$ calculs de moindres généralisés et, au pire, $k_g.k_a.k_c$ tests de subsomption. Naturellement, le gain n'est réel que si k_g, k_a et k_c sont petits devant le nombre d'exemples dans A^+ et A^-. Sinon, malgré la complexité constante, le temps d'exécution sera comparable à celui nécessaire à la version première de GloBo.

Ce faisant, nous perdons la garantie que nous avions sur la complétude et la correction de la solution.

— La complétude peut être perdue si tous les sous-concepts à découvrir ne sont pas représentés par les k_g graines choisies. k_g doit être au moins égal à la taille de la disjonction recherchée. Naturellement, cette valeur est inconnue et elle ne garantirait de toute façon pas la complétude.

— Une solution incorrecte peut aussi être construite par notre algorithme constant : nous avons pu construire un paquet ne couvrant aucun des k_c exemples négatifs mais incorrect par rapport à A^- entier.

Pour retrouver la propriété de correction vis-à-vis de A^-, nous allons sélectionner les paquets qui vont être candidats à la couverture, pour ne garder que ceux qui sont corrects contre *tous* les exemples négatifs disponibles. En revenant à un test de correction contre A^- entier, nous obtenons une méthode de complexité linéaire.

Cet apprentissage sera t-il équivalent à celui opéré par la version exhaustive de GloBo ? Étonnamment, des résultats récents sur l'échantillonnage des bases d'exemples vont dans ce sens [Brézellec, 1999, Amoura et al., 1999, Provost et al., 1999] : ces travaux indiquent que, pour la plupart des gros problèmes de l'UCI [Merz et Murphy, 1996], la précision obtenue en utilisant seulement 30 ou 40% de la base d'exemples est comparable à celle obtenue avec 70 % des exemples.

Il nous reste à fixer effectivement les valeurs de k_g, k_a et k_c. Comment trouver des valeurs qui soient les plus petites possibles et qui autorisent un apprentissage satisfaisant ?

k_a peut être identifié au nombre d'exemples nécessaires pour verrouiller nombre de sous-concepts. Les autres paramètres nécessitent une connaissance du problème (par exemple, un ordre de grandeur pour le nombre de sous-concepts), qui peut être obtenue par une exécution de Globo dans sa version cubique. Dans le cas du morpion, nous nous sommes inspirés de la courbe de la Figure 7.5 : puisque 25% des données permettent la découverte de la solution, nous pouvons nous contenter d'apprendre sur cette portion de données, même si elles sont disponibles en plus grand nombre. Finalement, nous avons utilisé les valeurs suivantes :

$$k_g = 150$$
$$k_a = 25$$
$$k_c = 80$$

Avec ces valeurs, la version linéaire de GloBo obtient les mêmes précisions que celles obtenues par la version cubique : la Figure 7.5 présente donc les résultats de

ces deux versions.

Et, naturellement, nous observons à la Figure 7.7 que les temps d'exécution évoluent de manière linéaire avec la taille de l'ensemble d'apprentissage. Il y a tout de même une portion de courbe cubique, au début, c'est-à-dire avant que le nombre d'exemples positifs ne dépassent k_g et k_a, et que le nombre de négatifs ne dépassent k_c.

GloBo dans sa version cubique demandait environ une heure de temps de calcul pour traiter les 100% de données disponibles pour le morpion. Sur la même machine, la version linéaire de GloBo traite le même problème, en obtenant les mêmes résultats, en moins de 25 secondes.

7.4.2 Version générique

Nous venons de voir qu'en appliquant des heuristiques stochastiques, nous pouvions passer d'une complexité cubique à une complexité linéaire.

Comme l'éventail de tels changements est très large, nous sommes conduits à définir la forme générique de GloBo : toutes les opérations devront être instanciées pour obtenir un système d'apprentissage.

Cet algorithme générique est donné à l'algorithme 7.5 tandis que la table 7.4 énumère les opérations génériques à instancier. Enfin, les tables 7.5 et 7.6 donnent les instanciations de ces opérations respectivement pour la version cubique et la version linéaire de GloBo.

Algorithme 7.5 (GloBo générique)

1. *Tant qu'un nouveau paquet est nécessaire :*
   (a) *Choisir une graine.*
   (b) *Choisir les exemples positifs candidats à l'ajout.*
   (c) *Mélanger aléatoirement ces candidats.*
   (d) *Choisir les exemples négatifs qui seront considérés lors de la construction du paquet.*
   (e) *Parcourir les candidats en ajoutant ceux qui préservent une certaine notion de correction vis-à-vis des négatifs sélectionnés.*
   (f) *Ajouter éventuellement certains positifs non candidats lors de l'ajout précédent.*
2. *Parmi les paquets obtenus, sélectionner ceux qui seront autorisés à participer à la couverture des exemples positifs.*
3. *Étant donné un test caractérisant une couverture satisfaisante, calculer la couverture minimale de A^+ par ces paquets.*
4. *Caractériser chaque paquet de la couverture par une hypothèse de $\mathcal{L}_h$.*

Les instanciations sont variées. Considérons quelques variations sur le choix de la graine. Nous pouvons tendre à la complétude en choisissant comme graine un exemple positif qui n'appartient encore à aucun paquet construit, à la manière de AQ. Si k_g est supérieur au nombre de sous-concepts à isoler, cette stratégie garantit

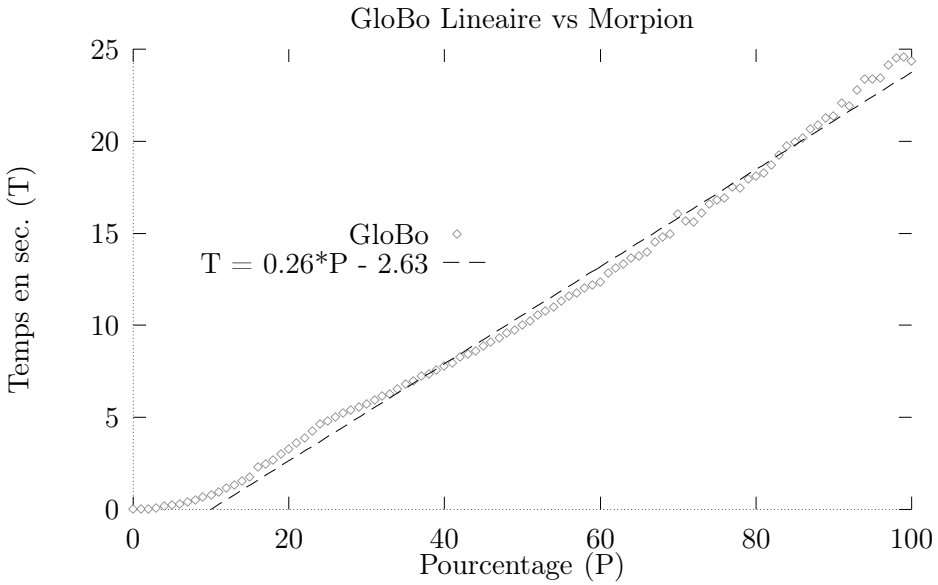


FIGURE 7.7 – Linéarité expérimentale

TABLE 7.4 – Opérations génériques de GloBo

Opérations	Références
Condition d'arrêt	1
Choix de la graine	1a
Choix des positifs	1b
Choix des négatifs	1d
Condition d'ajout	1e
Condition d'ajout subsidiaire	1f
Sélection des paquets	2
Couverture satisfaisante	3
Caractérisation conjonctive	4

TABLE 7.5 – Instanciation pour un algorithme cubique

Opérations	Instanciations
Condition d'arrêt	tous les positifs ont été utilisés comme graine
Choix de la graine	un positif n'ayant pas déjà servi de graine
Choix des positifs	prendre A^+ entier
Choix des négatifs	prendre A^- entier
Condition d'ajout	correction vis-à-vis de A^- entier
Condition d'ajout subsidiaire	pas d'ajout subsidiaire
Sélection des paquets	prendre tous les paquets construits
Couverture satisfaisante	tous les positifs sont couverts
Caractérisation conjonctive	calculer le moindre généralisé du paquet

TABLE 7.6 – Instanciation pour un algorithme linéaire

Opérations	Instanciations
Condition d'arrêt	k_g positifs ont été utilisés comme graine
Choix de la graine	tirage aléatoire parmi les positifs n'ayant pas déjà servi de graine
Choix des positifs	tirer aléatoirement k_a exemples dans A^+
Choix des négatifs	tirer aléatoirement k_c exemples dans A^-
Condition d'ajout	correction vis-à-vis des k_c négatifs
Condition d'ajout subsidiaire	ajouter l'exemple s'il est subsumé par le moindre généralisé du paquet
Sélection des paquets	prendre les paquets corrects vis-à-vis de A^- entier
Couverture satisfaisante	tous les positifs sont couverts
Caractérisation conjonctive	calculer le moindre généralisé du paquet

à nouveau la complétude, sans surcoût de complexité. Plus généralement, les positifs peuvent être ordonnés selon qu'ils correspondent à des graines intéressantes ou non. Une graine intéressante sera un positif non couvert comme nous l'avons dit, ou un positif n'intervenant que dans des paquets peu fiables (au sens de la mesure de qualité donnée par l'équation 6.1). Dans le cas d'un exemple impliqué dans un *small disjunct*, nous pourrons utiliser plusieurs fois cet exemple positif comme graine, produisant ainsi plusieurs paquets possibles et laissant la procédure de couverture minimale choisir le plus pertinent par rapport à la couverture de A^+.

L'algorithme de GloBo peut donc être décliné de très nombreuses manières. Nous allons maintenant voir plus dans le détail, deux adaptations possibles : l'une pour le non supervisé, l'autre pour l'apprentissage supervisé à partir de données bruitées.

7.4.3 GloBo pour le non supervisé

Instanciation

Nous allons voir ici que GloBo peut accomplir un apprentissage non supervisé en remplaçant la correction par rapport à A^- par une notion de spécificité des paquets. Étant donnés une distance d et E l'ensemble des instances disponibles, un paquet est satisfaisant si pour chaque paire d'instances de ce paquet, la distance les séparant ne dépasse pas une valeur limite :

$$\text{satisfaisant}(P) \Leftrightarrow \forall (a,b) \in P, d(a,b) \leq d_{max}$$

Nous nommerons *diamètre* la distance maximale entre deux éléments d'un même paquet. Notre notion de paquet satisfaisant est donc le respect d'une borne par ce diamètre.

Par souci de simplicité, nous ne décrivons ici que l'instanciation de GloBo par cette propriété mais toute fonction d'évaluation des classes vérifiant la relation de l'équation 6.1 (page 109) peut être utilisée ici.

Pour ajouter un élément à un paquet, nous avons à calculer les distances entre le nouvel élément et toutes les instances présentes dans le paquet courant (dans le pire des cas, $|E|-1$ distances à calculer). Le paquet ainsi formé est admissible si aucune de ces distances ne dépasse la valeur limite. Finalement, la complexité de GloBo pour un apprentissage non supervisé est $\Theta(|E|^2)$ en termes de calculs de distance. La Table 7.7 présente l'instanciation de GloBo pour ce type d'apprentissage.

Réinterprétons la formule 7.1 (page 120) dans ce cadre. Pour faciliter la découverte des sous-concepts attendus, GloBo a besoin d'instances en plus grand nombre possible pour chaque sous-concept, et que le diamètre soit limité à la valeur minimale autorisant l'apparition des paquets attendus (cela pour verrouiller au plus tôt ces paquets).

À ce stade, il est naturel de penser que la difficulté majeure est de correctement ajuster la limite sur le diamètre d'un paquet, mais nous verrons dans la suite que ce réglage peut être facilement réalisé.

Comme pour l'apprentissage supervisé, nous nous sommes seulement intéressés à la construction de paquets pertinents, et non pas par leurs caractérisations : on pourra par exemple utiliser les moindre généralisés des paquets, ou se servir de GloBo

TABLE 7.7 – Instanciation pour le non supervisé

Opérations	Instanciations
Condition d'arrêt	tous les exemples de E ont été utilisés comme graine
Choix de la graine	un exemple n'ayant pas déjà servi de graine
Choix des positifs	prendre E entier
Choix des négatifs	-
Condition d'ajout	respect du diamètre maximal
Condition d'ajout subsidiaire	pas d'ajout subsidiaire
Sélection des paquets	prendre tous les paquets construits
Couverture satisfaisante	tout E est couvert
Caractérisation conjonctive	calculer le moindre généralisé du paquet

dans sa version supervisée sur chacun des paquets (les autres paquets servant alors d'exemples négatifs).

Expérimentations

Nous avons utilisé ici le problème *Quadruped Animals* décrit dans [Gennari et al., 1989] concernant les mammifères à quatre pattes. Chaque instance appartient à l'une des quatre classes suivantes : chiens, chats, chevaux ou girafes. Chaque animal est décrit par 8 composantes : le cou, quatre jambes, le tronc, la tête et la queue. Chacun de ces éléments est représenté par un cylindre, lui-même défini par 9 attributs. Pour nos expérimentations, nous avons généré 100 instances en utilisant le générateur mis à disposition sur le site UCI [Merz et Murphy, 1996]. Enfin, précisons que nous avons simplement utilisé la distance euclidienne.

Le but est de minimiser à la fois le nombre de paquets et le diamètre des paquets. Ces deux optimisations sont incompatibles : pour un diamètre-limite faible, chaque élément est dans un paquet différent et le nombre de paquets est alors élevé. Inversement, si nous utilisons un seul paquet pour rassembler tous les éléments, le diamètre de ce paquet est maximal.

Par suite, notre compromis a été de travailler sur la quantité suivante : le produit du nombre de paquets par le diamètre *observé*. Le tracé de cette valeur en fonction du diamètre *autorisé* fait apparaître des plateaux (Figure 7.8).

Durant ces plateaux, le diamètre autorisé n'est pas pleinement utilisé : son augmentation ne modifie ni le nombre de paquets ni le diamètre observé. Intuitivement, on peut penser que pendant ces plateaux, GloBo traverse les espaces inter-paquets. Dans le but de valider cette idée, observons les sous-concepts formés pour des distances limites choisies dans ces plateaux.

Si nous choisissons un diamètre dans le premier plateau de la courbe (qui correspond au point le plus bas de la courbe), c'est-à-dire entre 23 et 29, quatre paquets sont formés qui correspondent parfaitement aux classes originales. Sur le deuxième

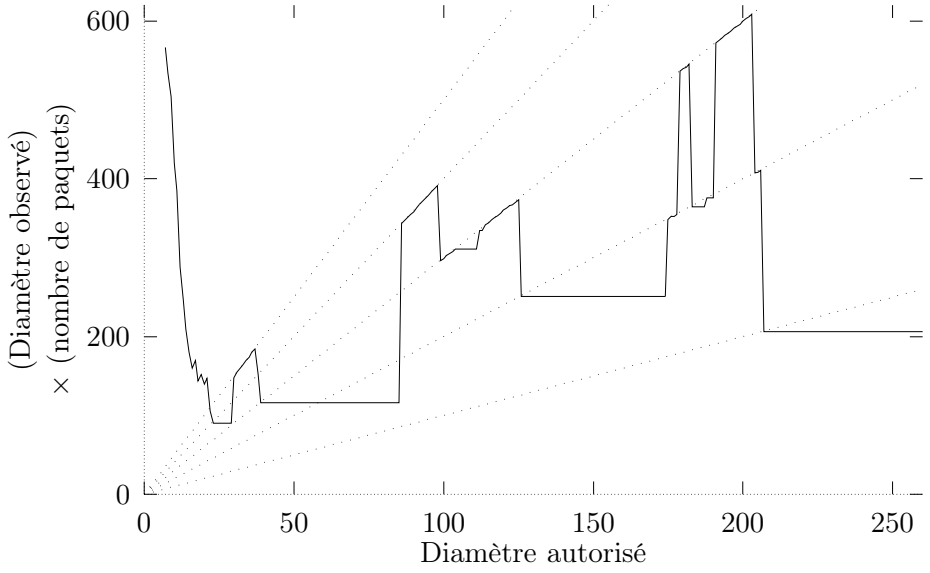


FIGURE 7.8 – Réglage empirique de la distance limite. Les droites en pointillés correspondent aux droites d'équation $y = kx$ pour $k = 1, 2, 3, 4, 5$.

plateau (39-85), GloBo trouve trois paquets : les chiens et les chats se sont rassemblés, les chevaux et les girafes forment toujours des paquets disjoints. Sur le troisième plateau de la courbe (126-174), il ne reste que deux paquets : les girafes ont rejoint les chevaux. Enfin, le dernier plateau (à partir de 207) correspond au rassemblement de toutes les instances dans un unique paquet.

