Apprentissage Supervisé Relationnel par Algorithmes d'Évolution

Sébastien Augier

ORSAY
N° d'ordre : 6420

UNIVERSIT de PARIS-SUD
U.F.R. SCIENTIFIQUE D'ORSAY

THSE

prsente pour obtenir

Le TITRE de DOCTEUR EN SCIENCES
Spécialité : Informatique

PAR

M. Sébastien AUGIER

SUJET : **Apprentissage Supervisé Relationnel par Algorithmes d'Évolution**

Soutenue le 19 décembre 2000 devant la Commission d'examen

M. Didier Fayard (président)

M. Yves Kodratoff

M. Jacques Nicolas

M. Marc Schoenauer (rapporteur)

M. Gilles Venturini (rapporteur)

Abstract :

This thesis investigates the application of evolutionary-based algorithms to the problem of inducing relational rules from positive and negative examples.

We first study a language bias that allows enough expressivity to cover at the same time relational learning from interpretations, and classical propositional languages. Even though the induction cost for these languages is characterized by NP-completeness on the subsumption test, a practical solution suitable for real complex problems is proposed.

The $\mathcal{SIAO}1$ system, which uses this language bias to learn relational rules, is then presented. It is based on an evolutionary search strategy characterized by :

- mutation and crossover operators driven by background knowledge and learning examples ;
- a bottom-up search direction that respects the ordering relation defined over the language.

$\mathcal{SIAO}1$ is then compared to other systems on various classical machine learning bases, proving its polyvalency. These tests also show that this system compares well to the other approaches and that few learning and evaluation biases are demanded from the user.

The third part of this work concerns two generic parallel architectures derived from asynchronous master-slave and pipeline models. These architectures are studied from the point of view of their scalability (ie. dependance on the dataset size) and expected speed-up. A simple but accurate prediction model to forecast the performance of these architectures is also proposed.

Résumé :

Cette thse concerne l'apprentissage de rgles relationnelles partir d'exemples et de contre-exemples, l'aide d'algorithmes volutionnaires.

Nous tudions tout d'abord un biais de langage offrant une expressivit suffisamment riche pour permettre de couvrir la fois le cadre de l'apprentissage relationnel par interprtations et les formalismes propositionnels classiques. Bien que le cot de l'induction soit caractris par la complexit NP-difficile du test de subsomption pour cette classe de langages, une solution capable de traiter en pratique les problmes rels complexes est propose.

Le systme $\mathcal{SIAO}1$, qui utilise ce biais de langage pour l'apprentissage de rgles relationnelles est ensuite prsent. Il est fond sur une stratgie de recherche volutionnaire qui se distingue principalement des approches classiques par :

- des oprateurs de mutation et de croisement dirigs par la thorie du domaine et par les exemples d'apprentissage ;
- le respect de la relation d'ordre dfinie sur le langage.

L'valuation du systme sur plusieurs bases faisant rfrence en apprentissage automatique montre que $\mathcal{SIAO}1$ est polyvalent, se compare favorablement aux autres approches et sollicite peu l'utilisateur en ce qui concerne la spcification de biais de recherche ou d'valuation.

La troisime partie de ce travail propose deux architectures parallles gnriques derives des modles matre-esclave asynchrone et du pipeline. Elles sont tudies dans le cadre de l'extraction de connaissances partir de donnes l'aide de $\mathcal{SIAO}1$ du point de vue de l'acclration qu'elles procurent d'une part et de leur capacit changer d'chelle d'autre part. Un modle de prdiction simple mais prcis des performances de chacune des architectures parallles est galement propos.

Apprentissage Supervisé Relationnel par Algorithmes d'Évolution

Sébastien Augier

12 décembre 2000

Le Maître dit : Zigong, crois-tu que je sois quelqu'un qui étudie une masse de choses et qui les retient par cœur ?
L'autre répondit : En effet. N'en est-il pas ainsi ?
Nullement. J'ai un seul fil pour enfiler le tout.

(Entretiens de Confucius, XV. 3)

À ma famille.

Remerciements

Je remercie Yves Kodratoff qui a supervisé ce travail. Sa perception à la fois large et actualisée du domaine, sa disponibilité, son dynamisme et son humour ont été d'un grand soutien. Merci à Gilles Venturini, qui a su susciter en DEA mon interêt pour les algorithmes génétiques et dont une partie des travaux ont servi de point de départ à cette thèse. Ses remarques et ses encouragements ont été précieux. Merci également à Marc Schoenauer pour ses commentaires avisés, tant au niveau méthodologique que théorique, qui m'ont permis de prendre du recul par rapport à ce travail. Je remercie également Didier Fayard d'avoir accepté de présider le jury de cette thèse. Je me souviens avec plaisir de son interêt pour les nombreuses applications des algorithmes génétiques ainsi que des discussions informelles et enrichissantes partagées. Merci à Jacques Nicolas qui a bien voulu faire partie de ce jury et dont l'expertise en apprentissage et en programmation logique apporte un point de vision complémentaire sur ce travail.

Que soient particulièrement remerciés Céline Rouveirol, Michèle Sebag et Antoine Cornuéjols, pour leur grande disponibilité et leurs remarques au cours de la rédaction de cette thèse.

Pensées amicales aux collègues du LRI, de l'équipe I&A et d'EPCAD, en particulier Jacques Alès-Bianchetti, Erick Alphonse, Alain Brunie-Taton, Lola Canamero, David Faure, Nicolas Graner, Guillaume Leblanc, Jérôme Maloberti, Hervé Mignot, Caroline Ravisé et Fabien Torre, pour les nombreux bons moments partagés et les discussions animées.

Que soient finalement remerciés Tobias Scheffer qui a eu la gentillesse de fournir le code de son algorithme et les équipes techniques du LRI et du CRI.

Chapitre 1

Apprentissage Automatique Symbolique Supervisé

Ce chapitre introduit les principaux concepts et algorithmes relatifs à l'apprentissage automatique supervisé. À cause de la multitude des approches et des travaux réalisés, il ne vise pas à l'exhaustivité mais s'attache tout d'abord à situer le domaine à la fois dans une perspective historique qui le lie à l'intelligence artificielle ainsi que dans le contexte actuel qui est caractérisé par un développement très important des applications industrielles (systèmes d'aide à la décision, *data mining*, etc.). Le corps du chapitre alterne d'une part la présentation des principales problématiques et résultats théoriques (Qu'est-ce que l'induction ? Comment formuler le problème dans le cadre de l'informatique ? Quelles sont les propriétés des objets manipulés ?), et d'autre part les principaux algorithmes d'apprentissage ainsi que leurs caractéristiques. Une dernière partie est plus spécifiquement consacrée à la programmation logique inductive car ce domaine fournit un cadre plus précis et de nombreux résultats tant théoriques que pratiques, permettant de mieux situer notre travail.