Ainsi, en combinant les sous-concepts appris sur les différents plateaux de la courbe, on peut construire une hiérarchie qui utilise les sous-concepts appris sur le plateau le plus bas comme concepts de base.

7.4.4 GloBo contre le bruit

Enfin, des versions de GloBo traitant les données bruitées sont envisageables. Deux cas sont à considérer : les exemples présents dans A^+ qui sont en réalité des contre-exemples (des positifs bruités) et les exemples présents dans A^- qui sont en réalité des exemples du concept à apprendre (des négatifs bruités).

— Pour résister aux négatifs bruités, l'affaiblissement de la condition d'ajout permettra aux généralisations des paquets de les couvrir. *Le moindre généralisé du paquet ne doit couvrir aucun exemple négatif* est affaibli en *le moindre généralisé du paquet couvre au plus η exemples négatifs* où η doit être fixé. Il est même envisageable de repérer ces négatifs bruités lors de la sélection des paquets, avant le calcul de la couverture : les exemples négatifs le plus souvent couverts pourront être supposés bruités, tandis que les paquets couvrant d'autres négatifs que ceux-là seront écartés.

— Quand aux positifs bruités, nous les repérerons au moment du calcul de la couverture minimale : la couverture sera jugée satisfaisante lorsque le meilleur nouveau paquet ne permettra de couvrir qu'un ou deux nouveaux exemples.

Nous supposons en cela que les exemples bruités ne peuvent s'associer dans un même paquet sans couvrir de négatifs et que les véritables positifs présents dans le paquet vérolé auront été couverts par ailleurs. Sous cette hypothèse, le taux de positifs bruités n'a pas besoin d'être connu a priori.

L'instanciation de GloBo reprenant ces modifications est donnée à la Table 7.8.

7.5 Conclusion

7.5.1 Bilan

Nous avons dans ce chapitre décrit le système GloBo qui propose une solution originale au problème d'apprentissage supervisé disjonctif que nous avons exposé au chapitre précédent.

Il est remarquable que GloBo parvienne à trouver de très bonnes solutions, à ce problème NP-difficile, en particulier en terme de taille minimale, et en temps polynomial. Cela est rendu possible par deux heuristiques :

1. La première réside dans le caractère aléatoire de la formation des paquets maximalement corrects. Cette heuristique est dirigée par les ensembles d'exemples A^+ et A^- et s'adapte donc à chaque nouveau problème. Le résultat est que GloBo forme les paquets les plus probables selon A^+ et A^-. Or, si ces ensembles sont représentatifs de ce qu'il faut apprendre, alors les paquets les plus probables sont précisément ceux qui nous intéressent.
2. La seconde heuristique consiste à favoriser les paquets de plus grande taille durant la couverture minimale approchée. Outre le gain de complexité apporté par cette heuristique, nous estimons que le plus gros paquet qui a pu se former durant la première phase est sans doute un sous-concept intéressant. Et à nouveau, les cas où elle pourrait être mise en défaut, par exemple si le plus grand paquet construit n'est pas un sous-concept recherché, sont des cas marginaux où les données ne sont pas représentatives de ce qu'il faut apprendre.

Cette heuristique offre aussi une réponse au problème des *small disjuncts* (section 6.3.2, page 105) : une fois les sous-concepts de plus grande taille, il est aisé de repérer les plus petits. En revanche, le problème reste entier lorsque le concept à découvrir est composé d'une multitude de petits sous-concepts (nous avons vu un tel cas avec le problème *Pima Indians Diabetes*, section 7.3.2, page 121).

Enfin, nous avons vu que GloBo autorisait beaucoup de variations et en avons détaillées trois : l'une de complexité linéaire, une autre pour le non supervisé et une dernière pour traiter les données bruitées.

7.5.2 Notion de représentativité

La notion de représentativité que nous avons définie à travers la formule 7.1 (page 120), ainsi que l'interprétation que nous en avons donnée rappelle l'*apprentissage à la limite* de Gold [Gold, 1967].

TABLE 7.8 – Instanciation contre le bruit

Opérations	Instanciations
Condition d'arrêt	tous les positifs ont été utilisés comme graine
Choix de la graine	un positif n'ayant pas déjà servi de graine
Choix des positifs	prendre A^+ entier
Choix des négatifs	prendre A^- entier
Condition d'ajout	le paquet ne couvre pas plus η exemples de A^-
Condition d'ajout subsidiaire	pas d'ajout subsidiaire
Sélection des paquets	prendre tous les paquets construits
Couverture satisfaisante	le meilleur paquet ne couvre pas plus de un nouvel exemple de A^+
Caractérisation conjonctive	calculer le moindre généralisé du paquet

Dans ce cadre, les exemples arrivent incrémentalement, à chaque nouvel exemple ou contre-exemple, le système d'apprentissage doit proposer une définition pour le concept. Au bout d'un certain temps, le système doit proposer la solution attendue et ne plus changer d'avis ensuite, quels que soient les nouveaux exemples présentés. lesquels une telle garantie existe sont dits apprenables à la limite.

Autrement dit, cette notion d'apprenabilité exige qu'à partir d'un ensemble de données assez grand, mais sans que cette taille soit bornée, ces données désignent à la méthode la définition à apprendre et seulement cette définition. C'est la même préoccupation que nous retrouvons dans notre notion de représentivité à ceci près qu'elle opère sur un ensemble d'apprentissage qui pour nous est immuable puisque c'est là toutes les données dont nous pouvons disposer : nous voulons que ces données soient telles que la définition cherchée ait la plus grande probabilité d'être apprise par GloBo.

Cette mesure de représentativité a un autre intérêt, cette fois lié au *Selective Superiority Problem* que nous avons évoqué section 1.7 (page 18) : nous avions vu que pour chaque gain obtenu dans une situation donnée, il existe une autre situation où un gain inverse sera obtenu. Naturellement, ce résultat s'applique également à GloBo. Cependant, nous avons identifié les situations dans lesquelles les mauvais résultats vont survenir : il s'agit simplement, et naturellement, des cas où les données disponibles ne sont pas représentatives du concept à apprendre. Autrement dit, tous les problèmes ne sont pas des problèmes bien posés et l'on ne peut exiger d'aucun système qu'il excelle sur de telles données. Plus particulièrement, nous imaginons mal que le dual d'un problème bien posé soit lui aussi un problème bien posé. Il faudrait pour cela que (A^+, A^-) soient représentatifs des deux concepts duaux, c'est-à-dire que A^+ contiennent des exemples de chaque sous-concept de ces deux concepts, et que A^- contiennent les exemples capables de séparer les sous-concepts dans les deux

cas.

7.5.3 Autres travaux

Notre approche peut être comparée aux techniques de catégorisation qui groupent les exemples en fonction de leurs similarités. Notre notion de similarité est cependant plus générale puisque, dans la version générique de GloBo, elle est encapsulée dans la définition de la propriété P que les paquets doivent satisfaire. Dans le cas de l'apprentissage supervisé, la *P-similarité* entre exemples dépend de l'ensemble des exemples négatifs : selon A^-, deux exemples positifs peuvent être très proches et dans le même paquet, ou bien associés à des paquets différents. Il est bien clair qu'une méthode qui, dans le cadre de l'apprentissage supervisé, n'utiliserait qu'une distance purement syntaxique (deux exemples restant à un même distance indépendamment des exemples négatifs), ne pourrait pas construire les paquets corrects que nous recherchons.

RISE [Domingos, 1995] utilise une telle distance syntaxique mais, à chaque fois qu'un exemple est ajouté au paquet le plus proche, le nouveau paquet doit être validé contre les exemples positifs et négatifs. Nous pensons que cette recherche gloutonne est avantageusement remplacée par le processus de sélection stochastique de GloBo.

Dans le même ordre d'idée, INDIE [Armengol et Plaza, 1997] utilise une méthode heuristique, basée sur une distance entre partitions, pour progressivement s'approcher de la partition correcte idéale des exemples. Ces deux systèmes souffrent de la même limitation : les paquets sont construits par une recherche gloutonne.

7.5.4 Perspectives

Une perspective majeure est de permettre à GloBo de travailler en ordre un, que ce soit dans le cadre d'un apprentissage supervisé ou non.

Dans le premier cas, nous sommes confrontés à la difficulté du test de subsomption en ordre un : la θ-subsomption [Plotkin, 1970] n'est pas un test efficace. Cependant, il existe des relations plus faibles que la θ-subsomption qui autorisent des traitements plus rapides. Malheureusement, ces relations induisent plusieurs moindres généralisés pour deux exemples [Sebag et Rouveirol, 1997]. Un objectif sera donc une version de GloBo gérant les moindres généralisés multiples.

Dans le second cas, c'est-à-dire en apprentissage non supervisé, la difficulté consiste à bâtir des distances entre clauses, problème qui a déjà été étudié et pour lequel des solutions sont connues [Bisson, 1992, Emde et Wettschereck, 1996, Sebag, 1997].

Après GloBo et ses heuristiques stochastiques, nous allons voir au chapitre suivant une variation liée à l'éco-résolution, l'objectif principal restant le même : trouver une couverture des exemples positifs de taille minimale.

Bibliographie

[Amoura et al., 1999] Amoura, L., Chauchat, J., et Rakotomalala, R. (1999). Echantillonnage rapide pour le traitement des variables continues dans les graphes d'in-

duction. In [Sebag, 1999a], pages 69–76.

[Armengol et Plaza, 1997] Armengol, E. et Plaza, E. (1997). Induction of feature terms with indie. In van Someren, M. et Widmer, G., éditeurs, *Proceedings of the ninth European Conference on Machine Learning*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 33–48. Springer-Verlag.

[Bisson, 1992] Bisson, G. (1992). Learning in FOL with a similarity measure. In Swartout, W., éditeur, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 82–87. MIT Press.

[Brézellec, 1999] Brézellec, P. (1999). Sondage et apprentissage. In [Sebag, 1999a], pages 63–68.

[Clark et Niblett, 1989] Clark, P. et Niblett, T. (1989). The cn2 induction algorithm. *Machine Learning*, 3(4) :261–283.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7) :1895–1924.

[Dietterich et al., 1997] Dietterich, T. G., Lathrop, R. H., et Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2) :31–71.

[Domingos, 1995] Domingos, P. (1995). Rule induction and instance-based learning : a unified approach. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1226–1232.

[Emde et Wettschereck, 1996] Emde, W. et Wettschereck, D. (1996). Relational instance based learning. In Saitta, L., éditeur, *Proceedings 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann.

[Gennari et al., 1989] Gennari, J. H., Langley, P., et Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40 :11–61.

[Gold, 1967] Gold, E. M. (1967). Language Identification in the Limit. *Information and Control*, 10 :447–474.

[Hogg et al., 1996] Hogg, T., Huberman, B. A., et Williams, C. P. (1996). Phase transitions and the search problem (editorial). *Artificial Intelligence*, 81(1–2) :1–15.

[Holland, 1975] Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* MIT Press.

[Merz et Murphy, 1996] Merz, C. J. et Murphy, P. M. (1996). UCI repository of machine learning databases.

[Michalski, 1983] Michalski, R. S. (1983). A theory and methodology of inductive learning. In Michalski, R. S., Carbonell, J. G., et Mitchell, T. M., éditeurs, *Machine Learning : An Artificial Intelligence Approach*, volume I, pages 83–134, Palo Alto, CA. Tioga.

[Paschos, 1997] Paschos, V. T. (1997). A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2) :171–209.

[Plotkin, 1970] Plotkin, G. (1970). A note on inductive generalization. In Meltzer, B. et Mitchie, D., éditeurs, *Machine Intelligence*, volume 5, pages 153–165. Edinburgh University Press.

[Provost et Fawcett, 1997] Provost, F. et Fawcett, T. (1997). Analysis and visualization of classifier performance : Comparison under imprecise class and cost distributions. In Heckerman, D., Mannila, H., Pregibon, D., et R., U., éditeurs, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48. AAAI Press.

[Provost et al., 1999] Provost, F., Jensen, D., et Oates, T. (1999). Efficient progressive sampling. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 23–32, San Diego, CA. ACM Press.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

[Sebag, 1996] Sebag, M. (1996). Delaying the choice of bias : A disjunctive version space approach. In L., S., éditeur, *13th International Conference on Machine Learning (ICML'96)*, pages 444–452. Morgan Kaufmann.

[Sebag, 1997] Sebag, M. (1997). Distance induction in first order logic. In Raedt de, L., éditeur, *Proceedings of the 15th International Joint Conference on Artificial Intelligence, Workshop on Inductive Logic Programming*.

[Sebag, 1999] Sebag, M., éditeur (1999). *Actes de la Première Conférence d'Apprentissage*.

[Sebag et Rouveirol, 1997] Sebag, M. et Rouveirol, C. (1997). Tractable induction and classification in FOL via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–892.

[Srinivasan et al., 1999] Srinivasan, A., King, R., et Bristol, D. (1999). An assessment of submissions made to the predictive toxicology evaluation challenge. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 270–276. Morgan Kaufmann.

[Srinivasan et al., 1997] Srinivasan, A., King, R. D., Muggleton, S. H., et Sternberg, M. J. E. (1997). The predictive toxicology evaluation challenge. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 4–9. Morgan Kaufmann.

[Torre, 1999] Torre, F. (1999). GloBo : un algorithme stochastique pour l'apprentissage supervisé et non supervisé. In [Sebag, 1999a], pages 161–168.

CHAPITRE 8

Les Vraizamis

Cakyamuni le Solitaire,
dit Sidarta Gautama le sage,
dit le Bouddah,
se saisit d'un morceau de craie rouge,
traça un cercle et dit :

Quand les hommes, même s'ils l'ignorent,
doivent se retrouver un jour, tout peut arriver
à chacun d'entre eux et ils peuvent suivre des
chemins divergents. Au jour dit, inéluctablement,
ils seront réunis dans le cercle rouge.

Rama Krishna.

Après avoir défini le problème de l'apprentissage supervisé disjonctif comme la recherche d'une couverture minimale de A^+ par des sous-ensembles maximalement corrects, nous avons proposé au chapitre précédent une solution stochastique, le système GloBo. Nous avons vu au cours de cette étude que les exemples « savent », à travers la correction de leur moindre généralisé, s'ils peuvent ou non se rassembler dans le même sous-concept. D'où l'idée de laisser les exemples se regrouper par eux-mêmes. Et effectivement, nous allons montrer dans ce chapitre que l'éco-résolution peut aussi offrir une solution originale à notre problème [Torre, 1999b].

Nous commencerons par présenter l'environnement dans lequel nous allons plonger les exemples positifs : il s'agit du monde des Vraizamis où les individus se rassemblent suivant un critère d'amitié. Nous verrons ensuite qu'en liant ce critère à la correction d'un paquet, l'évolution des Vraizamis conduit à une solution du problème d'apprentissage. Enfin, nous discuterons de la complexité de la méthode, de ses risques de blocage, de ses liens avec l'algorithme GloBo, et de l'influence de la topologie du monde des Vraizamis.

8.1 Introduction

Il est connu que pour de nombreuses tâches les agents intervenant dans des mondes artificiels doivent être dotés de la faculté d'apprentissage. Nous proposons la problématique inverse : en laissant évoluer un monde artificiel, peut-on résoudre des problèmes d'apprentissage tels que nous l'entendons, c'est-à-dire définir des concepts à partir d'exemples ?