1.1 Historique

L'apprentissage automatique (AA) est initialement apparu comme une composante de l'intelligence artificielle complétant et enrichissant les approches basées sur le modèle *intelligence = connaissances + raisonnement*. Par exemple, dans le cadre des systèmes experts, le goulot d'étranglement provoqué par la synthèse puis le codage des masses de connaissances nécessaires à une expertise fiable et détaillée d'un domaine est à l'origine dans un premier temps du développement d'outils d'aide à l'acquisition des connaissances, ces derniers ayant finalement été complétés par des systèmes capables de générer directement des règles de décision à partir de données brutes. Le tableau 1.1 met en évidence le gain apporté par l'utilisation d'outils d'apprentissage pour la conception et pour la maintenance des systèmes experts. Les coûts en années-hommes de ces deux postes pour les systèmes GASOIL et BMT, qui utilisent des techniques d'apprentissage automatique sont en effet inférieurs de plus d'un ordre de grandeur à ceux des systèmes-témoins MYCIN et XCON. Le développement de l'apprentissage, dont les applications pratiques concernent

Nom	Domaine d'application	Règles	Coût dév[a].	Coût main[a].	Outils d'AA
MYCIN	Diagnostic médical	400	100	ND	ND
XCON	Configuration de systèmes VAX	8.000	180	30	ND
GASOIL	Configuration de systèmes de séparation d'hydrocarbures	2.800	1	0,1	ExpertEase, Extran7
BMT	Configuration de systèmes pare-feux	30.000	9	2	1st Class, RuleMaster

[a]Les coûts de développement et de maintenance sont exprimés en années-hommes.

TAB. 1.1: Coût de développement et de maintenance de différents systèmes experts, suivant qu'ils mettent en œuvre ou non des techniques d'apprentissage automatique (source : [Muggleton92]).

un nombre croissant de domaines, est donc allé de pair avec l'évolution de la représentation de la notion d'intelligence, qui ne se limite plus à l'accumulation et au traitement d'un savoir encyclopédique immuable. Peu de chercheurs doutent en effet aujourd'hui qu'un système puisse être qualifié d'intelligent s'il n'est pas capable d'apprendre afin de se perfectionner et de s'adapter.

Une seconde motivation pour développer des algorithmes d'apprentissage, relative aux nombreuses applications directes de ce domaine en entreprise, s'est dégagée puis a prédominé depuis les années 90 [Kodratoff00]. L'apprentissage automatique s'est entre autres imposé comme l'une des techniques-clé du processus d'extraction des connaissances à partir des données (ECD, cf. figure 1.1), dont le développement peut s'expliquer par l'effet cumulatif d'un certain nombre de facteurs :

- Depuis les années 80, de nombreuses entreprises ont crée puis archivé des bases de données contenant de nombreuses informations relatives à leurs clients, leurs produits et leurs concurrents. De nombreuses institutions ont connu un mouvement semblable.

- Plus récemment, le développement des réseaux reliant les entreprises a accentué le rôle central des bases de données tout en décuplant les possibilités d'analyse (augmentation de la fiabilité des données à l'aide du recoupement et de l'enrichissement des bases d'une part, échanges de données structurées grâce, par exemple, au langage XML d'autre part).

- Dans ce contexte, certaines entreprises ou institutions ont construit une structure dédiée, appelée entrepôt de données (*data warehouse*). Ces structures sont des bases de données ayant pour fonction d'intégrer les informations orientées-sujet (utiles à la prise de décision), provenant de tous les secteurs de l'entreprise, de manière non volatile : les mises-à-jour ne détruisent pas les anciennes données afin de conserver un historique.

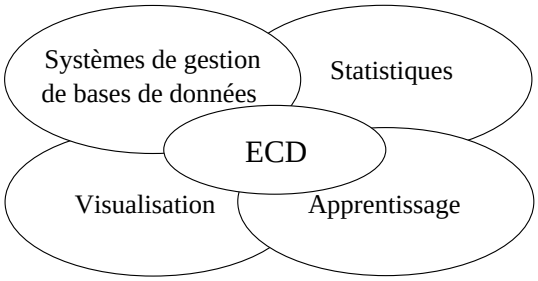


FIG. 1.1: Pluridisciplinarité de l'ECD.

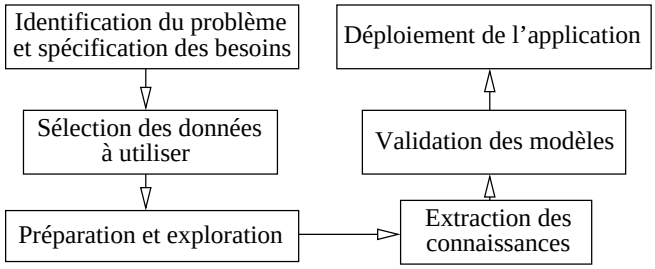


FIG. 1.2: Le processus d'ECD.

L'ECD [Kodratoff95] (appelé KDD en anglais pour *knowledge discovery in databases* [Frawley et al.91]) a la particularité de s'être construit de manière pragmatique en tant que processus s'intégrant au métier de l'entreprise et couvrant toutes les étapes allant de l'identification du problème jusqu'à l'exploitation de la solution (figure 1.2). L'étape d'extraction des connaissances, qui met en jeu des techniques d'apprentissage automatique, est plus connue dans le monde industriel sous le nom de *data mining*.

1.2 Poser le problème

1.2.1 Problématique générale

L'apprentissage au sens large dénote la capacité d'un agent à améliorer ses performances par l'expérience. Plus précisément, cette thèse se situe dans le cadre de l'apprentissage

Automatique : l'agent est une machine exécutant un programme. En conséquence, l'apprentissage peut être modélisé par un algorithme.