Récemment, il a été montré que des systèmes simulant le comportement des fourmis pouvaient trouver d'intéressantes solutions pour des problèmes comme par exemple la recherche de chemins minimaux dans les graphes [Dorigo et al., 1996] (on sait cependant résoudre ce problème avec une complexité polynomiale), ou encore la construction de couvertures de graphes de très grande taille [Kuntz et al., 1997].

Nous sommes ici dans la droite ligne de Marvin Minsky [Minsky, 1988] : selon lui, l'intelligence est atteinte par l'interaction d'agents simples et dénués d'intelligence individuelle mais coopérant. Notre but est donc d'apprendre avec des agents qui n'ont absolument aucune faculté d'apprentissage individuelle. Notre approche s'inscrit aussi dans le programme de *l'éco-résolution* [Drogoul et Dubreuil, 1993] : nous allons utiliser des zamis, des créatures qui à l'état naturel se regroupent en bandes d'amis, les plus grandes possibles, dans le but d'atteindre l'état de Vraizamis. Nous proposons de laisser évoluer ces créatures pour résoudre un problème de partition minimale, auquel peut se ramener tout problème d'apprentissage.

Notre approche avec les Vraizamis est plutôt une approche à la Bouddah : nous laissons les individus se regrouper comme ils l'entendent (mais en respectant la correction vis-à-vis de A^- tout de même), et nous attendons que les groupes *naturels* se forment. Ainsi, nous pensons que les Vraizamis résolvent le problème décrit à la section 6.4, simplement en posant que la population est constituée des exemples positifs A^+ et que leur critère d'amitié est la correction par rapport à A^-.

Le résultat sera donc une partition que nous espérons minimale des exemples positifs par des paquets maximalement corrects d'exemples positifs.

Comme GloBo, les Vraizamis seront aussi capables d'effectuer des tâches d'apprentissage non supervisé : il suffira par exemple de définir le critère d'amitié en termes de spécificité des paquets.

8.2 Monde des Vraizamis

8.2.1 Rapide survol

Le monde des zamis est un anneau sur lequel se trouvent des niches qui constituent l'habitat des zamis. Dans ce monde, les règles suivantes sont observées.

1. Chaque niche peut loger un nombre illimité de créatures (c'est-à-dire que tous les zamis de l'anneau peuvent éventuellement tenir dans une seule niche).
2. L'anneau est orienté, les zamis peuvent le parcourir en passant d'une niche à l'autre, dans le sens des aiguilles d'une montre uniquement.
3. Au départ, il y a autant de niches que de zamis, et chaque zami est seul dans sa niche.

4. Les zamis ne partent jamais en même temps ! À un instant donné, une seule niche (déterminée) peut voir un (unique) zami partir et plus personne ne partira jusqu'à ce que celui-ci ait trouvé une place.
5. La nature amicale du zami et son envie de devenir un Vraizami le pousse rapidement à sortir de sa niche pour aller voir dans la niche suivante si les créatures qui s'y trouvent peuvent devenir des camarades de jeu. Deux cas peuvent se produire lorsqu'un zami arrive dans une niche.
   — Le nouveau groupe ainsi formé n'est pas viable, certaines créatures ne considèrent pas le nouveau venu comme leur zami et l'une des créatures de la niche va être expulsée vers la niche suivante pour ramener la paix. Nous savons qu'un seul départ peut suffire puisqu'en particulier il suffit de rejeter celui qui vient d'arriver. Cela dit, il est possible que le rejeté ne soit pas le dernier arrivé. Il est tout de même possible que ce soit le même zami qui est systématiquement rejeté tout au long de l'anneau et finit par revenir à son point de départ (où il est nécessairement bien accueilli).
   — Le second cas survient lorsque les créatures présentes et le nouveau venu s'entendent bien, auquel cas, aucune expulsion n'est requise même si un membre du groupe peut spontanément choisir de partir. Cependant, si le nouveau venu a laissé derrière lui une niche vide, celle-ci va être détruite puisque le zami est maintenant plus heureux ailleurs.

Notons que, après la propagation d'une éjection forcée, l'anneau ne contient que des niches où tout le monde s'entend bien.

Au fur et à mesure, qu'un groupe de zamis grandit, le départ de l'un d'entre eux devient de plus en plus difficile : les zamis ont alors tendance à se sédentariser. Cet effet est accentué par la disparition des niches.

8.2.2 Scènes de vie

Voici quelques illustrations de ces règles dans un monde où les zamis sont des entiers, un ensemble d'entiers formant un groupe de Vraizamis s'ils ont au moins deux facteurs premiers en commun.

Nous considérons un monde ne contenant que six entiers qui sont 6, 15, 18, 42, 75, et 105. Pour une meilleure compréhension des événements qui vont être décrits, nous donnons les facteurs premiers intervenant dans la décomposition de ces entiers :

Notons en particulier que 6, 18 et 42 forment un groupe de zamis, et que 42 peut aussi s'entendre avec 105.

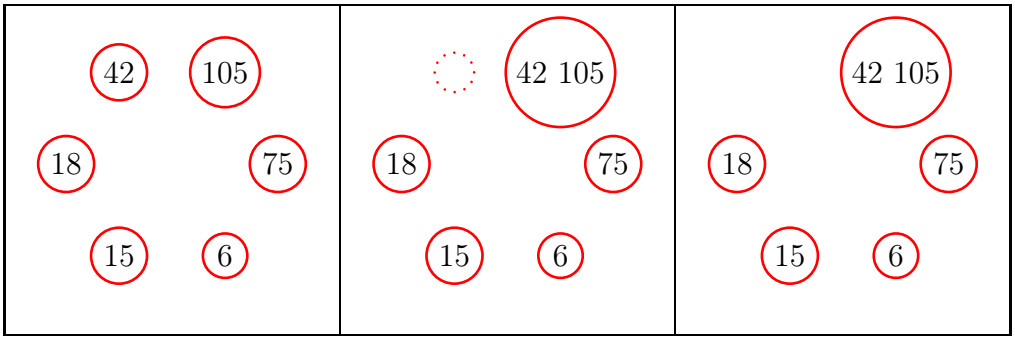
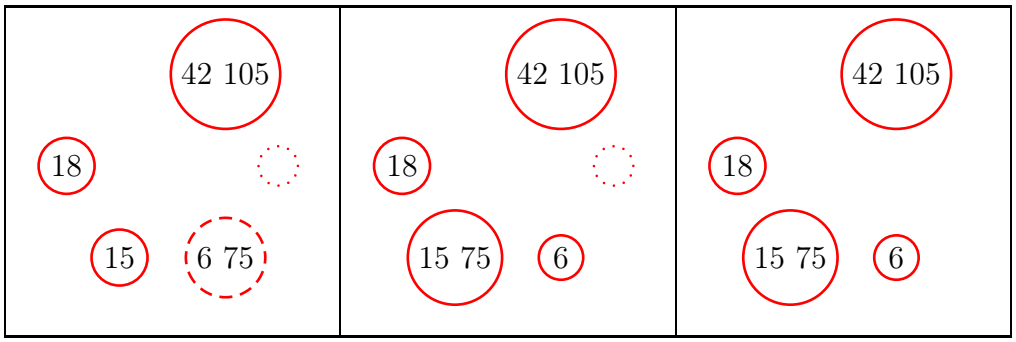


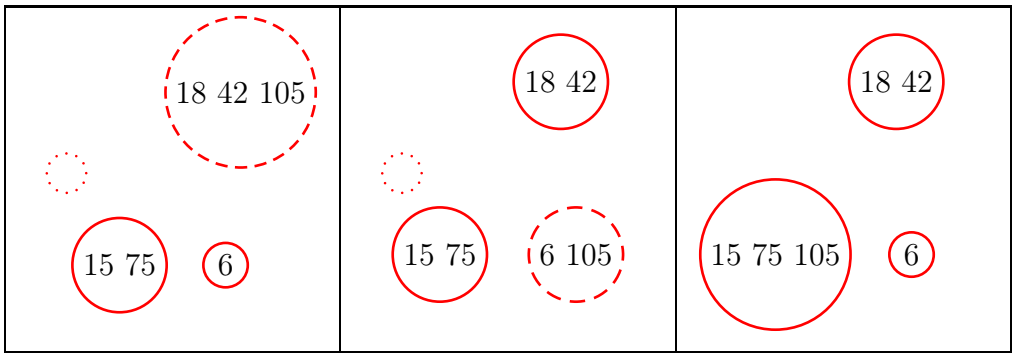
TABLE 8.1 – Amitié entre entiers

Individu	Facteurs premiers
6	2, 3
15	3, 5
18	2, 3
42	2, 3, 7
75	3, 5
105	3, 5, 7

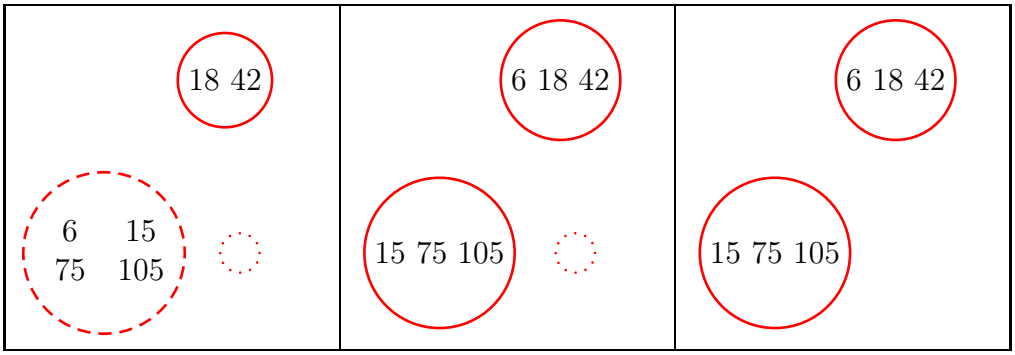
Au départ ils sont chacun dans une niche différente puis 42 décide de partir vers la case suivante, c'est-à-dire chez 105. Comme ils ont deux facteurs premiers en commun (3 et 7), 42 est accepté et la niche laissée vide est détruite.



42 ou 105 pourraient décider de s'en aller mais ça n'est pas le cas et c'est donc à la niche suivante de proposer un éventuel départ : en effet, 75 sort de cette niche. Malheureusement, la niche suivante contient 6 avec lequel 75 n'est pas ami, l'un d'eux doit partir. C'est 75 qui est désigné (au hasard) et qui arrive chez 15 avec lequel il s'entend bien. La niche d'origine de 75 peut maintenant être détruite.



Ensuite, 18 quitte sa niche pour rejoindre 42 et 105 mais le nouveau groupe n'est pas viable, quelqu'un doit s'en aller : 18 et 105 sont les seuls susceptibles de ramener l'harmonie par leur départ et c'est 105 qui est choisi (au hasard encore une fois) pour rejoindre 6. Encore une fois, cela se passe mal, 105 est à nouveau désigné et poursuit jusqu'à 15 et 75. À ce moment, et pas avant, la niche vacante peut être détruite.



Enfin, 6 se décide à sortir de sa niche, la niche suivante lui est hostile et comme il est le seul candidat au départ, il rejoint 18 et 42.

De nouveaux départs peuvent se produire mais sans changer l'aspect actuel de l'anneau : ainsi, si 15 passe dans la niche de 6, 18 et 42, quelqu'un doit en sortir et 15 est le seul candidat (les autres départs ne ramèneraient pas la paix dans la niche), si bien que 15 revient à son point de départ. Et il en est ainsi pour chaque entier de l'anneau. Nous dirons dans ce cas que l'anneau s'est stabilisé.

Nous pouvons enfin remarquer que, même si l'on fait des choix différents de ceux exposés ci-dessus, la solution découverte (peut-être au terme d'un temps assez long) est finalement la même.

8.3 Algorithmes

Nous considérons maintenant que l'amitié des Vraizamis est donnée par la correction de la niche vis-à-vis de A^-. Comme pour GloBo, d'autres instanciations sont envisageables (une correction affaiblie pour traiter les données bruitées, une limite sur une distance pour l'apprentissage non supervisé, etc.).

Algorithme 8.1 (propagation)
Étant donné e l'individu éjecté et N la niche courante.

1. *Si e est accepté dans N, c'est-à-dire si $N \cup \{e\}$ est correct par rapport à A^-, arrêter la propagation.*
2. *Sinon, choisir e' dans $N \cup \{e\}$ tel que $N \cup \{e\} - e'$ soit correct par rapport à A^-, et propager l'éjection de e' à la niche suivant N.*

Algorithme 8.2 (un cycle)

1. *Désigner au hasard N, la niche courante.*
2. *Tant que l'on n'a pas fait un tour de l'anneau :*
   (a) *En fonction de la probabilité d'éjection spontanée dans la niche N, décider d'éjecter ou non un membre de cette niche. S'il n'y a pas d'éjection, recommencer cette boucle avec la niche suivante, sinon poursuivre cette boucle.*
   (b) *Choisir aléatoirement e l'individu de N à éjecter.*
   (c) *Propager l'éjection de e à la niche suivant N (Algorithme 8.1).*

(d) Si e avait laissé une niche vide et que personne n'y est revenu durant la propagation, détruire cette niche.

Remarquez, que cette notion de cycle ne se confond pas avec celle de tour : une propagation et des éjections forcées peuvent amener un cycle de plusieurs tours.

La probabilité d'une éjection spontanée dans une niche est liée au nombre de créatures présentes dans cette niche et à un paramètre, noté T, que nous assimilons à une température. La formule exacte de la probabilité d'éjection d'une niche N est la suivante :

$$p(N) = \exp\left(-\frac{|N|}{T}\right)$$

En clair, plus le groupe d'amis formé est de taille importante, moins un départ est probable. Précisons que la température est fixée a priori et ne varie pas en cours de simulation.

Une fois que l'on a déterminé qu'une niche devait produire une éjection spontanée, l'éjecté est choisi de manière équiprobable. De la même manière, lorsqu'une éjection forcée est nécessaire, l'éjecté est choisi de manière équiprobable parmi les candidats, c'est-à-dire parmi ceux dont le départ ramènerait l'harmonie dans le niche.

Finalement, notre système est paramétré par le nombre de cycles à effectuer et la température de l'anneau. Nous verrons à la section suivante dédiée aux expérimentations comment ces paramètres peuvent être réglés.

8.4 Expérimentations

Nous allons maintenant décrire les expérimentations réalisées avec les Vraizamis dans les domaines du supervisé et du non supervisé. Au cours de cette description, nous évoquerons le système GloBo décrit au chapitre précédent et basé sur la même idée : couvrir les exemples positifs par un nombre minimal de paquets maximalement corrects. Cependant, GloBo est plus traditionnel dans le sens où il n'utilise pas l'évolution de créatures, même si ce système est stochastique. D'autre part, GloBo cherche une couverture là où les Vraizamis cherchent une partition.

Il nous fallait fixer les valeurs des paramètres de notre système.

— La température T est choisie de manière à ce que la probabilité d'éjection spontanée soit supérieure à 0.9 pour un individu seul dans sa niche, et inférieure à 0.1 dans une niche où tous les individus de l'anneau se seraient rassemblés.

— Il est plus délicat de fournir une règle pour déterminer le nombre de cycles. Cette valeur n'est pas liée au nombre d'individus car une augmentation du nombre d'individus est compensée par un cycle plus long. Une méthode consiste à essayer une valeur et de voir si elle conduit à une stabilisation de l'anneau, ce qui est assez facile à détecter comme nous l'avons vu à la section 8.2.2.

En vérité, ces réglages n'ont pas besoin d'être extrêmement fins : les intervalles de valeurs qui conduisent à une solution sont suffisamment larges pour que l'on puisse espérer tomber dedans sans trop d'efforts. Ainsi, toutes les expérimentations

qui suivent, bien que portant sur des problèmes différents, ont été réalisées avec un nombre de cycles valant 1000.

8.4.1 Apprentissage supervisé

Les expérimentations ont été réalisées sur le problème de la fin de jeu du morpion.