Supervisé : l'expérience est constituée de descriptions de situations (instances) classées par un professeur. L'ensemble de ces instances classées forme la base d'apprentissage, et l'agent est évalué sur sa capacité à classer correctement de nouvelles instances formant la base de test.
Par opposition, on parle d'apprentissage non supervisé lorsque l'agent doit regrouper les instances en des classes qu'il aura lui-même élaborées. Une telle tâche est typiquement réalisée à l'aide d'une mesure de distance entre les instances, appelée mesure de similarité. L'un des objectifs de l'agent est alors de trouver une partition minimisant la distance entre les instances d'une même classe et maximisant la distance entre les instances de classes différentes.
Ainsi, dans le domaine du marketing, l'opération consistant à caractériser les cibles les plus réceptives à une campagne de publipostage à l'aide de données décrivant le comportement de prospects soumis à une pré-étude relève de la problématique de l'apprentissage supervisé, alors que l'utilisation d'un outil de classification afin de segmenter cette même population de prospects relève de l'apprentissage non supervisé.

Symbolique : l'agent doit produire des connaissances intelligibles pour l'utilisateur, par exemple sous forme de règles. Contrairement aux réseaux neuromimétiques ou au raisonnement à partir de cas, qui représentent la connaissance respectivement par un ensemble de poids associés à une architecture d'une part, et par une base de cas contenant de nombreux exemples d'autre part, l'adjectif symbolique fait référence à des connaissances concises ayant du sens pour un expert du domaine d'application, que ces connaissances mettent en jeu des objets purement symboliques ou également des données numériques.

1.2.2 Structure du processus de résolution

L'apprentissage automatique symbolique supervisé sera noté AS par la suite, l'abréviation AA faisant référence au contexte général de l'apprentissage automatique. La figure 1.3

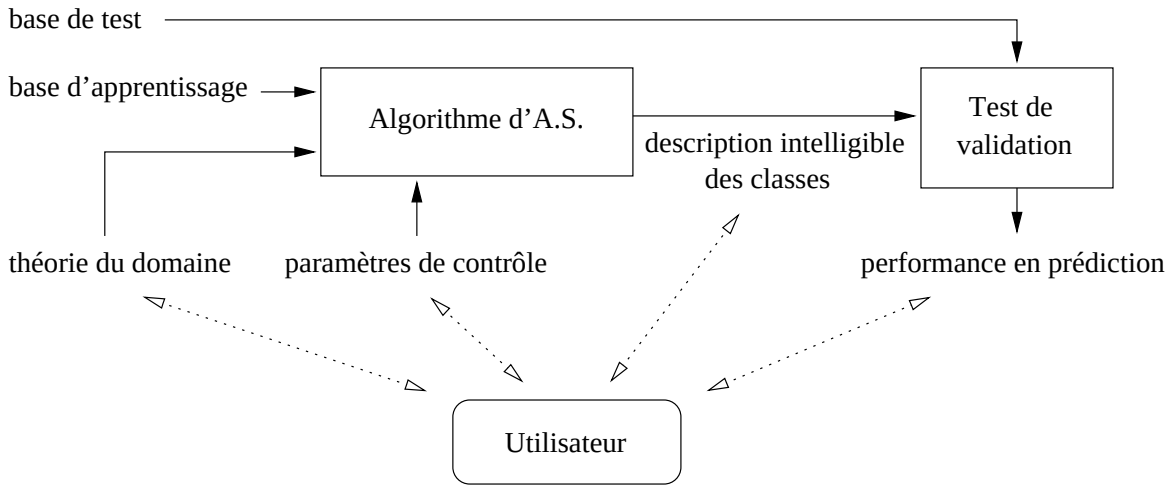


FIG. 1.3: Utilisation d'un algorithme d'AS.

schématise le contexte d'utilisation d'un algorithme d'AS. Les données fournies en entrée sont :

La base d'apprentissage, notée $\mathcal{B}$, qui contient la description des exemples (les instances classées) dans un langage $\mathcal{L}_e$. Lorsque $\mathcal{L}_e$ est un langage d'ordre 0, les variables ou attributs décrivant les instances sont dits endogènes, et ceux caractérisant la classe (ou concept-cible) sont dits exogènes.

La théorie du domaine, fournie par l'utilisateur, qui complète la description des exemples par des connaissances spécifiques au domaine d'application. Par exemple, certains algorithmes permettent la spécification de hiérarchies de concepts correspondant aux valeurs d'attributs symboliques ; ainsi dans le domaine financier, l'attribut symbolique `secteur_d_activité` pourrait être enrichi par une hiérarchie du type :

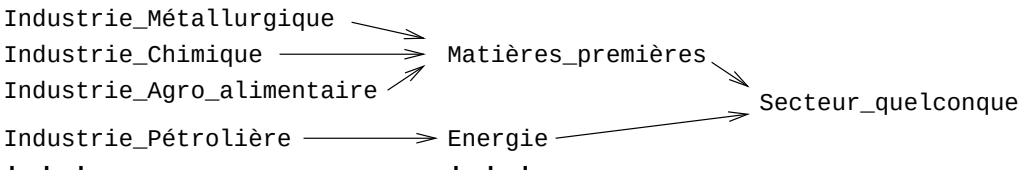


FIG. 1.4: Un exemple de hiérarchie sur des constantes.

Les paramètres de contrôle, également fournis par l'utilisateur, permettent à ce dernier, lorsqu'il en modifie la valeur par défaut, d'adapter l'algorithme à ses besoins ou de fournir des connaissances sur la nature du concept qui seront exploitées lors de la recherche de la description optimale des classes. Spécifier une durée d'apprentissage maximale, favoriser l'obtention de règles courtes et générales ou au contraire plus longues mais plus précises, donner une estimation du bruit des données, fixer une performance en terme de capacité prédictive à atteindre, ou encore interdire la production de règles faisant intervenir plus de 10 attributs sont autant d'exemples de paramètres de contrôle.

En sortie, l'algorithme d'AS retourne une description intelligible des classes. Cette description, appelée également hypothèse, est exprimée dans un langage $\mathcal{L}_h$ (habituellement $\mathcal{L}_e \subset \mathcal{L}_h$), et est la plupart du temps accompagnée par une mesure de sa précision sur l'ensemble d'apprentissage.