Pour prédire la classe d'un nouvel exemple, nous utilisons simplement les moindres généralisés des paquets formés par les Vraizamis.

Comme dans le cas de GloBo, nous avons tout d'abord utilisé le test `5x2cv` [Dietterich, 1998] pour déterminer la confiance avec laquelle on peut affirmer que les Vraizamis sont meilleurs que les autres systèmes sur cette tâche. Les résultats obtenus, identiques à ceux de GloBo, sont donnés à la Table 8.2.

Nous ne donnons pas la comparaison avec GloBo car dans toutes les exécutions nécessaires au test `5x2cv`, GloBo et les Vraizamis obtiennent 100% de bonnes prédictions, ce qui empêche naturellement de les différencier.

Ensuite, nous avons voulu tester chaque système en fonction du nombre d'exemples disponibles. Sur la Figure 8.1, nous donnons les prédictions moyennes obtenues sur 10 tirages, pour des ensembles d'apprentissage dont la taille varie de 5 à 95% de l'ensemble de données original. Toutes ces exécutions ont été réalisées avec une température de 40 et un millier de cycles.

Les Vraizamis parviennent à de très bonnes prédictions en n'utilisant que 15% des données disponibles.

Une telle exécution est illustrée Figure 8.2 : au début de l'évolution, des paquets non attendus mais corrects par rapport à A^- se forment ; mais rapidement, la pression exercée par la disparition des niches oblige les exemples à se rassembler en paquets de plus grande taille, et ainsi à converger vers les huit sous-concepts du morpion.

GloBo obtient des résultats comparables mais nous pouvons remarquer que les Vraizamis sont plus efficaces avec moins de données. Cela tient sans doute au fait que GloBo cherche des couvertures et que son espace de recherche est donc plus grand : avec peu d'exemples, plus de solutions s'offrent à GloBo qu'aux Vraizamis.

8.4.2 Apprentissage non supervisé

À nouveau, nous avons utilisé le problème *Quadruped Animals* concernant les mammifères à quatre pattes. Enfin, précisons que nous avons simplement utilisé une température de 120 et un millier de cycles.

TABLE 8.2 – Test `5x2cv` (Vraizamis)

	statistique $\tilde{t}$	Confiance
CN2	-2.215	93.00%
C4.5rules	-4.420	99.40%
C4.5	-7.836	$> 99.90\%$

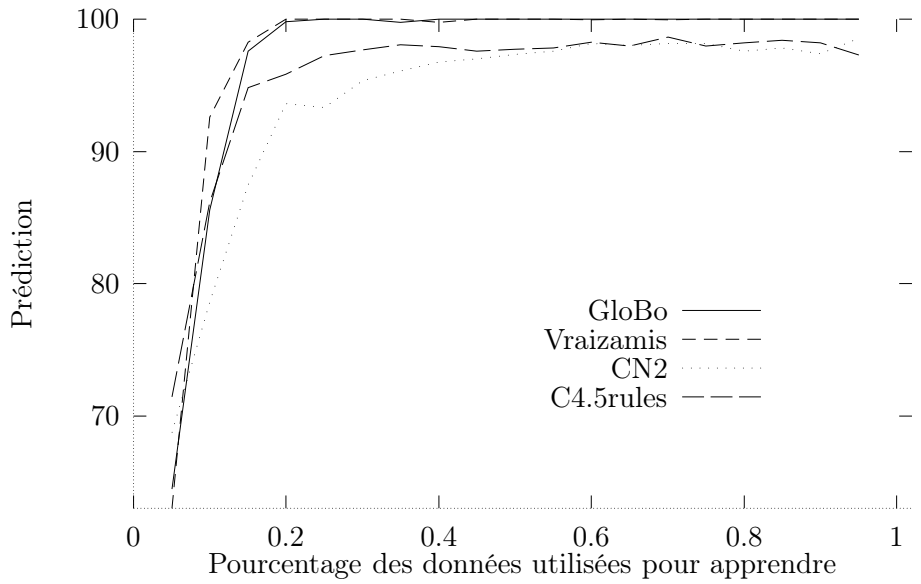


FIGURE 8.1 – Prédictions pour le problème du morpion.

Pour choisir la distance maximale dans un paquet, nous avons repris les valeurs favorables fournies par GloBo ; cela par simple souci d'économie de temps car les Vraizamis pourraient aussi bien découvrir eux-mêmes ces valeurs en construisant une courbe du type de celle de la Figure 7.8 (page 133).

Pour ces valeurs, nous retrouvons les mêmes paquets que ceux formés par GloBo. Nous avons tout de même remarqué expérimentalement que les Vraizamis réussissaient au-delà des intervalles définis par GloBo : cela s'explique simplement par le fait que GloBo cherche une couverture si bien que son espace de recherche est plus grand et qu'il est donc plus rapide à utiliser l'augmentation de la distance autorisée. Au fur et à mesure que la distance autorisée augmente, des couvertures deviennent possibles mais il faut attendre un peu plus pour qu'une nouvelle partition apparaisse.

8.5 Discussion et bilan

8.5.1 Complexité

Tout d'abord, il faut noter que le critère d'arrêt de notre algorithme est simplement lié au nombre de cycles voulu par l'utilisateur. Lorsque l'algorithme s'arrête une solution est disponible car à la fin de chaque cycle, l'état de l'anneau représente une solution au problème de couverture.

Commençons donc par calculer le nombre d'opérations effectuées au cours d'un cycle. Au pire, il y a $p - 1$ agrégations par cycle et autant de vérification de la propriété P.

Si nous posons que le nombre de cycles est indépendant du nombre d'exemples (ce que peuvent laisser penser nos expérimentations), nous avons alors une complexité linéaire dans le nombre d'individus.

En revanche, si nous supposons que le nombre de cycles est proportionnel au

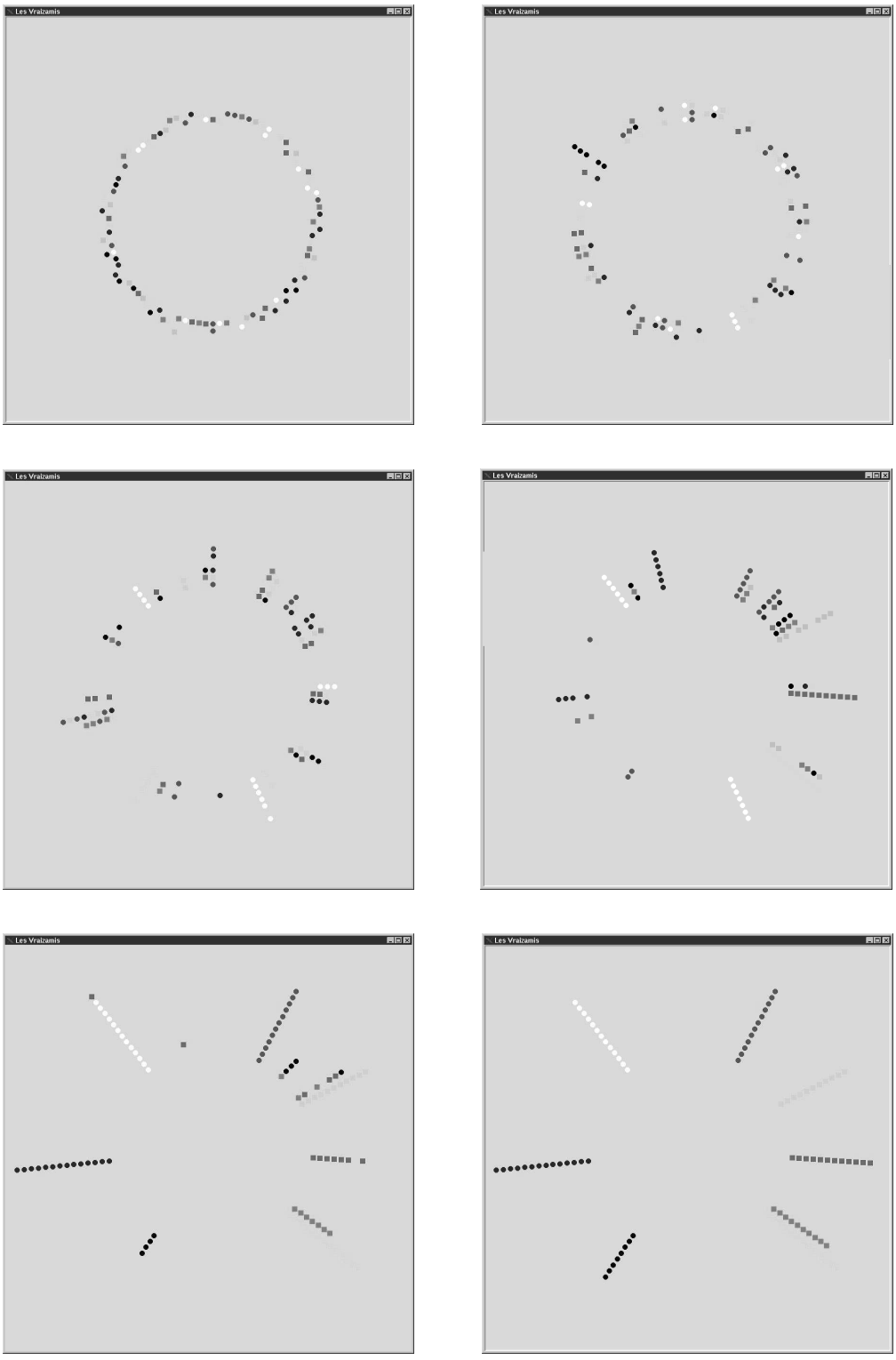


FIGURE 8.2 – Vraizamis et Morpion : les images

nombre d'individus (ce qui est sans doute plus raisonnable), nous obtenons une complexité quadratique. La vérification de la correction d'un paquet peut conduire à un test de couverture contre chaque exemple négatif disponible, si bien que la complexité des Vraizamis est cubique dans le nombre d'exemples disponibles, en termes de tests de subsomption, comme GloBo.

8.5.2 Topologie et convergence

A priori l'ordre de départ n'a pas d'importance : dans nos expérimentations la probabilité d'éjection est au départ d'environ 0.9, si bien que le premier cycle peut être vu comme un mélange aléatoire des individus. Au cours de ces expérimentations, aucun lien entre l'ordre de départ et la solution obtenue n'a effectivement été observé.

La topologie que nous avons décrite se justifie par le fait qu'elle est la plus simple imaginable (aussi parce qu'elle donne de bons résultats). À chaque instant, nous disposons d'une solution facilement identifiable, il n'est pas nécessaire de déterminer une direction pour le déplacement des zamis, et ceux-ci n'ont pas besoin de se souvenir des endroits favorables. Cette simplicité extrême est visible à travers les paramètres du système qui ne sont que deux : la température qui permet de régler la probabilité d'éjection spontanée et le nombre de cycles.

De plus, les solutions ne peuvent pas être cassées : la taille de l'anneau, et donc des solutions découvertes, diminue de manière monotone. Le prix pour cela est que les Vraizamis peuvent s'égarer dans un minimum local et ne plus pouvoir en sortir.

Depuis la situation de départ (un individu par niche), la solution optimale quelle qu'elle soit peut être atteinte (c'est la relation 6.1, page 109 qui nous garantit cela). Mais elle ne le sera pas nécessairement. Il peut y avoir un blocage : nous nous retrouvons alors dans une configuration de l'anneau depuis laquelle il n'est plus possible d'atteindre une solution optimale. Typiquement, de « mauvais » paquets se sont formés et l'anneau s'est stabilisé (c'est-à-dire qu'un individu éjecté revient à son point de départ) : il n'est plus possible d'atteindre la solution optimale par éjections d'individu unique, il faudrait briser les paquets formés et échanger des groupes d'individus. Nous parlons ici de solution optimale en termes de taille, indépendemment de toute mesure liée au problème ; en effet, rappellons que la situation de blocage constitue une solution au problème d'apprentissage même si elle n'est pas de taille minimale.

Ces blocages étaient prévisibles : le problème que nous nous proposions de résoudre est NP-difficile alors que notre algorithme est polynomial. Par suite, les Vraizamis ne fourniront pas systématiquement la meilleure solution. Il faudra se contenter d'une bonne solution, la meilleure de temps en temps.

Nous pourrions envisager, pour sortir d'un blocage, d'éjecter les individus par groupe, plutôt que un par un. Cela dit, lorsque l'on travaille sur un problème NP-difficile, le passage d'une solution approchée à la solution optimale est précisément NP-difficile. Par conséquent, nous pensons qu'il est vain de compliquer l'algorithme pour approcher un peu plus la solution optimale.

Expérimentalement, nous avons constaté que les blocages surviennent rarement et que la solution attendue est souvent découverte. Nous pouvons fournir un début d'explication dans le cas de l'apprentissage supervisé. Si les exemples positifs n'ap-

partenant pas à un même sous-concept sont séparables deux à deux, c'est-à-dire si nous disposons dans A^- d'un exemple négatif pour les séparer, il ne peut pas y avoir de blocage.

Bien sûr, vouloir que les exemples de sous-concepts différents soient séparables deux à deux est une exigence trop forte. Cependant, nous pouvons demander que le concept à découvrir soit convenablement représenté par les exemples fournis : il n'y aura alors pas de blocage et la solution sera découverte, si la simulation dure suffisamment longtemps. Nous revenons donc à une notion de représentativité des données tout à fait comparable à celle que nous avons défini durant l'étude de GloBo.

8.5.3 Autres travaux

En définitive, les approches *generate-and-test* à la Mitchell [Mitchell, 1982] ne sont peut-être pas toujours les plus appropriées : la généralisation peut être vue comme un problème de recherche mais il y a d'autres façons, parfois plus pertinentes, d'appréhender le problème. Comme nous l'avions indiqué dans l'introduction, les systèmes à base de fourmis illustrent cette idée. Citons aussi le clustering paramagnétique [Blatt et al., 1996] utilisé pour l'apprentissage non supervisé. Dans ce cas, on associe un spin à chaque instance, puis ces instances interagissent en fonction de leurs spins respectifs, de leur proximité et de la température. Finalement, lorsque la température est suffisamment basse, des régions de l'espace apparaissent alignées, c'est-à-dire que tous les éléments d'une telle région présentent le même spin et, naturellement, ces régions correspondent aux paquets cherchés.

Toutes ces méthodes reprennent l'idée qu'il y a un arrangement naturel des données qui ne demande qu'à apparaître. Parmi les variations permettant de faire apparaître cet ordre naturel, nous constatons que les Vraizamis constituent la méthode la plus simple. À ce titre, les performances obtenues par les Vraizamis sont d'autant plus remarquables.

Il est assez clair que les Vraizamis entrent dans le cadre de l'éco-résolution [Drogoul et Dubreuil, 1993]. En revanche, il est difficile d'établir un lien entre la Vie Artificielle et les Vraizamis même si la présentation, volontairement anthropomorphique, que nous en avons donnée peut laisser penser le contraire.

En effet, même la plus souple des définitions de la Vie Artificielle n'accorderait pas la vie à nos zamis : en particulier, nos créatures apparaissent spontanément, sans naissance liée à une reproduction quelconque, ni d'adaptation ni de mort [Fernández Ostolaza et Moreno Bergareche, 1997].

De même, il semble délicat de considérer les Vraizamis comme un système multi-agents. Les critères nécessaires sont difficilement remplis par les Vraizamis [Ferber, 1995] :

- les actions sont très limitées : le zami ne peut que changer de niche, et pas toujours de son plein gré ; il ne réalise rien, si ce n'est se déplacer ;
- la communication très simplifiée, voire inexistante : les zamis ont simplement l'intuition d'être bien ou non dans un groupe ;
- la perception de l'environnement se réduit à cette sensation ;
- il n'y a pas de coopération entre zamis.