Un test de validation de l'hypothèse est alors réalisé à l'aide d'une base de test comportant des instances distinctes de celles qui apparaissent dans la base d'apprentissage afin d'obtenir une estimation de la capacité prédictive des connaissances apprises. Les méthodes de validation les plus courantes sont [Kohavi95] :

La validation simple (holdout), faite en partitionnant l'ensemble des n instances en un ensemble d'apprentissage et un ensemble de test. Il est conseillé d'utiliser environ $\frac{2.n}{3}$ instances pour l'apprentissage et les $\frac{n}{3}$ instances restantes pour le test.

La validation itérée (random subsampling) procède à un nombre $k > 1$ arbitraire de validations simples et rend comme estimation la moyenne des taux d'erreur observés. La variance des taux d'erreur peut être utilisée pour estimer la stabilité de l'algorithme d'apprentissage.

La validation par extraction (leave-k-out) consiste à choisir la taille $k > 0$ de l'ensemble de test, puis à générer la totalité des C_n^k ensembles de test possibles ainsi que les ensembles d'apprentissage associés. En pratique seule la méthode *leave-one-out* qui nécessite n apprentissages est utilisée : l'explosion combinatoire rend la validation par extraction inutilisable sur des ensembles d'apprentissage de taille normale dès $k = 2$.

La validation croisée (cross-validation) reprend le principe de la validation par extraction, tout en diminuant drastiquement sa complexité. L'ensemble des instances est divisé en $k > 1$ partitions de tailles équivalentes, puis k apprentissages sont réalisés sur les k unions différentes possibles de $(k-1)$ des partitions, la partition restante servant d'ensemble de test. La valeur $k = 10$ est très couramment utilisée.

La validation croisée stratifiée (stratified cross-validation) est une variation de la validation croisée qui s'attache à conserver dans chaque partition une répartition des classes équivalente à celle qui est observée sur l'ensemble des instances.

En supposant fixés une fois pour toutes le test de validation ainsi que les bases d'apprentissage et de test, le rôle de l'utilisateur consiste à :

- Évaluer la description fournie par l'algorithme d'AS du point de vue de sa valeur intrinsèque en tant que connaissance d'une part, et de son pouvoir prédictif via l'estimation de sa précision par le test de validation d'autre part.
- Ajuster, lorsque cela est nécessaire, la théorie du domaine et éventuellement les paramètres de contrôle. Bien que cet aspect soit le plus souvent passé sous silence, le fait que les algorithmes d'AS fassent en pratique partie d'un processus d'interaction avec l'utilisateur implique la nécessité de pouvoir adapter l'algorithme aux problématiques rencontrées : des caractéristiques telles que la tolérance à l'erreur ou la notion d'intelligibilité ne peuvent être déterminées dans l'absolu.

1.2.3 Apprentissage supervisé et optimisation

Reformulation du problème

La description de l'AS faite ci-dessus peut s'énoncer comme un problème d'optimisation.

Définition 1.1 *Problème d'AS (sous forme de problème d'optimisation)*

Étant donnés :

- *Un ensemble $\mathcal{B} = \mathcal{B}^+ \cup \mathcal{B}^-$, appelé base d'apprentissage, comportant des exemples ($\mathcal{B}^+$) et contre-exemples ($\mathcal{B}^-$) du concept-cible $\mathcal{C}$ représentés dans le langage $\mathcal{L}_e$;*
- *Un espace de recherche correspondant à l'ensemble des formules définies sur $\mathcal{L}_h$;*
- *Une fonction d'évaluation $f_{\mathcal{B}} : \mathcal{L}_h \longrightarrow \Re$, qui, à toute hypothèse h de l'espace de recherche, associe un réel $f_{\mathcal{B}}(h)$ représentant le score de cette hypothèse sur la base d'apprentissage $\mathcal{B}$.*

Trouver l'hypothèse $\mathcal{H}$ telle que $f_{\mathcal{B}}(\mathcal{H})$ soit maximale.

Bien que l'analyse de la problématique de l'AS permette la mise en évidence d'un grand nombre de caractéristiques spécifiques à ce domaine, la reformulation sous forme d'un problème d'optimisation est bénéfique pour au moins deux raisons :

1. Sur le plan pratique, certaines techniques développées dans ce domaine, qui bénéficie de nombreuses années de recherche, peuvent être adaptées à l'AS.
2. Sur le plan théorique, des avancées récentes concernant les algorithmes de recherche et d'optimisation ont des répercussions très importantes quant à la perception même de la tâche à accomplir et des résultats que l'on peut espérer atteindre.

Le théorème de conservation

Un exemple récent de résultat théorique fondamental est fourni par les théorèmes de conservation (*no free lunch theorems*) relatifs aux méthodes de recherche et d'optimisation [Wolpert et al.96b, Wolpert et al.96a] dont le résultat central est résumé ci-dessous :

Théorème 1.1 *Théorème de conservation*
Soit un problème d'optimisation défini à l'aide d'une fonction $f : \mathcal{X} \to \mathcal{Y}$ *dont on cherche à déterminer* $x^* \in \mathcal{X}$ *tel que* $y^* = f(x^*) \in \mathcal{Y}$ *ait une propriété donnée (typiquement que cette valeur soit maximale ou minimale). Supposons de plus :*

1. *Que* $\mathcal{X}$ *et* $\mathcal{Y}$ *sont finis (c'est toujours le cas pour les problèmes traités par des machines digitales), de cardinalités respectives* $|\mathcal{X}|$ *et* $|\mathcal{Y}|$. $\mathcal{F} = \mathcal{Y}^{\mathcal{X}}$, *de cardinal* $|\mathcal{Y}|^{|\mathcal{X}|}$, *qui représente l'ensemble des fonctions* f *possibles, est alors également fini.*
2. *Que l'algorithme d'optimisation examine, à chaque étape, une (ou plusieurs) nouvelle(s) solution(s) dans l'espace* $\mathcal{X}$ *uniquement en fonction des solutions précédemment rencontrées et de leurs coûts. On note alors* $d_m = \{(d_m^x(1), d_m^y(1)), ..., (d_m^x(m), d_m^y(m))\}$ *l'ensemble ordonné des* m *solutions distinctes examinées et* d_m^x *(resp.* d_m^y*) la restriction de* d_m *aux instances (resp. aux coûts)*[1].
3. *Que la mesure de la performance d'un algorithme* a *après un nombre* m *quelconque d'itérations (au sens de* d_m*) est une fonction de* d_m^y, *notée* $\Phi(d_m^y)$. *Par exemple, dans le cas où l'on cherche* $x^* \in \mathcal{X}$ *tel que* $f(x^*)$ *soit maximal, on choisira* $\Phi(d_m^y) = max(\{d_m^y(i)/i \in [1..m]\})$.

Alors, en notant $P(d_m^y|f, m, a)$ *la probabilité conditionnelle d'obtenir l'ensemble ordonné* d_m^y *pour le problème d'optimisation* f *soumis à l'algorithme* a *durant* m *itérations (évaluations distinctes), on* a, *quels que soient les algorithmes d'optimisation* a_1 *et* a_2 *:*

$$\sum_{f \in \mathcal{F}} P(d_m^y|f, m, a_1) = \sum_{f \in \mathcal{F}} P(d_m^y|f, m, a_2)$$

et donc, pour toute mesure de performance $\Phi(d_m^y)$, *la valeur moyenne de* $P(\Phi(d_m^y)|f, m, a)$ *pour toutes les fonctions* $f \in \mathcal{F}$ *est indépendante de* a.

Ce résultat a été étendu aux cas plus généraux des fonctions qui varient dans le temps et des algorithmes d'optimisation stochastiques. Il concerne donc l'ensemble des algorithmes d'optimisation fonctionnant en boîte noire , c'est-à-dire pour lesquels seule la phase d'exploration apporte de l'information (les $f \in \mathcal{F}$ sont équiprobables : $\forall f \in \mathcal{F}, P(f) =$

[1] Cette hypothèse amène deux remarques :

(a) Le théorème s'applique également aux algorithmes pouvant évaluer plusieurs fois une instance $x \in \mathcal{X}$ donnée (comme par exemple les algorithmes de montée (*hill-climbing*), les algorithmes génétiques, le recuit simulé, etc.) ; cependant il ne comptabilise que les solutions distinctes, dans l'ordre dans lequel elles sont examinées.

(b) La liste d_m comptabilise le nombre d'évaluations d'instances : par exemple, dans le cas d'un algorithme de montée (*hill-climbing*), elle comprend non seulement les points de la suite ascendante parcourue, mais également tous leurs voisins qui ont aussi dû être évalués.

$\frac{1}{|\mathcal{Y}|^{|X|}}$) ; en particulier aucune information relative à la structure de l'espace de recherche ou de la solution n'est disponible. La conservation de la performance moyenne implique que ce qu'un algorithme a_1 gagne en performance Φ sur une classe de problèmes par rapport à un algorithme a_2 est obligatoirement perdu sur une autre classe de problèmes.

En particulier, les algorithmes génétiques, le recuit simulé, les algorithmes de montée (*hill-climbing*), de descente (*hill-descending*) et la recherche aléatoire ont, sur l'ensemble de tous les problèmes, une performance moyenne identique. Sur le plan pratique, le fait qu'aucun algorithme d'optimisation ne soit meilleur qu'un autre dans l'absolu a pour conséquence :

- qu'il est nécessaire d'identifier les classes de problèmes sur lesquelles un algorithme donné obtient de meilleures performances ;
- que toute connaissance relative à la structure de l'espace de recherche devra être exploitée autant que possible.

1.3 Stratégies de résolution

Les sections précédentes ont été l'occasion de présenter le contexte du travail, de poser le problème à résoudre et de souligner le lien pouvant être établi avec le domaine très vaste de l'optimisation. Cette section est consacrée aux algorithmes d'AS et a pour double objectif :

- de montrer la spécificité de la problématique de l'apprentissage, et d'énoncer un théorème de conservation relatif à ce cadre ;
- de présenter quelques algorithmes représentatifs des principales approches du domaine.

1.3.1 Spécificité du problème

Une problématique liée à l'induction

La principale spécificité d'un problème d'AS par rapport à un problème de recherche ou d'optimisation classique provient du principe mis en œuvre pour le résoudre : l'induction. D'un point de vue logique, l'induction est une règle de transformation permettant d'inférer à partir d'une observation une loi plus générale :

$$\frac{\exists x\ A(x) \Rightarrow B(x)}{\forall x\ A(x) \Rightarrow B(x)} \left(\begin{array}{c} \textit{inférence} \\ \textit{inductive} \end{array} \right) \qquad \frac{\forall x\ A(x) \Rightarrow B(x)\ ;\ A(c)}{B(c)} \left(\begin{array}{c} \textit{inférence} \\ \textit{déductive} \end{array} \right)$$

Parce qu'elle ne conserve pas la correction (c'est-à-dire qu'il est possible de généraliser une observation correcte en une théorie fausse), l'induction n'est pas autant étudiée en logique que la déduction, qui consiste, étant données une théorie et une observation, à inférer les faits qui en découlent. Cependant, si la déduction est à la base des systèmes de preuve (par

exemple dans le mécanisme du chaînage avant mis en œuvre dans les systèmes experts), elle ne permet que l'exploitation des théories, et non leur création. La non-correction de la règle d'induction met en évidence le risque lié à la généralisation d'observations lors de la construction d'un modèle. Ce risque est inhérent à la démarche : apprendre, c'est générer un modèle expliquant non seulement les observations, mais pouvant être également appliqué à des instances inconnues.

En conséquence, le saut inductif est une caractéristique de l'apprentissage supervisé et ne peut être évité. En particulier :

- Lorsque $\mathcal{B}$ contient l'ensemble de toutes les instances possibles classées, on parle de compression des données et non d'apprentissage supervisé. En pratique, ce cas n'est jamais rencontré à cause de la combinatoire mise en jeu dans $\mathcal{L}_e$ (par exemple, dans le cas attribut-valeur, la connaissance exhaustive d'un problème d'apprentissage défini sur a attributs de modalités $m_1, ..., m_a$ nécessite $\prod_{i=1}^{a} m_i$ observations).
- Un algorithme se contentant de mémoriser $\mathcal{B}$ et ne classant que les instances déjà connues est un système de requêtes sur une base de données et non un algorithme d'apprentissage supervisé. Ce cas ne doit pas être confondu avec celui des algorithmes mémorisant $\mathcal{B}$ et classant de nouvelles instances à l'aide d'une mesure de similarité puisque la mesure de similarité correspond à une composante d'apprentissage réalisant une opération de généralisation.