Ce travail trouve aussi quelques lointains échos dans le domaine des émotions

[Cañamero et Van de Velde, 1997] où l'on pense que les émotions doivent nécessairement être présentes dans une Intelligence Artificielle [Cañamero et Van de Velde, 1997, Minsky, 1988]. Parmi ces émotions se trouvent des éléments comparables à notre critère d'amitié : un agent peut ressentir une antipathie violente vis-à-vis d'un autre agent et s'enfuir le plus loin et le plus rapidement possible ; à l'inverse, un coup de foudre tout aussi violent peut amener au rapprochement des agents concernés.

8.5.4 Perspectives

Nous avons vu que les Vraizamis réussissaient particulièrement bien lorsque les sous-concepts à former étaient disjoints. Cependant, dans le cas où les sous-concepts doivent se recouvrir, il est bien clair que les solutions fournies par les Vraizamis sont un peu moins pertinentes mais nous pouvons tout de même espérer voir apparaître des sous-concepts intéressants, même s'ils sont un peu trop spécifiques.

Pour construire des couvertures plutôt que des partitions, nous pourrons autoriser les individus à se dupliquer, pas au début de l'évolution car les doublons retarderaient sans doute la convergence mais plus tard, lorsque le nombre de niches semble s'être stabilisé. Une créature pourrait alors spontanément décider d'émettre une copie d'elle-même vers la niche suivante avec la condition que deux jumeaux ne peuvent survivre dans la même niche : si la copie revient à son point de départ et y retrouve son double, l'un des deux sera détruit. Cela nous rapprochera de la Vie Artificielle puisque cette réplication peut être assimilée à une auto-reproduction et qu'elle pourra être suivie d'une mort.

Bibliographie

[Blatt et al., 1996] Blatt, M., Wiseman, S., et Domany, E. (1996). Superparamagnetic clustering of data. *Physical Review Letters*, 76(3251).

[Cañamero et Van de Velde, 1997] Cañamero, D. et Van de Velde, W. (1997). Socially emotional : Using emotions to ground social interaction. In Dautehhahn, K., éditeur, *Socially Intelligent Agents - AAAI Fall Symposium*, pages 10–15. Technical Report FS-97-02. Menlo Park, CA : The AAAI Press.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7) :1895–1924.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., et Colorni, A. (1996). The ant system : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1) :29–41.

[Drogoul et Dubreuil, 1993] Drogoul, A. et Dubreuil, C. (1993). A distributed approach to N-puzzle solving. In *Proc. 12th Int. Work. Distributed Artificial Intelligence*, pages 95–108.

[Ferber, 1995] Ferber, J. (1995). *Les systèmes multi-agents - Vers une intelligence collective*. InterEditions.

[Fernández Ostolaza et Moreno Bergareche, 1997] Fernández Ostolaza, J. et Moreno Bergareche, A. (1997). *La vie artificielle*. Seuil.

[Kuntz et al., 1997] Kuntz, P., Layzell, P., et D., S. (1997). A colony of ant-like agents for partitioning in vlsi technology. In Husbands, P. et Harvey, I., éditeurs, *Proceedings of the Fourth European Conference on Artificial Life*, pages 417–424. MIT Press.

[Minsky, 1988] Minsky, M. (1988). *La société de l'esprit*. InterEditions, Paris.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18 :203–226.

[Torre, 1999] Torre, F. (1999). Les Vraizamis. In Sebag, M., éditeur, *Actes de la Première Conférence d'Apprentissage*, pages 177–184.

Conclusion

Finir est souvent plus difficile que commencer.
Jack Beauregard, Mon nom est Personne.

La première partie de cette thèse a été consacrée à l'utilisation des biais de langage, et plus généralement de toutes les informations disponibles sur la solution, en vue de rendre plus rapide le processus d'apprentissage.

Plus particulièrement, nous nous sommes intéressés à l'intégration de ces connaissances dans l'algorithme *générer-et-tester* appliqué à la Programmation Logique Inductive.

Le pré-supposé de notre démarche est que cette accélération de l'apprentissage doit passer par un élagage *dynamique* et *sûr* de l'espace de recherche :

— dynamique dans le sens où l'opérateur peut atteindre n'importe quelle hypothèse de l'espace de recherche mais que nous choisissons délibérément de ne pas raffiner certaines hypothèses ;

— sûr parce que tout en coupant ainsi des chaînes de raffinement, nous refusons de prendre le risque de perdre une solution.

Tout au long de cette première partie, nous avons réalisé une intégration progressive des propriétés connues de la solution dans l'algorithme *générer-et-tester*.

Nous avons tout d'abord cherché à réaliser un élagage par rapport à une propriété donnée sans modifier les choix d'instanciation de l'algorithme *générer-et-tester*. En particulier, nous avons supposé que l'opérateur de raffinement nous était imposé. Or, nous avons montré que l'opérateur étant fixé, seules certaines propriétés permettent un élagage sûr : les *propriétés privées* par rapport à cet opérateur.

Nous avons illustré l'utilisation de ces propriétés privées à travers le système Grandma qui nous permet de générer un langage d'hypothèses restreint par des propriétés à satisfaire. Si ces propriétés font apparaître les notions de correction et de complétude par rapport aux exemples, Grandma constitue un véritable système d'apprentissage. De manière plus générale, Grandma permet d'évaluer le pouvoir élagant de chaque biais et l'intérêt de chacune de leurs associations.

Nous avons ensuite adopté l'approche inverse de la précédente : au lieu de nous intéresser aux propriétes privées pour un opérateur fixé, nous sommes partis des

propriétés que nous voulions privées et avons cherché un opérateur approprié. A priori, il s'agit de la démarche la plus naturelle : un expert ne fournit que très rarement un opérateur ; en revanche, il amène des connaissances sur la solution et il nous semble déraisonnable de ne pas les utiliser au mieux.

Dans cet esprit mais de manière plus générale, nous avons caractérisé une relation que devait respecter les opérateurs pour rendre une propriété privée : la *relation naturelle* de cette propriété. Les opérateurs pertinents pour notre tâche d'apprentissage sont donc inclus dans cette relation naturelle.

Cette relation autorisant en général l'existence de nombreux opérateurs, nous avons choisi de nous intéresser aux *opérateurs idéaux* éventuellement présents dans cette relation. En sachant toutefois que de tels opérateurs n'existent pas pour les relations de généralité classiques comme la θ-subsomption ou l'implication logique.

En énonçant plusieurs conditions nécessaires à l'existence d'opérateurs idéaux, nous avons lié cette existence aux propriétés utilisées pour définir la relation naturelle. Nous avons ainsi pu repérer les biais de langage les plus favorables à l'existence d'opérateurs idéaux.

Finalement, l'intégration de la volonté d'élagage dynamique dans la relation de généralité nous a permis d'exhiber un opérateur idéal pour l'une de nos relations naturelles.

Nous avons constaté ensuite que certaines associations de propriétés possèdaient des relations naturelles très faibles, c'est-à-dire qui ne mettent en relation que très peu d'hypothèses.

Nous avons montré pourtant que tous les biais de langages pouvaient, en même temps, être utilisés efficacement. Pour cela, nous avons paramétré l'ensemble de l'algorithme *générer-et-tester* en fonction des propriétés connues de la solution :

— la relations de généralité (relation naturelle),
— les hypothèses servant de départ à la recherche (les bases),
— un notion de parcourabilité,
— une notion de complétude,
— deux notions d'optimalité (forte et faible),
— et enfin, une notion d'opérateur parfait.

Qu'il s'agisse de l'algorithme *générer-et-tester* ou non, il y a toujours un fossé entre les réglages d'un système d'apprentissage et les connaissances amenées par un expert du domaine. En particulier, nous ne pouvons pas envisager que l'on exige de cet expert qu'il règle lui-même les paramètres du système d'apprentissage.

Cette première partie est donc une contribution à la démarche qui consiste à déterminer automatiquement la configuration adéquate d'un système en fonction des connaissances que l'expert est en mesure de nous apporter. Nous avons perpétué en cela l'esprit du système HAIKU : permettre à l'utilisateur de définir des biais de haut niveau, compréhensibles dans son domaine, et adapter automatiquement et au mieux la procédure d'apprentissage en fonction de ces biais. Nous estimons que ce problème reste un défi majeur pour les années à venir.

Comme effets de bord notables de notre travail dans ce domaine, nous pouvons citer :

— la découverte d'un opérateur idéal alors qu'il est établi qu'il n'en existe pas

pour la θ-subsomption ;
— le dépassement du résultat négatif que constitue le *No Free Lunch Theorem* à travers des systèmes adaptables à chaque nouveau problème.

Nous avons pu finalement constater que les optimisations présentées dans la première partie de cette thèse ne permettent pas de gérer l'explosion combinatoire amenée par la recherche d'une définition disjonctive. Cela a motivé le travail décrit dans la seconde partie et portant sur l'apprentissage disjonctif.

Après avoir observé ces difficultés et les méthodes existantes, notre premier pas a consisté à caractériser le type de définition disjonctive que nous allions chercher à obtenir par la suite : des paquets maximalement corrects, en nombre minimal.

Nous avons tout d'abord présenté une méthode stochastique : le système GloBo. Celui-ci construit aléatoirement des paquets maximalement corrects, puis calcule une couverture minimale des exemples positifs par ces paquets.

Les expérimentations montrent des résultats satisfaisants en terme de prédiction puisque GloBo obtient des résultats sensiblement équivalents aux autres systèmes, ou parfois de meilleurs résultats comme dans le cas du *PTE Challenge* où GloBo a particulièrement brillé. Du point de vue de la taille de la solution, nous avons pleinement atteint notre objectif : GloBo découvre systématiquement les définitions les plus courtes. En résumé, à prédictivité au moins égale, GloBo obtient les solutions les plus compréhensibles.

Le coût pour ces résultats est une complexité cubique de l'algorithme utilisé par GloBo. Cependant, nous avons proposé une version générique qui, convenablement instanciée, autorise des apprentissages linéaire, non supervisé et permet de travailler avec des données bruitées.

En outre, l'analyse de GloBo nous a permis de définir une notion de confiance dans la solution découverte qui peut aussi être vue comme une mesure d'adéquation entre cette solution et les données.

Enfin, en conservant l'idée d'une couverture minimale par des paquets maximalement corrects, nous avons proposé une variation utilisant l'éco-résolution : les Vraizamis. Les résultats obtenus sont comparables à ceux du résultat GloBo, que ce soit en terme de prédiction ou de taille de la solution et, de la même manière, les Vraizamis sont eux aussi sensibles à la notion de représentativité définie pour GloBo.

Le trait majeur de nos systèmes est la recherche d'une couverture minimale par des paquets maximalement corrects. Nos expérimentations ont validé ce choix.

La notion de représentativité des données que nous avons définie, fournit une nouvelle réponse au *No Free Lunch Theorem* : si le problème est bien posé, GloBo produira une solution satisfaisante ; en revanche, si le problème est difficile pour GloBo, c'est-à-dire si les données ne sont pas représentatives du concept recherché, nos expérimentations nous amènent à penser que cette difficulté sera tout aussi réelle pour les autres systèmes.

Il reste à établir plus précisément dans quelle mesure cette représentativité est universelle et non pas liée spécifiquement à notre algorithme.

La poursuite des déclinaisons de GloBo présente selon nous des intérêts multiples : pour des langages utilisant la logique du premier ordre, pour l'apprentissage rapide et plus particulièrement pour le traitement du bruit. Nous estimons que ce dernier passe par l'identification des exemples bruités dans chacune des classes, puis par un traitement approprié de ces exemples. GloBo, avec ses différents niveaux et leur généricité, est un choix judicieux pour tester de telles approches.

Enfin, nous sommes d'avis que les problèmes comme l'apprentissage disjonctif dont on peut considérer qu'ils sont encore ouverts, méritent d'être considérés sous des angles différents. Ainsi, l'étude d'approches multi-agents et de méthodes impliquant la Vie Artificielle pour la résolution de problèmes disjonctifs devrait être riche d'enseignements.

CHAPITRE 9

Preuves

9.1 Relations naturelles

Preuve de la proposition 4.1 (unicité de la relation naturelle)
Tout d'abord, montrons que P est privée par rapport à $\succeq_P$. Raisonnons par l'absurde, supposons qu'il existe C et D dans $\mathcal{L}_h$ et une valeur du paramètre de P telles que

$$(C \succeq_P D) \wedge \overline{P(C)} \wedge P(D)$$

c'est-à-dire :

$$\left(P(C) \vee \overline{P(D)}\right) \wedge \overline{P(C)} \wedge P(D)$$

ce qui est trivialement impossible. Par suite, P est privée pout $\succeq_P$.

Ensuite, nous avons à prouver que $\succeq_P$ est l'unique relation naturelle de P. Supposons qu'il existe une relation $\mathcal{R}$ telle que P est privée par rapport à $\mathcal{R}$ et qu'il existe une paire (C, D) dans $\mathcal{R}$ qui ne soit pas dans $\succeq_P$. Par conséquent, pour certains paramètres de P, nous avons

$$\overline{P(C) \vee \overline{P(D)}}$$

c'est-à-dire :

$$\overline{P(C)} \wedge P(D)$$

Or, puisque P est privée par rapport à $\mathcal{R}$, nous devrions avoir

$$(C \,\mathcal{R}\, D) \wedge \overline{P(C)} \Rightarrow \overline{P(D)}$$

Cette contradiction démontre qu'une relation qui n'est pas contenue dans $\succeq_P$ ne peut pas rendre P privée et, en conclusion, $\succeq_P$ est l'unique relation naturelle de P. □

Preuve de la proposition 4.2 (définition de la relation naturelle)
Nous devons montrer que, pour toutes hypothèses H et H' dans $\mathcal{L}_h$,

$$\left[\forall k \in \mathcal{D}_f : (f(H) \,\mathcal{R}\, k) \vee \overline{(f(H') \,\mathcal{R}\, k)}\right] \Leftrightarrow f(H) \,\mathcal{R}\, f(H')$$

— Montrons tout d'abord que :

$$\left[\forall k \in \mathcal{D}_f : (f(H)\,\mathcal{R}\,k) \vee (\overline{f(H')\,\mathcal{R}\,k})\right] \Rightarrow f(H)\,\mathcal{R}\,f(H')$$

Raisonnons par l'absurde : supposons que cette implication est fausse. Il existerait donc deux hypothèses C et D dans $\mathcal{L}_h$ telles que

$$[\forall k : f(C)\,\mathcal{R}\,k \vee \overline{f(D)\,\mathcal{R}\,k}] \wedge \overline{f(C)\,\mathcal{R}\,f(D)}$$

Si nous choisissons $k = f(D)$, alors $\overline{f(D)\,\mathcal{R}\,k}$ ne peut pas survenir puisque la relation $\mathcal{R}$ est un pré-ordre, et donc reflexive. Par conséquent, il reste

$$f(C)\,\mathcal{R}\,f(D) \wedge \overline{f(C)\,\mathcal{R}\,f(D)}$$

Ce qui est naturellement imposible et nous pouvons conclure que l'implication de départ est vraie.