Validation du saut inductif

La conservation de la correction lors du saut inductif est vérifiée lors de l'apprentissage à l'aide de la fonction $f_{\mathcal{B}}$ qui évalue la couverture des hypothèses candidates sur les instances d'apprentissage[2]. La figure 1.5 montre la vision ensembliste associée à l'évaluation

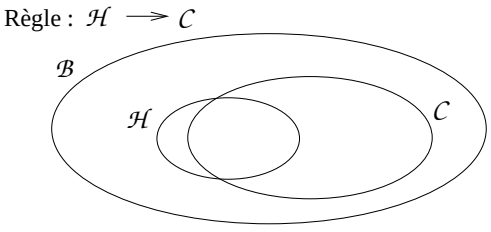


FIG. 1.5: Diagramme pour $\mathcal{H} \rightarrow \mathcal{C}$.

	$\mathcal{C}$	$\bar{\mathcal{C}}$	
$\mathcal{H}$	$n_{\mathcal{C}\wedge\mathcal{H}}$	$n_{\bar{\mathcal{C}}\wedge\mathcal{H}}$	$n_{\mathcal{H}}$
$\bar{\mathcal{H}}$	$n_{\mathcal{C}\wedge\bar{\mathcal{H}}}$	$n_{\bar{\mathcal{C}}\wedge\bar{\mathcal{H}}}$	$n_{\bar{\mathcal{H}}}$
	$n_{\mathcal{C}}$	$n_{\bar{\mathcal{C}}}$	$n_{\mathcal{B}}$

FIG. 1.6: Tableau de contingence $(\mathcal{H},\mathcal{C})$.

d'une hypothèse $\mathcal{H}$ caractérisant des instances d'un concept $\mathcal{C}$ dans l'ensemble des exemples d'apprentissage $\mathcal{B}$. Les effectifs peuvent être représentés à l'aide d'un tableau de contingence (figure 1.6). En particulier, les deux notions suivantes sont centrales pour toute validation d'une hypothèse obtenue par induction :

[2]L'hypothèse $\mathcal{H}$ couvre l'instance I de la base d'apprentissage $\mathcal{B}$ si $I \in \mathcal{H}$

Définition 1.2 *Cohérence logique*
L'hypothèse $\mathcal{H}$ est cohérente par rapport à la base d'apprentissage $\mathcal{B}$ ssi $\mathcal{H}$ ne couvre aucun contre-exemple de $\mathcal{B}$. On dit également que $\mathcal{H}$ est correcte par rapport à $\mathcal{B}$.

Définition 1.3 *Complétude logique*
L'hypothèse $\mathcal{H}$ est complète par rapport à la base d'apprentissage $\mathcal{B}$ ssi $\mathcal{H}$ couvre tous les exemples de $\mathcal{B}$.

Il est parfois préférable de substituer aux définitions booléennes ci-dessus (qui proviennent d'un cadre purement logique) des définitions plus graduées :

Définition 1.4 *Taux de cohérence*
Le taux de cohérence de l'hypothèse $\mathcal{H}$ par rapport à la base d'apprentissage $\mathcal{B}$ est défini par rapport à la couverture de $\mathcal{H}$ et vaut $\frac{n_{\mathcal{C}\wedge\mathcal{H}}}{n_{\mathcal{H}}}$. Le taux de cohérence absolu vaut : $1-\frac{n_{\bar{\mathcal{C}}\wedge\mathcal{H}}}{n_{\bar{\mathcal{C}}}}$.

Définition 1.5 *Taux de complétude*
Le taux de complétude de l'hypothèse $\mathcal{H}$ par rapport à la base d'apprentissage $\mathcal{B}$ est défini par : $\frac{n_{\mathcal{C}\wedge\mathcal{H}}}{n_{\mathcal{C}}}$.

Les taux de complétude et de cohérence d'une hypothèse sont toujours dans l'intervalle $[0, 1]$. Ils peuvent être utilisés dans des heuristiques par les algorithmes d'AS.

La validation du saut inductif par évaluation de la couverture de $\mathcal{H}$ sur $\mathcal{B}$ n'est pas une démarche triviale. En effet, elle repose sur l'hypothèse (implicite dans les systèmes d'AS) que la base d'apprentissage $\mathcal{B}$ est représentative du concept-cible recherché (la loi de conservation de la précision en généralisation, décrite plus bas, sera l'occasion d'expliciter cette notion). Plus précisément, l'hypothèse justifiant un raisonnement inductif peut être formulée de la manière suivante :

> Toute hypothèse $\mathcal{H}$ dont la couverture dans l'ensemble $\mathcal{B}$ est suffisamment voisine du concept-cible $\mathcal{C}$, approximera $\mathcal{C}$ assez précisément sur les instances inconnues.

De ce point de vue, il est possible de considérer l'évaluation du modèle fourni par l'algorithme d'AS sur la base de test comme une validation *a posteriori* de l'hypothèse d'induction pour un couple (algorithme, base d'apprentissage) donné.

1.3.2 Application du théorème de conservation

Le théorème de conservation énoncé dans le cadre général des algorithmes d'optimisation ne peut s'appliquer tel quel au domaine de l'apprentissage supervisé puisque la base $\mathcal{B}$ fournit des informations sur le problème à optimiser. Comme le montre le paragraphe ci-dessous, le périmètre de la notion de conservation doit être redéfini afin de ne prendre en compte que les sous-ensembles de $\overline{\mathcal{B}}$: on parle alors de conservation en généralisation.

Une loi de conservation dans le cadre de l'apprentissage supervisé

Cullen Schaffer [Schaffer94] a proposé un résultat très similaire au théorème de conservation dans le cadre de l'apprentissage. Ce résultat concerne l'apprentissage supervisé à partir d'exemples positifs et négatifs décrits dans un langage attribut-valeur. Les hypothèses sont très similaires à celles énoncées dans le cadre plus général des problèmes d'optimisation (prise en compte de l'ensemble de tous les problèmes possibles, équiprobabilité et dénombrabilité de ces derniers, déterminisme de l'algorithme par rapport à ses entrées). Les mesures choisies sont d'une part la précision en généralisation, c'est-à-dire le ratio du nombre d'instances de la base de test correctement classées par l'algorithme d'apprentissage sur le nombre total d'instances de la base de test, et d'autre part la performance en généralisation qui est définie comme la précision en généralisation moins $\frac{1}{2}$.