— Maintenant, considérons la formule suivante

$$\left[\forall k \in \mathcal{D}_f : (f(H)\,\mathcal{R}\,k) \vee (\overline{f(H')\,\mathcal{R}\,k})\right] \Leftarrow f(H)\,\mathcal{R}\,f(H')$$

et supposons qu'elle est fause. À nouveau, il doit exister C et D dans $\mathcal{L}_h$, et k dans $\mathcal{D}_f$ vérifiant

$$\overline{f(C)\,\mathcal{R}\,k} \wedge f(D)\,\mathcal{R}\,k \wedge f(C)\,\mathcal{R}\,f(D)\ .$$

Mais, en utilisant le fait que $\mathcal{R}$ est une relation transitive, cela revient à la contradiction suivante :

$$\overline{f(C)\,\mathcal{R}\,k} \wedge f(C)\,\mathcal{R}\,k$$

Nous avons prouvé les deux implications, par suite, l'implication est vraie. □

Preuve de la proposition 4.3 (propriété duale)

$$\begin{aligned} H \succeq_{\overline{P}} H' &\Leftrightarrow \forall \left(\overline{P}(H) \vee \overline{\overline{P}(H')}\right) \\ &\Leftrightarrow \forall \left(P(H') \vee \overline{P(H)}\right) \\ &\Leftrightarrow H' \succeq_P H \end{aligned}$$

Cela découle directement de l'expression de la relation naturelle de P. □

Preuve de la proposition 4.4 (conjonction de propriétés)

Nous devons montrer que

$$\forall \left((P_1 \wedge P_2)(H) \vee \overline{(P_1 \wedge P_2)(H')}\right) \Leftrightarrow H \geq_{P_1} D \wedge H \geq_{P_2} D$$

ou, reformulé, que

$$\forall \left[(P_1(H) \wedge P_2(H)) \vee \overline{P_1(H')} \vee \overline{P_2(H')}\right] \Leftrightarrow H \geq_{P_1} D \wedge H \geq_{P_2} D$$

Posons que les propriétés P_1 et P_2 sont exprimées comme suit.

$$\begin{aligned} P_1(H) &\Leftrightarrow f_1(H)\,\mathcal{R}_1\,k_1 \\ P_2(H) &\Leftrightarrow f_2(H)\,\mathcal{R}_2\,k_2 \end{aligned}$$

Par conséquent, nous devons prouver que

$$\begin{aligned} &\forall k_1, k_2 : \left[(f_1(H)\,\mathcal{R}_1\,k_1 \wedge f_2(H)\,\mathcal{R}_2\,k_2) \vee \overline{f_1(H')\,\mathcal{R}_1\,k_1} \vee \overline{f_2(H')\,\mathcal{R}_2\,k_2}\right] \\ \Leftrightarrow\ & f_1(H)\,\mathcal{R}_1\,f_1(H') \wedge f_2(H)\,\mathcal{R}_2\,f_2(H') \end{aligned}$$

— Commençons par l'implication suivante :

$$\begin{array}{ll} & \forall k_1, k_2 : \Big[(f_1(H)\,\mathcal{R}_1\,k_1 \wedge f_2(H)\,\mathcal{R}_2\,k_2) \vee \overline{f_1(H')\,\mathcal{R}_1\,k_1} \vee \overline{f_2(H')\,\mathcal{R}_2\,k_2}\Big] \\ \Rightarrow & f_1(H)\,\mathcal{R}_1\,f_1(H') \wedge f_2(H)\,\mathcal{R}_2\,f_2(H') \end{array}$$

Supposons que celle-ci soit fausse. Cela signifie qu'il existe C et D dans $\mathcal{L}_h$ telles que

$$\begin{array}{ll} & \forall k_1, k_2 : \Big[(f_1(C)\,\mathcal{R}_1\,k_1 \wedge f_2(C)\,\mathcal{R}_2\,k_2) \vee \overline{f_1(D)\,\mathcal{R}_1\,k_1} \vee \overline{f_2(D)\,\mathcal{R}_2\,k_2}\Big] \\ \wedge & \overline{f_1(C)\,\mathcal{R}_1\,f_1(D) \wedge f_2(C)\,\mathcal{R}_2\,f_2(D)} \end{array}$$

ou encore,

$$\begin{array}{ll} & \forall k_1, k_2 : \Big[(f_1(C)\,\mathcal{R}_1\,k_1 \wedge f_2(C)\,\mathcal{R}_2\,k_2) \vee \overline{f_1(D)\,\mathcal{R}_1\,k_1} \vee \overline{f_2(D)\,\mathcal{R}_2\,k_2}\Big] \\ \wedge & \Big[\overline{f_1(C)\,\mathcal{R}_1\,f_1(D)} \vee \overline{f_2(C)\,\mathcal{R}_2\,f_2(D)}\Big] \end{array}$$

Choisissons maintenant $k_1 = f_1(D)$ et $k_2 = f_2(D)$. Il vient alors que

$$[(f_1(C)\,\mathcal{R}_1\,k_1 \wedge f_2(C)\,\mathcal{R}_2\,k_2)] \wedge \Big[\overline{f_1(C)\,\mathcal{R}_1\,k_1} \vee \overline{f_2(C)\,\mathcal{R}_2\,k_2}\Big]$$

Ce qui n'est jamais possible.

— Supposons maintenant que l'implication suivante est fausse.

$$\begin{array}{ll} & \forall k_1, k_2 : \Big[(f_1(H)\,\mathcal{R}_1\,k_1 \wedge f_2(H)\,\mathcal{R}_2\,k_2) \vee \overline{f_1(H')\,\mathcal{R}_1\,k_1} \vee \overline{f_2(H')\,\mathcal{R}_2\,k_2}\Big] \\ \Leftarrow & f_1(H)\,\mathcal{R}_1\,f_1(H') \wedge f_2(H)\,\mathcal{R}_2\,f_2(H') \end{array}$$

Par suite, il existe C et D dans $\mathcal{L}_h$, k_1 et k_2 dans $\mathcal{D}_f$ satisfaisant

$$\begin{array}{ll} & \Big(\overline{f_1(C)\,\mathcal{R}_1\,k_1} \vee \overline{f_2(C)\,\mathcal{R}_2\,k_2}\Big) \wedge f_1(D)\,\mathcal{R}_1\,k_1 \wedge f_2(D)\,\mathcal{R}_2\,k_2 \\ \wedge & f_1(C)\,\mathcal{R}_1\,f_1(D) \wedge f_2(C)\,\mathcal{R}_2\,f_2(D) \end{array}$$

Puisque $\mathcal{R}_1$ et $\mathcal{R}_2$ sont des relations transitives, nous trouvons :

$$\Big(\overline{f_1(C)\,\mathcal{R}_1\,k_1} \vee \overline{f_2(C)\,\mathcal{R}_2\,k_2}\Big) \wedge f_1(C)\,\mathcal{R}_1\,k_1 \wedge f_2(C)\,\mathcal{R}_2\,k_2$$

□

9.2 Conditions d'existence d'un opérateur idéal

Preuve de la proposition 4.5 (inclusion de relations)
Trivialement, si $\succeq_a \subseteq \succeq_b$ alors $\succeq_a \cap \succeq_b = \succeq_a$. □

Preuve du lemme 4.1 (augmentation du nombre de variables)
Nous allons prouver ce lemme dans le cas de chaînes infinies non couvertes strictement descendantes :

$$D_1 > D_2 > \ldots > D_n > D_{n+1} > \ldots > C$$

La preuve pour des chaînes ascendantes est tout à fait comparable.

Chacune des hypothèses D_i dans cette chaîne θ-subsume la clause C ; par conséquent, la profondeur de C est une limite supérieure à la profondeur des D_i.

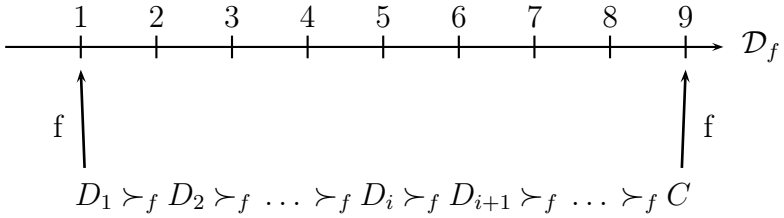
Or, il y a un nombre fini de clauses construites sur un ensemble fini de symboles de prédicat et de fonction, une profondeur bornée et un ensemble fini de variables.

Par suite, le seul moyen d'obtenir une chaîne infinie non couverte strictement descendante sous θ-subsomption de C, consiste à ajouter de nouveaux littéraux bâtis sur des symboles de prédicats apparaissant dans C, de profondeur inférieure à celle de C et contenant de nouvelles variables (voir exemple 4.5). □

Preuve du lemme 4.2 (valeur constante)
Suivant la définition d'une chaîne infinie non couverte (Définition 4.4),

$$D_1 \succeq_f D_2 \succeq_f \ldots \succeq_f D_i \succeq_f D_{i+1} \succeq_f \ldots \succeq_f C$$

Par conséquent, la valeur de $f(D_i)$ est encadrée par $f(D_1)$ et $f(C)$ comme montré sur la figure suivante...



... et nous constatons d'emblée sur ce dessin qu'une infinité de D_i doivent avoir la même valeur par f. □

Preuve du théorème 4.1 (chaînes sous $\succeq_f^\theta$)
À partir des lemmes 4.1 et 4.2. □

Preuve de la proposition 4.6 (ensembles couvrants infinis)
Considérons une clause D vérifiant $C \succ^\theta D$ et que $f(C) = f(D)$; nous savons que $C \succ_f^\theta D$. Supposons maintenant qu'il existe dans $\mathcal{L}_h$, une hypothèse E telle que

$$C \succ_f^\theta E \succ_f^\theta D$$

Cela signifierait que $f(E) = f(C) = f(D)$ et que $C \succ^\theta E \succ^\theta D$, mais cela contredit notre hypothèse. Par conséquent, C est une couverture supérieure de D pour la relation $\succeq_f^\theta$. Si C couvre deux clauses incomparables pour la θ-subsomption, (et donc incomparables pour la relation $\succeq_f^\theta$), alors toutes deux doivent apparaître dans la couverture inférieure de C par rapport à $\succeq_f^\theta$. Nous pouvons en déduire que, s'il existe un nombre infini de telles clauses D, alors la couverture inférieure de C est elle aussi infinie. □

9.3 Idéalité de $\rho_{||}^\theta$

Nous devons prouver que notre opérateur $\rho_{||}^\theta$ est idéal, c'est-à-dire, montrer qu'il est à la fois localement fini, strict et complet par rapport à la relation $\succeq_{||}^\theta$.

9.3.1 $\rho^{\theta}_{||}$ est localement fini

L'ajout est localement fini

On ajoute un littéral dont le symbole est un symbole de prédicat quelconque et les arguments de nouvelles variables. Par conséquent, il y a autant de littéraux ajoutables que de symboles de prédicat disponibles. Si cet alphabet est fini, la transformation est localement finie.

L'unification est localement finie

Le fait que C soit réduite ou non n'y change rien : le nombre de variables dans une clause est fini et, par suite, le nombre d'unifications possibles de deux variables distinctes de C, est lui aussi fini. S'il est nécessaire d'ajouter des littéraux en même temps que l'unification, l'argument que nous venons de donner pour l'ajout est valable.

Le raffinement des clauses équivalentes est localement fini

Le point 4 calcule les clauses équivalentes à C, et applique les points précédents sur ces clauses. Or, nous venons de voir que ces transformations étaient localement finies. Il nous reste donc à montrer que les clauses équivalentes à C, c'est-à-dire θ-équivalentes et de même taille, sont en nombre fini et calculable.

Lemme 9.1 (théorème 2 de [Plotkin, 1970])
Si C est réduite, les clauses réduites qui lui sont θ-équivalentes sont des renommages.

Lemme 9.2
Les clauses non réduites θ-équivalentes à C réduite, sont de taille strictement supérieure à celle de C.

Preuve
Les clauses non réduites équivalentes à C, contiennent C elle-même et sont donc de taille strictement supérieure à celle de C. □

Par suite, ces clauses ne vont pas nous intéresser puisque, de tailles supérieures à celle de C, elles ne sont pas équivalentes à C sous $\succeq^{\theta}_{||}$.

De ces deux lemmes, nous déduisons que le point 4 de notre opérateur ne concerne que les clauses non réduites. Nous nous plaçons donc maintenant dans le cas où C n'est pas réduite, et l'on note C_r, une clause réduite θ-équivalente à C.

Lemme 9.3
Si E est une clause θ-équivalente à C, E contient C_r.

Preuve
Si E est réduite, C_r et E sont des renommages (lemme 9.1) et notre lemme est vérifié. Si E n'est pas réduite, il existe E_r réduite, contenue dans E et, par hypothèse, θ-équivalente à C_r. Par suite, C_r et E_r sont des renommages et, finalement, C_r est contenue dans E. □

Il s'en suit que les clauses cherchées vont être obtenues par ajout de n littéraux L_i sur C_r, où $n = |C| - |C_r|$. La clause résultat doit toujours être équivalente à C_r ; pour cela, on demande que $\{L_i\}_{1 \leq i \leq n}$ θ-subsume C_r.

Ces littéraux sont en nombre fini. Il suffit pour s'en convaincre de constater que les L_i sont construits à partir de symboles de prédicat apparaissant dans C_r, et que la profondeur de ces littéraux est bornée par la profondeur de C_r (puisque A θ-subsume B entraîne que la profondeur de A est inférieure à celle de B).

Un algorithme trivial consiste à générer l'ensemble de tous ces littéraux, à choisir dans cet ensemble des combinaisons de taille n et à vérifier pour chacune de ces combinaisons qu'elle θ-subsume C_r.

Cet algorithme est trivial mais peu efficace. On songera à construire les clauses équivalentes en généralisant les littéraux déjà présents dans C_r.

L'ajout sur les clauses réduites est localement fini

Le nombre de littéraux à ajouter est fixé et le nombre de littéraux candidats est borné.

9.3.2 $\rho^{\theta}_{||}$ est strict

L'ajout est strict

Nous ajoutons un littéral à la clause C. Ce littéral ne peut pas être déjà présent puisqu'il contient de nouvelles variables. Par conséquent, $C \succeq^{\theta} D$ (les deux clauses peuvent être θ-équivalentes si le prédicat du littéral ajouté apparaissait déjà dans C) et la taille augmente bien de 1. Par conséquent, la subsomption de D par C sous $\succeq^{\theta}_{||}$ est stricte.

L'unification est stricte

L'ajout de littéraux n'est là que pour assurer le maintien de la taille de la clause. Nous allons montrer que l'unification de deux variables par notre opérateur (points 2 et 3) produit une clause strictement plus spécifique pour la θ-subsomption.

- C **est réduite.** Soient V_1, V_2 deux variables apparaissant dans C. On a bien C qui θ-subsume $C\{V_1/V_2\}$ avec la substitution $\{V_1/V_2\}$. Supposons que C soit θ-équivalente à $C\{V_1/V_2\}$, c'est-à-dire il existe θ telle que $C\{V_1/V_2\}\theta \subseteq C$. Cela est contradictoire, puisque cela amène que C n'est pas réduite.
- C **n'est pas réduite.** Soient V_1, V_2 deux variables apparaissant dans C et telles que $\{V_1/V_2\}$ n'appartient pas à une substitution σ vérifiant $C\sigma \subseteq C$. Supposons que C soit θ-équivalente à $C\{V_1/V_2\}$, c'est-à-dire il existe θ telle que $C\{V_1/V_2\}\theta \subseteq C$. Alors $\sigma = \{V_1/V_2\}\theta$ vérifie $C\sigma \subseteq C$ et $\{V_1/V_2\} \in \sigma$, ce qui contredit notre hypothèse.

Le raffinement des clauses équivalentes est strict

Cela ne dépend que des points précédents.

L'ajout sur les clauses réduites est strict

On ajoute à C_r des littéraux jusqu'à en avoir autant que dans C. La clause résultat est donc de même taille. Comme on ajoute un littéral avec un symbole de prédicat n'apparaissant pas dans C, on obtient une clause strictement plus spécifique pour la θ-subsomption.

9.3.3 $\rho^\theta_{||}$ est complet

Soient C et D deux clauses de l'espace de recherche vérifiant $C \succeq^\theta_{||} D$. Pour prouver la complétude de $\rho^\theta_{||}$, nous devons montrer qu'il existe de chaîne de raffinements

$$C = A_0 \succeq^\theta_{||} A_1 \succeq^\theta_{||} \ldots \succeq^\theta_{||} A_n \sim^\theta_{||} D$$

En guise de preuve, nous allons donner un algorithme qui calcule cette chaîne.

On va utiliser la fonction Occ définie comme suit : $\textsc{Occ}(p, C)$ est le nombre d'occurrences du prédicat p dans la clause C.

L'intuition

Nous allons suivre la procédure suivante :

1. Enlever les prédicats en surnombre dans A_i par rapport à D. Nous avons ensuite $\textsc{Occ}(p, A_i) \leq \textsc{Occ}(p, D)$.
2. Ajouter les symboles de prédicats manquants dans A_i par rapport à D. Nous avons alors $\textsc{Occ}(p, A_i) = \textsc{Occ}(p, D)$.
3. Appliquer les substitutions réduisant la taille de A_i pour arriver à une A_j pour laquelle il existe θ vérifiant $A_j\theta = D$. On a alors A_j θ équivalente à D et il ne reste donc qu'à appliquer θ. Dans la suite, nous travaillons sur cette substitution θ.
4. Appliquer les substitutions de θ qui produisent des clauses équivalentes à A_i.
5. Appliquer les substitutions restantes de θ.

On peut se convaincre que cet algorithme termine et fournit un A_i équivalent à D. Il reste à s'assurer que ce cheminement peut être suivi par notre opérateur. C'est ce que nous allons vérifier en détaillant cet algorithme.

L'algorithme

1. Soient $A_0 = C$ et $i = 0$.
2. Si, pour tout p, $\textsc{Occ}(p, A_i) \leq \textsc{Occ}(p, D)$, on passe à l'étape 3.
   Sinon, il existe p tel que $\textsc{Occ}(p, A_i) > \textsc{Occ}(p, D)$. Cela peut signifier deux choses.
   - A_i n'est pas réduite et p intervient dans littéral L redondant.
     - Il existe q tel que $\textsc{Occ}(q, A_i) = 0$ et $\textsc{Occ}(q, A_i) > 0$. Par le point 5, on fait disparaître L de A_i, ce qui nous fournit une clause A'_i, θ-équivalente à A_i. Ensuite, par application du point 1, on ajoute q (avec de nouvelles variables) à A'_i pour obtenir A_{i+1}.

— Par le point 4 de l'opérateur, on construit A'_i en remplaçant L par L' qui est un littéral maximalement spécifique tel que A'_i est toujours équivalente à A_i, L' θ-subsume strictement un littéral de D qui n'intervenait pas dans la subsomption de A_i par D. À noter que L' peut très bien réutiliser p.
Puis, on fait un pas de spécialisation sur A'_i grâce à l'un des premiers points de l'opérateurs, ce qui nous fournit A_{i+1}. Cette spécialisation existe par définition de L'.

— Soit θ la substitution telle que $A_i\theta \subseteq D$. Il y a dans θ une unification u telle que $A_i >^{\theta} A_iu$ et $|A_i| > |A_iu|$. Cette unification existe, sinon nous serions dans le cas précédent. On applique u avec le point 2 ou le point 3 selon que A_i est réduite ou non. Nous avons les mêmes alternatives que dans le cas précédent pour ajouter un nouveau littéral.

Incrémenter i et recommencer l'étape 2.

3. Si, pour tout p, $\text{Occ}(p, A_i) = \text{Occ}(p, D)$, on passe à l'étape 4.
Sinon, il existe p tel que $\text{Occ}(p, A_i) < \text{Occ}(p, D)$. On construit A_{i+1} par ajout du littéral composé de p et de nouvelles variables sur A_i, et cela en utilisant le point 1 de l'opérateur.
Incrémenter i et recommencer l'étape 3.

4. Soit θ la substitution telle que $A_i\theta \subseteq D$.
S'il n'existe pas dans θ une unification u telle que $|A_iu| < |A_i|$, on passe à l'étape 5.
Sinon, on va appliquer u en cherchant à conserver $\text{Occ}(p, A_{i+1}) = \text{Occ}(p, D)$. Encore une fois, il y a deux possiblités.
— Si A_i θ-subsume strictement A_iu, on applique u par le point 2 ou le point 3, et l'on ajoute, pour obtenir A_{i+1}, un littéral utilisant le symbole de prédicat disparu et de nouvelles variables.
— Si A_i est θ-équivalente à A_iu, on applique le point 4 pour pouvoir travailler sur $A'_i = A_iu \cup \{L\}$ telle que A'_i est toujours θ-équivalente à A_i et L est un littéral maximalement spécifique θ-subsumant strictement un littéral de D qui n'intervenait pas dans la subsomption de A_i par D (L utilise le symbole de prédicat du littéral qui a disparu).
Puis, on fait un pas de spécialisation sur A'_i grâce à l'un des premiers points de l'opérateurs, ce qui nous fournit A_{i+1}. Cette spécialisation existe par définition de L.

Incrémenter i et recommencer l'étape 4.

5. S'il n'existe pas de substitution σ incluse dans θ telle que $A_i\sigma$ est θ-équivalente à A_i, on passe à l'étape 6.
Sinon, on choisit la substitution σ incluse dans θ vérifiant $A_i\sigma$ est θ-équivalente à A_i et, pour toute unification u de $\theta - \sigma$, $A_iu >^{\theta}$. Grâce au point 4, on travaille sur A'_i défini par $A'_i = A_i\sigma$, puis on applique une unification de $\theta - \sigma$ à l'aide de l'un des points 2 et 3. Si une réduction de taille s'est produite, on ajoute, pour obtenir A_{i+1}, un littéral utilisant le symbole de prédicat disparu et de nouvelles variables.
Incrémenter i et recommencer l'étape 5.

6. Soit θ la substitution telle que $A_i\theta \subseteq D$.
   Si $\theta = \emptyset$, on passe à l'étape 7.
   Sinon, on prend une unification u dans θ et on l'applique à l'aide d'un des points 2 ou point 3. On obtient ainsi A_{i+1}.
   Incrémenter i et recommencer l'étape 6.
7. Soit $n = i$. Fin de l'algorithme.

ANNEXE A

Compléments

A.1 Clauses définies et substitutions

— Une *constante* est le nom d'un objet précis : `0`, `zéro`, `jean` ou encore `train42` peuvent être des constantes.
— Une *variable* permet de nommer un objet sans l'identifier précisément comme avec une constante. Typiquement, nous utilisons une lettre majuscule pour dénoter une variable.
— Une *fonction* est un symbole et une arité. Par exemple, `s` d'arité 1, pourra représenter la fonction qui à un entier n associe son successeur, c'est-à-dire $n + 1$. Il est important de noter qu'il s'agit seulement d'une représentation : il n'y a pas de calcul effectif associé aux fonctions. Enfin, notons aussi qu'une constante peut être vue comme une fonction d'arité 0.
— Un *terme* est une constante, une variable, ou un symbole de fonction associé à ses arguments qui sont aussi des termes (le nombre de ces termes doit correspondre à l'arité de la fonction). Ainsi, l'ensemble des termes que nous pouvons construire avec la constante `0` et le symbole unaire `s` représente les entiers naturels (il s'agit de la représentation de Péano) : `s(s(0))` représente l'entier 2 et, si `s` et `0` sont les seuls symboles disponibles, alors `s(V)` représente un entier strictement supérieur à zéro.
— Un *prédicat* est assez comparable à une fonction : il nécessite aussi un symbole, une arité et prend une valeur booléenne. `plus`, d'arité 3, est un prédicat qui sera vrai lorsque le dernier argument sera la somme des deux premiers.
— Un *atome* est un symbole de prédicat avec des termes comme arguments. Nous pourrons lui associer une valeur de vérité, selon les valeurs de ses arguments. L'atome `plus(0,X,X)` indique ainsi que somme de zéro et d'une valeur `X` quelconque, vaut `X`.
— Un *littéral* est un atome nié ou non : s'il est nié le littéral est dit *négatif*, *positif* sinon.
— Une *clause définie* : rassemble des atomes sous la forme suivante :

$$A_0 \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

A_0 est la tête de la clause, les autres atomes formant son corps. En faisant valoir que le corps de la clause implique la tête, il est aisé de reformuler la clause comme suit :

$$A_0 \vee \overline{A_1} \vee \overline{A_2} \vee \ldots \vee \overline{A_n}$$

Une clause définie est donc une disjonction de littéraux : l'un positif, les autres négatifs. Implicitement, les variables de la tête sont quantifiées universellement, celles n'apparaissant que dans le corps sont quantifiées existentiellement.

Enfin, nous introduisons la notion de *substitution*.

Définition A.1 (substitution)
Une substitution est un ensemble de couples terme/variable. L'application d'une substitution θ à une clause C fournit une clause, notée $C\theta$, obtenue en remplaçant les variables de C apparaissant dans θ par les termes correspondants.

A.2 Le test 5×2cv

Comme son nom l'indique, la méthode proposée par [Dietterich, 1998] pour départager deux algorithmes, est basée sur une validation croisée deux fois, itérée cinq fois, et cela pour chacun des deux algorithmes en compétition.

À chaque itération, les données disponibles sont partitionnées en deux ensembles de même taille : S_1 et S_2. Pour chacune de ces itérations, les erreurs estimées p_M^i sont calculées : i indique lequel des S_i a servi de base d'apprentissage et M est l'algorithme utilisé pour l'apprentissage. Nous obtenons donc quatre valeurs : $p_A^{(1)}$, $p_B^{(1)}$, $p_A^{(2)}$ et $p_B^{(2)}$.

Cela nous permet de calculer ensuite la différence entre les deux algorithmes, sur chacun des ensembles d'apprentissage :

$$p^{(1)} = p_A^{(1)} - p_B^{(1)}$$
$$p^{(2)} = p_A^{(2)} - p_B^{(2)}$$

puis, la moyenne de ces deux différences :

$$\overline{p} = \frac{p^{(1)} + p^{(2)}}{2}$$

Finalement, la variance est estimée par

$$s^2 = \left(p^{(1)} - \overline{p}\right)^2 + \left(p^{(2)} - \overline{p}\right)^2$$

Puisque ce procédé est appliqué cinq fois, nous obtenons cinq valeurs de la variance qui seront notées s_i^2, i allant de 1 à 5. Enfin, il faut distinguer le premier $p^{(1)}$ obtenu : il sera noté $p_1^{(1)}$.

Thomas Dietterich propose alors de calculer la quantité $\tilde{t}$ définie comme suit :

$$\tilde{t} = \frac{p_1^{(1)}}{\sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}}$$

Une fois cette mesure effectuée, il reste à comparer la valeur obtenue à la Table A.1), pour déterminer avec quelle confiance nous pouvons préférer l'algorithme A à l'algorithme B.

TABLE A.1 – Confiance à accorder selon Dietterich

$\tilde{t}$	Confiance
0.267	20%
0.727	50%
1.476	80%
2.015	90%
2.571	95%
3.365	98%
4.032	99%
4.773	99.5%
5.893	99.8%
6.869	99.9%
11.178	99.99%

BIBLIOGRAPHIE

[Adé et al., 1995] Adé, H., De Raedt, L., et Bruynooghe, M. (1995). Declarative bias for specific-to-general ILP systems. *Machine Learning*, 20 :119–154.

[Aha, 1991] Aha, D. W. (1991). Incremental constructive induction : An instance-based approach. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 117–121. Morgan Kaufmann.

[Alblas, 1991] Alblas, H. (1991). Introduction to Attribute Grammars. In Alblas, H. et Melichar, B., éditeurs, *Proceedings of International Summer School on Attribute Grammars, Applications and Systems*, volume 545 of *LNCS*, pages 1–15. Springer-Verlag.

[Amoura et al., 1999] Amoura, L., Chauchat, J., et Rakotomalala, R. (1999). Echantillonnage rapide pour le traitement des variables continues dans les graphes d'induction. In [Sebag, 1999a], pages 69–76.

[Armengol et Plaza, 1997] Armengol, E. et Plaza, E. (1997). Induction of feature terms with indie. In van Someren, M. et Widmer, G., éditeurs, *Proceedings of the ninth European Conference on Machine Learning*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 33–48. Springer-Verlag.

[Bergadano et Gunetti, 1993] Bergadano, F. et Gunetti, D. (1993). An interactive system to learn functional logic programs. In *Proceedings of IJCAI-93*, pages 1044–1049. Morgan Kaufmann.

[Bergadano et Gunetti, 1995] Bergadano, F. et Gunetti, D. (1995). Learning clauses by tracing derivations. In *ILP : from ML to Software Engineering*. MIT Press.

[Bisson, 1992] Bisson, G. (1992). Learning in FOL with a similarity measure. In Swartout, W., éditeur, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 82–87. MIT Press.

[Blatt et al., 1996] Blatt, M., Wiseman, S., et Domany, E. (1996). Super-paramagnetic clustering of data. *Physical Review Letters*, 76(3251).

[Boström, 1995] Boström, H. (1995). Covering vs. divide-and-conquer for top-down induction of logic programs. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1194–1200.

[Brézellec, 1999] Brézellec, P. (1999). Sondage et apprentissage. In [Sebag, 1999a], pages 63–68.

[Brodley, 1993] Brodley, C. E. (1993). Addressing the selective superiority problem : Automatic algorithm/model class selection. In Utgoff, P., éditeur, *Proceedings 10th International Conference on Machine Learning*, pages 17–24. Morgan Kaufmann.

[Buntine, 1988] Buntine, W. (1988). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36(2) :375–399.

[Cañamero et Van de Velde, 1997] Cañamero, D. et Van de Velde, W. (1997). Socially emotional : Using emotions to ground social interaction. In Dautehhahn, K., éditeur, *Socially Intelligent Agents - AAAI Fall Symposium*, pages 10–15. Technical Report FS-97-02. Menlo Park, CA : The AAAI Press.

[Caruana et al., 1998] Caruana, R., de Sa, V., Kearns, M., et McCallum, A., éditeurs (1998). *NIPS*98 Workshop : Integrating Supervised and Unsupervised Learning.*

[Champesme et al., 1995a] Champesme, M., Brézellec, P., et Soldano, H. (1995a). Empirically conservative search space reductions. In Raedt, L. D., éditeur, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 387–402. Department of Computer Science, Katholieke Universiteit Leuven.

[Champesme et al., 1995b] Champesme, M., Brézellec, P., et Soldano, H. (1995b). Réduction de l'espace de recherche en apprentissage : résultats théoriques et expérimentaux. In Nicolas, J., éditeur, *Dixièmes Journées Francophones sur l'Apprentissage*, pages 65–84.

[Clark et Niblett, 1989] Clark, P. et Niblett, T. (1989). The cn2 induction algorithm. *Machine Learning*, 3(4) :261–283.

[Cohen et Feigenbaum, 1982] Cohen, P. R. et Feigenbaum, E. A. (1982). *The Handbook of Artificial Intelligence*, volume 3. HeurisTech Press and William Kaufmann, Stanford, California and Los Altos, California.

[Cohen, 1994] Cohen, W. W. (1994). Grammatically biased learning : Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68 :303–366.

[Cornuéjols et Moulet, 1997] Cornuéjols, A. et Moulet, M. (1997). Machine learning : a survey. In Tzafestas, S., éditeur, *Knowledge based sytems*, chapitre 2, pages 61–86. World Scientific.

[De Raedt, 1992] De Raedt, L. (1992). *Interactive Theory Revision : An Inductive Logic Programming Approach.* Academic Press.

[De Raedt et Bruynooghe, 1993] De Raedt, L. et Bruynooghe, M. (1993). A theory of clausal discovery. In Bajcsy, R., éditeur, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann.

[Dehaspe et De Raedt, 1996] Dehaspe, L. et De Raedt, L. (1996). Dlab : A declarative bias formalism. In Rás, Z. W. et Michalewicz, M., éditeurs, *Proceedings of the International Symposium on Methodologies for Intelligent Systems (ISMIS'96)*, volume 1079 of *LNAI*, pages 613–622. Springer-Verlag.

[Diday et al., 1982] Diday, E., Lemaire, J., Pouget, J., et Testu, F. (1982). *Éléments d'analyse de données*. Dunod.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7) :1895–1924.

[Dietterich et al., 1997] Dietterich, T. G., Lathrop, R. H., et Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2) :31–71.

[Domingos, 1995] Domingos, P. (1995). Rule induction and instance-based learning : a unified approach. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1226–1232.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., et Colorni, A. (1996). The ant system : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1) :29–41.

[Drogoul et Dubreuil, 1993] Drogoul, A. et Dubreuil, C. (1993). A distributed approach to N-puzzle solving. In *Proc. 12th Int. Work. Distributed Artificial Intelligence*, pages 95–108.