Théorème 1.2 *Loi de conservation de la précision en généralisation*
La performance en généralisation moyenne d'un algorithme d'AS sur l'ensemble des problèmes d'apprentissage est nulle, et sa précision en généralisation est égale à $\frac{1}{2}$.

Ce résultat est illustré intuitivement de la manière suivante : considérons un problème d'apprentissage défini par le couple $(\mathcal{B}, \mathcal{T})$, avec (par définition) $\mathcal{B} \cap \mathcal{T} = \emptyset$. Supposons alors $\mathcal{B}$ fixée : l'algorithme d'AS, dont les entrées sont constantes, fournit toujours la même hypothèse $\mathcal{H}$. Par conséquent, pour une base d'apprentissage $\mathcal{B}$ donnée, la classification des instances de la base de test $\mathcal{T}$ est fixée. Soient $t_1, ..., t_m$ les m instances de $\mathcal{T}$, $\mathcal{C}(t_1), ..., \mathcal{C}(t_m)$ leurs classes respectives et $\mathcal{H}(t_1), ..., \mathcal{H}(t_m)$ les classes prédites par $\mathcal{H}$. Parmi les 2^m étiquetages possibles des m instances de test, la moitié exactement est telle que $\mathcal{C}(t_i) = \mathcal{H}(t_i)$, et l'autre moitié telle que $\mathcal{C}(t_i) \neq \mathcal{H}(t_i)$. De plus, comme par hypothèse tous les étiquetages des instances de $\mathcal{T}$ sont équiprobables, la précision en généralisation moyenne de $\mathcal{H}$ est égale à $\frac{1}{2}$ (d'où l'on déduit également que la performance moyenne en généralisation est nulle).

De la représentativité des données

La loi de conservation de la précision en généralisation doit-elle faire perdre tout espoir de pouvoir apprendre des concepts à partir d'exemples ? Bien entendu, la réponse, comme dans le cas des problèmes d'optimisation où de nombreux succès ont été obtenus sur des problèmes réels en dépit du théorème de conservation, est négative. En effet, certaines hypothèses sont éloignées des problèmes réels puisqu'elles postulent :

- l'équiprobabilité des univers possibles ;
- l'absence de connaissances spécifiques au problème à résoudre.

Afin d'illustrer cette différence, considérons par exemple le problème d'apprentissage binaire suivant défini sur m attributs également binaires $A_1, ..., A_m$ (on suppose $m > 0$

assez grand) :

	A_1	A_2	...	A_{m-1}	A_m	$\mathcal{C}$
	0	0	...	0	0	0
	⋮				⋮	⋮
$\mathcal{B}$	0	1	...	1	1	0
	1	0	...	0	0	1
	⋮				⋮	⋮
	1	1	...	1	0	1
$\mathcal{T}$	1	1	...	1	1	$\mathcal{C}_\mathcal{T}$?

Du point de vue théorique, le théorème de conservation dit qu'en l'absence de connaissances sur le problème et l'univers, il n'est pas possible de décider la valeur de $\mathcal{C}_\mathcal{T}$ puisque les événements $(\mathcal{C}_\mathcal{T} = 0)$ et $(\mathcal{C}_\mathcal{T} = 1)$ sont équiprobables. La probabilité d'erreur, quel que soit le choix fait pour $\mathcal{C}_\mathcal{T}$, est donc de $\frac{1}{2}$.

D'un point de vue empirique, les tables de contingence relatives aux différents attributs sont les suivantes :

Table pour A_1 :

	$\mathcal{C}$	$\bar{\mathcal{C}}$	
A_1	$2^{m-1}-1$	0	$2^{m-1}-1$
$\bar{A}_1$	0	2^{m-1}	2^{m-1}
	$2^{m-1}-1$	2^{m-1}	2^m-1

Table pour A_i, $i \neq 1$:

	$\mathcal{C}$	$\bar{\mathcal{C}}$	
A_i	$2^{m-2}-1$	2^{m-2}	$2^{m-1}-1$
$\bar{A}_i$	2^{m-2}	2^{m-2}	2^{m-1}
	$2^{m-1}-1$	2^{m-1}	2^m-1

Il ressort donc qu'une dépendance entre $\mathcal{C}$ et A_1 est *a priori* très hautement probable, et que toutes les dépendances entre $\mathcal{C}$ et $A_i, (i \neq 1)$ sont *a priori* très hautement improbables. Par conséquent, on choisit (avec d'autant plus de certitude que m est grand) le modèle $\mathcal{C} = A_1$, d'où l'on déduit que $\mathcal{C}_\mathcal{T} = 1$.

Cet exemple illustre le fait que le théorème de conservation de la généralisation prédictive, qui pose que tous les univers sont possibles et équiprobables va à l'encontre de l'hypothèse de base de l'induction qui pose que tout modèle satisfaisant sur $\mathcal{B}$ sera satisfaisant sur $\mathcal{T}$, ou encore que :

$\mathcal{B}$ est représentative de l'univers par rapport à l'ensemble des modèles considérés .

Pour décider de la valeur de $\mathcal{C}_\mathcal{T}$, l'approche empirique simple présentée ci-dessus procède par un certain nombre d'étapes dont on va décider qu'elles sont raisonnables dans le cadre de la classe des problèmes réels :

- la première étape consiste à poser implicitement l'hypothèse qu'à performance identique sur $\mathcal{B}$, les modèles les plus simples doivent être préférés ;

- la seconde étape concerne l'établissement d'un ordre d'évaluation des différents modèles, d'où l'observation, dans un premier temps, des tables de contingence qui correspondent aux dépendances $\mathcal{C} = A_i$ et $\mathcal{C} = \neg A_i$;
- La troisième étape consiste à choisir un critère d'évaluation de la qualité des modèles (par exemple le test du χ^2 entre les tables des A_i et les tables correspondant aux modèles $\mathcal{C} = A_i$ et $\mathcal{C} = \neg A_i$) ;
- la dernière étape consiste à inférer à l'aide du modèle obtenu la valeur de $\mathcal{C}_\mathcal{T}$.