[Emde et Wettschereck, 1996] Emde, W. et Wettschereck, D. (1996). Relational instance based learning. In Saitta, L., éditeur, *Proceedings 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann.

[Erdem et Flener, 1997] Erdem, E. et Flener, P. (1997). A redefinition of least generalizations and its application to inductive logic program synthesis. Rapport interne. non publié.

[Esposito et al., 1996] Esposito, F., Laterza, A., Malerba, D., et Semeraro, G. (1996). Refinement of datalog programs. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD)*, pages 73–94.

[Fayyad et al., 1993] Fayyad, U. M., Weir, N., et Djorgovski, S. (1993). SKICAT : A machine learning system for automated cataloging of large scale sky surveys. In Utgoff, P., éditeur, *Proceedings 10th International Conference on Machine Learning*, pages 112–119. Morgan Kaufmann.

[Feigenbaum, 1977] Feigenbaum, E. A. (1977). The art of artificial intelligence : Themes and case studies of knowledge engineering. In [Reddy, 1977], pages 1014–1029.

[Ferber, 1995] Ferber, J. (1995). *Les systèmes multi-agents - Vers une intelligence collective*. InterEditions.

[Fernández Ostolaza et Moreno Bergareche, 1997] Fernández Ostolaza, J. et Moreno Bergareche, A. (1997). *La vie artificielle*. Seuil.

[Fisher, 1987] Fisher, D. H. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2(2) :139,172.

[Frank et Witten, 1998] Frank, E. et Witten, I. H. (1998). Generating accurate rule sets without global optimization. In Shavlik, J., éditeur, *Proceedings 15th International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann.

[Garey et Johnson, 1979] Garey, M. et Johnson, D. (1979). *Computers and Intractability - A Guide to Theory of NP-Completeness*. Freeman, W.H. and Co., New York.

[Gennari et al., 1989] Gennari, J. H., Langley, P., et Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40 :11–61.

[Gold, 1967] Gold, E. M. (1967). Language Identification in the Limit. *Information and Control*, 10 :447–474.

[Goncalves, 1996] Goncalves, M.-E. (1996). Handling quantifiers in ILP. In Muggleton, S., éditeur, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 76–96. Stockholm University, Royal Institute of Technology.

[Goncalves, 1999] Goncalves, M.-E. (1999). *Vers des opérateurs d'apprentissage pour la logique du premier ordre - Étude de leurs propriétés de correction, complétude et minimalité*. Thèse de doctorat, Université Paris-Sud.

[Gottlob, 1987] Gottlob, G. (1987). Subsumption and implication. *Information Processing Letters*, 24(2) :109–111.

[Hogg et al., 1996] Hogg, T., Huberman, B. A., et Williams, C. P. (1996). Phase transitions and the search problem (editorial). *Artificial Intelligence*, 81(1–2) :1–15.

[Holland, 1975] Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. MIT Press.

[Holte et al., 1989] Holte, R. C., Acker, L. E., et Porter, B. W. (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 813–818.

[Idestam-Almquist, 1995] Idestam-Almquist, P. (1995). Generalization of clauses under implication. *Journal of Artificial Intelligence Research*, 3 :467–489.

[Kapur et Narendran, 1986] Kapur, D. et Narendran, P. (1986). Np-completeness of the set unification and matching problems. In *Proceedings of 8th Conference on Automated Deduction*, volume 230, pages 489–495. Springer-Verlag.

[Kietz et Wrobel, 1992] Kietz, J.-U. et Wrobel, S. (1992). Controlling the complexity of learning in logic through syntactic and task-oriented models. In Muggleton, S., éditeur, *Inductive Logic Programming*, pages 335–359. Academic Press.

[Kodratoff, 1989] Kodratoff, Y. (1989). Characterising machine learning programs, a european compilation. Rapport interne 507, Laboratoire de Recherche en Informatique, Université Paris Sud.

[Kuntz et al., 1997] Kuntz, P., Layzell, P., et D., S. (1997). A colony of ant-like agents for partitioning in vlsi technology. In Husbands, P. et Harvey, I., éditeurs, *Proceedings of the Fourth European Conference on Artificial Life*, pages 417–424. MIT Press.

[Lachiche, 1997] Lachiche, N. (1997). *De l'induction confirmatoire à la classification : contribution à l'apprentissage automatique*. Thèse de doctorat, Université Henri Poincaré - Nancy 1.

[Langley et al., 1987] Langley, P., Simon, H. A., Bradshaw, G. L., et M., Z. J. (1987). *Scientific Discovery : Computational Explorations of the Creative Process.* Machine Intelligence, eds : Meltzer, and Michie, vars. PublishersT Press. ACM CR 8807-0489.

[Levesque et Brachman, 1989] Levesque, H. J. et Brachman, R. J. (1989). A fundamental tradeoff in knowledge representation and reasoning (revised version). In Brachman et Levesque, éditeurs, *Readings in knowledge representation*, chapitre 4, pages 41–70. Morgan Kaufmann.

[Lloyd, 1987] Lloyd, J. W. (1987). *Foundations of Logic Programming.* Springer, Berlin, 2 edition.

[Marcinkowski et Pacholski, 1992] Marcinkowski, J. et Pacholski, L. (1992). Undecidability of the horn-clause implication problem. In IEEE, éditeur, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 354–362, Pittsburgh, PN. IEEE Computer Society Press.

[Martin, 1996] Martin, L. (1996). *Induction de programmes logiques avec négation.* Thèse de doctorat, Université d'Orléans.

[Merz et Murphy, 1996] Merz, C. J. et Murphy, P. M. (1996). UCI repository of machine learning databases.

[Michalski et al., 1986] Michalski, R., Mozetic, I., Hong, J., et Lavrac, N. (1986). The AQ15 inductive learning system : an overview and experiments. In *Proceedings of IMAL 1986*, Orsay. Université de Paris-Sud.

[Michalski, 1983] Michalski, R. S. (1983). A theory and methodology of inductive learning. In Michalski, R. S., Carbonell, J. G., et Mitchell, T. M., éditeurs, *Machine Learning : An Artificial Intelligence Approach*, volume I, pages 83–134, Palo Alto, CA. Tioga.

[Michie et al., 1994] Michie, D., Muggleton, S., Page, D., et Srinivasan, A. (1994). To the international computing community : A new East-West challenge. Rapport interne, Oxford University Computing laboratory, Oxford,UK.

[Minsky, 1988] Minsky, M. (1988). *La société de l'esprit.* InterEditions, Paris.

[Mitchell, 1977] Mitchell, T. M. (1977). Version spaces : a candidate elimination approach to rule learning. In [Reddy, 1977], pages 305–310.

[Mitchell, 1980] Mitchell, T. M. (1980). The need for biases in learning generalizations. In *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann. Published in 1991.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18 :203–226.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

[Muggleton, 1991] Muggleton, S. (1991). Inductive logic programming. *New Generation Computing Journal*, 8(4) :295–317.

[Muggleton, 1995] Muggleton, S. (1995). Inverse entailment and PROGOL. *New Generation Computing Journal*, 13 :245–286.

[Muggleton et De Raedt, 1994] Muggleton, S. et De Raedt, L. (1994). Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19 :629–679.

[Muggleton et Feng, 1990] Muggleton, S. et Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan.

[Muggleton et Buntine, 1988] Muggleton, S. H. et Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings 5th International Conference on Machine Learning*, pages 339–352, San Mateo, CA. Morgan Kaufmann.

[Nédellec et Rouveirol, 1994] Nédellec, C. et Rouveirol, C. (1994). Specifications of the HAIKU system. Rapport interne 928, Laboratoire de Recherche en Informatique, Université Paris Sud.

[Nédellec et al., 1996] Nédellec, C., Rouveirol, C., Adé, H., Bergadano, F., et Tausend, B. (1996). Declarative bias in ILP. In De Raedt, L., éditeur, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press.

[Niblett, 1993] Niblett, T. (1993). A note on refinement operators. In Brazdil, P. B., éditeur, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 329–335. Springer-Verlag.

[Nienhuys-Cheng et de Wolf, 1996a] Nienhuys-Cheng, S. et de Wolf, R. (1996a). Least generalizations and greatest specializations of sets of clauses. *Journal of Artificial Intelligence Research*, 4 :341–363.

[Nienhuys-Cheng et de Wolf, 1996b] Nienhuys-Cheng, S. et de Wolf, R. (1996b). Least Generalizations under Implication. In Muggleton, S., éditeur, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 262–363. Stockholm University, Royal Institute of Technology.

[Nienhuys-Cheng et de Wolf, 1997] Nienhuys-Cheng, S. et de Wolf, R. (1997). *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence Tutorial*. Springer-Verlag.

[Nok et O. Gascuel, 1995] Nok, R. et O. Gascuel, O. (1995). On learning decision committees. In Prieditis, A. et Russell, S., éditeurs, *Proceedings 12th International Conference on Machine Learning*, pages 413–420. Morgan Kaufmann.

[Pagallo et Haussler, 1990] Pagallo, G. et Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1) :71–99.

[Paschos, 1997] Paschos, V. T. (1997). A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2) :171–209.

[Plotkin, 1970] Plotkin, G. (1970). A note on inductive generalization. In Meltzer, B. et Mitchie, D., éditeurs, *Machine Intelligence*, volume 5, pages 153–165. Edinburgh University Press.

[Plotkin, 1971] Plotkin, G. (1971). A further note on inductive generalization. In *Machine Intelligence*, volume 6. Edinburgh University Press.

[Provost et Fawcett, 1997] Provost, F. et Fawcett, T. (1997). Analysis and visualization of classifier performance : Comparison under imprecise class and cost

distributions. In Heckerman, D., Mannila, H., Pregibon, D., et R., U., éditeurs, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48. AAAI Press.

[Provost et al., 1999] Provost, F., Jensen, D., et Oates, T. (1999). Efficient progressive sampling. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 23–32, San Diego, CA. ACM Press.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1) :81–106.

[Quinlan, 1990] Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3) :239–266.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

[Reddy, 1977] Reddy, R., éditeur (1977). *Proceedings of the 5th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

[Rouveirol, 1992] Rouveirol, C. (1992). Extensions of inversion resolution applied to theory completion. In Muggleton, S., éditeur, *Inductive Logic Programming*, chapitre 3. Academic Press, London.

[Sammut et Banerji, 1986] Sammut, C. et Banerji, R. (1986). Learning concepts by asking questions. In Michalski, R., Carbonell, J., et Mitchell, T., éditeurs, *Machine Learning : An Artificial Intelligence Approach*, volume 2, chapitre 7, pages 167–192. Morgan Kaufmann.

[Schaffer, 1994] Schaffer, C. (1994). A conservation law for generalization performance. In Cohen, W. W. et Hirsh, H., éditeurs, *Proceedings 11th International Conference on Machine Learning*, pages 259–265. Morgan Kaufmann.

[Scheffer et Herbrich, 1997] Scheffer, T. et Herbrich, R. (1997). Unbiased assessment of learning algorithms. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 798–803.

[Schmidt-Schauss, 1988] Schmidt-Schauss, M. (1988). Implication of clauses is undecidable. *TCS : Theoretical Computer Science*, 59(3) :287–296.

[Schölkopf et al., 1998] Schölkopf, B., Burgess, C., et Smola, A. (1998). *Advances in Kernel Methods*. MIP Press.

[Sebag, 1996] Sebag, M. (1996). Delaying the choice of bias : A disjunctive version space approach. In L., S., éditeur, *13th International Conference on Machine Learning (ICML'96)*, pages 444–452. Morgan Kaufmann.

[Sebag, 1997] Sebag, M. (1997). Distance induction in first order logic. In Raedt de, L., éditeur, *Proceedings of the 15th International Joint Conference on Artificial Intelligence, Workshop on Inductive Logic Programming*.

[Sebag, 1999a] Sebag, M., éditeur (1999a). *Actes de la Première Conférence d'Apprentissage*.

[Sebag, 1999b] Sebag, M. (1999b). Constructive induction : A version space-based approach. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

[Sebag et Rouveirol, 1997] Sebag, M. et Rouveirol, C. (1997). Tractable induction and classification in FOL via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–892.

[Shapiro, 1981] Shapiro, E. Y. (1981). Inductive inference of theories from facts. Rapport interne 192, Yale University Department of Computer Science.

[Srinivasan et al., 1999] Srinivasan, A., King, R., et Bristol, D. (1999). An assessment of submissions made to the predictive toxicology evaluation challenge. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 270–276. Morgan Kaufmann.

[Srinivasan et al., 1997] Srinivasan, A., King, R. D., Muggleton, S. H., et Sternberg, M. J. E. (1997). The predictive toxicology evaluation challenge. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 4–9. Morgan Kaufmann.

[Srinivasan et al., 1994] Srinivasan, A., Muggleton, S., King, R. D., et Sternberg, M. J. E. (1994). Mutagenesis : ILP experiments in a non-determinate biological domain. In Wrobel, S., éditeur, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH.

[Stahl et Tausend, 1994] Stahl, I. et Tausend, B. (1994). MILES - a modular inductive logic programming experimentation system. Rapport interne 6020 (ESPRIT BRA ILP).

[Tausend, 1994] Tausend, B. (1994). Representing biases for inductive logic programming. In Bergadano, F. et De Raedt, L., éditeurs, *European Conference on Machine Learning 94*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 427–430. Springer-Verlag.

[Torre, 1995] Torre, F. (1995). Exploitation de biais de langage en apprentissage symbolique automatique dans un cadre logique. Rapport de DEA.

[Torre, 1996] Torre, F. (1996). Utilisation de Grammaires en Programmation Logique Inductive. In Gascuel, O., éditeur, *11èmes Journées Françaises d'Apprentissage (JFA'96), Sète, France*, pages 317–320.

[Torre, 1999a] Torre, F. (1999a). GloBo : un algorithme stochastique pour l'apprentissage supervisé et non supervisé. In [Sebag, 1999a], pages 161–168.

[Torre, 1999b] Torre, F. (1999b). Les Vraizamis. In [Sebag, 1999a], pages 177–184.

[Torre et Rouveirol, 1997a] Torre, F. et Rouveirol, C. (1997a). Natural ideal operators in inductive logic programming. In van Someren, M. et Widmer, G., éditeurs, *Proceedings of the ninth European Conference on Machine Learning*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 274–289. Springer-Verlag.

[Torre et Rouveirol, 1997b] Torre, F. et Rouveirol, C. (1997b). Opérateurs naturels en programmation logique inductive. In Soldano, H., éditeur, *12èmes Journées Françaises d'Apprentissage (JFA'97)*, pages 53–64.

[Turing, 1950] Turing, A. M. (1950). Computing machinery and Intelligence. *Mind*, 59 :433–460.

[Utgoff, 1986] Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In Michalski, R. S., Carbonell, J. G., et Mitchell, T. M., éditeurs, *Machine Learning : An Artificial Intelligence Approach*, volume II, pages 107–148. Morgan Kaufmann.

[van der Laag et Nienhuys-Cheng, 1994a] van der Laag, P. et Nienhuys-Cheng, S. H. (1994a). A note on ideal refinement operators in ILP. In Wrobel, S., éditeur, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 247–262. Gesellschaft für Mathematik und Datenverarbeitung MBH.

[van der Laag, 1995] van der Laag, P. R. J. (1995). *An Analysis of Refinement Operators in Inductive Logic Programming*. Thèse de doctorat, Erasmus Universiteit, Rotterdam, the Netherlands.

[van der Laag et Nienhuys-Cheng, 1994b] van der Laag, P. R. J. et Nienhuys-Cheng, S. (1994b). Existence and nonexistence of complete refinement operators. In Bergadano, F. et de Raedt, L., éditeurs, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag.

[Vapnik, 1995] Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

[Wolpert et Macready, 1995] Wolpert, D. H. et Macready, W. G. (1995). No free lunch theorems for search. Rapport interne SFI-TR-95-02-010, The Santa Fe Institute.