Chacune de ces étapes, qui précise un peu plus la classe des modèles contenant la définition recherchée du concept-cible, contredit l'hypothèse d'équiprobabilité des univers possibles posée par le théorème de conservation. En effet, elles introduisent implicitement des connaissances permettant de guider le choix d'un modèle, en préférant telle solution à telle autre. Ces préférences, appelées biais, sont étudiées plus en détail à la section 1.4.

1.3.3 Structure de l'espace de recherche

La notion de couverture des éléments de $\mathcal{L}_e$ permet de définir une relation de généralité entre les hypothèses :

Définition 1.6 *Relation de généralité*
Soient h_1 et h_2 deux hypothèses. Soit H_1 (resp. H_2) l'ensemble des éléments de $\mathcal{L}_e$ couverts par h_1 (resp. h_2). L'hypothèse h_1 est plus générale que l'hypothèse h_2 (noté $h_1 \geq_g h_2$) si et seulement si tout élément de H_2 est également élément de H_1.

La relation de généralité est une relation d'ordre large (elle est réflexive, antisymétrique et transitive). De plus, puisqu'il est possible d'avoir $H_1 \backslash H_2 \neq \emptyset$ et $H_2 \backslash H_1 \neq \emptyset$ (ie $h_1 \not\geq_g h_2$ et $h_2 \not\geq_g h_1$), la relation d'ordre est partielle. On note $h_1 >_g h_2$ (*h_1 est strictement plus générale que h_2*) le fait que $h_1 \geq_g h_2$ avec $h_2 \not\geq_g h_1$. De même, il est possible de définir les notions duales plus spécifique que et strictement plus spécifique que .

Le rôle central de la relation de généralité vient du fait qu'elle structure l'espace de recherche $\mathcal{L}_h$ indépendamment des exemples d'apprentissage. Habituellement, $\geq_g$ possède un unique plus grand élément (l'hypothèse qui couvre tous les exemples) et plusieurs plus petits éléments (appartenant à $\mathcal{L}_e$) : l'espace a donc une structure de sup-demi treillis (la définition mathématique peut être trouvée en annexe A). À titre d'exemple, la figure 1.7 décrit le sup-demi treillis relatif aux langages $\mathcal{L}_e$ (défini à l'aide de deux attributs A et B à valeur dans $\{a1, a2\}$ et $\{b1, b2\}$, respectivement) et $\mathcal{L}_h$ (qui introduit le symbole $*$ couvrant toutes les valeurs de l'attribut auquel il s'applique). Lorsqu'un plus petit élément est choisi comme base de la structure (par exemple $(a1, b1)$), la restriction obtenue est un treillis (représenté en gras).

Cette structure peut être mise à profit pour guider les algorithmes de recherche et leur permettre d'élaguer des régions non pertinentes de $\mathcal{L}_h$. Il est cependant nécessaire de prendre garde, lors de l'implantation de tout algorithme, à la complexité du test $\geq_g$ (qui

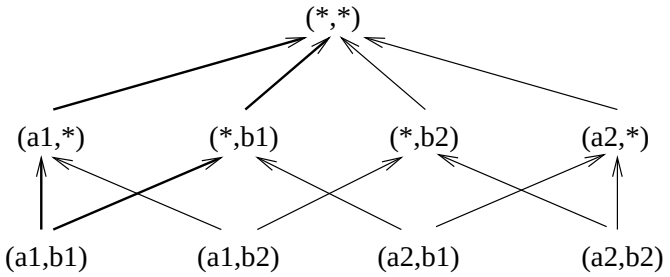


FIG. 1.7: Structure de $\mathcal{L}_h$ dans le cas d'un langage attribut-valeur simple.

dépend de l'expressivité de $\mathcal{L}_h$). Ce sujet, auquel le chapitre 3 est consacré, ne sera pas plus détaillé ici ; bornons-nous pour l'instant à remarquer les faits suivants :

- du point de vue de son implantation sur une machine, la relation de généralité peut se ramener à la notion de preuve puisque h_1 est plus général que h_2 si et seulement si h_2 est une conséquence logique (via le système de preuve implanté en machine) de h_1 : $h_2 \geq_g h_1 \Leftrightarrow h_2 \vdash h_1$;
- il a été démontré que le test $\vdash$ n'est pas décidable lorsque $\mathcal{L}_h$ est le langage de la logique du premier ordre. Il n'existe donc pas d'algorithme permettant de décider en un temps fini si, pour deux hypothèses quelconques h_1 et h_2 de la logique du premier ordre, $h_2 \geq_g h_1$.

Les deux sous-sections ci-après sont consacrées respectivement aux techniques dites ascendantes et descendantes , selon la direction choisie pour explorer $\mathcal{L}_h$. L'usage consistant à placer les exemples en bas du graphe de généralisation, les méthodes ascendantes correspondent à des stratégies de recherche développant une ou plusieurs hypothèses (initialement sur-spécifiques) par généralisations successives, alors que les méthodes descendantes spécialisent des hypothèses initialement sur-générales. Chaque direction de recherche est illustrée par une méthode faisant référence : l'apprentissage basé sur la méthodologie de l'étoile dans le cas de l'apprentissage par généralisation et l'approche diviser pour régner dans le cas de l'apprentissage par spécialisation.

1.3.4 Apprendre c'est oublier

La méthodologie de l'étoile

Une première technique de parcours de l'espace de recherche consiste à oublier les détails non discriminants des exemples afin d'obtenir des règles de classification générales. Plus précisément, les méthodes ascendantes généralisent des modèles sur-spécifiques de manière à augmenter leur taux de couverture tout en relaxant aussi peu que possible la contrainte de cohérence. La méthodologie de l'étoile, détaillée dans [Michalski84], et dont l'algorithme générique figure ci-dessous, est un exemple de méthode ascendante.

`/* Algorithme générique de l'étoile. */`

$\mathcal{H} = \emptyset$

`Tant que` $(\mathcal{B}^+ \neq \emptyset)$ `faire`

- `Sélectionner aléatoirement un exemple positif` Ex `(le noyau) dans` $\mathcal{B}$.
- `Construire l'étoile` $G(Ex|\mathcal{B}^-, m)$, `et choisir la description` D `qui maximise le critère de préférence.`
- $\mathcal{H} \leftarrow \mathcal{H} \cup D$.
- $\mathcal{B} \leftarrow \mathcal{B} \setminus \{Ex \in \mathcal{B}^+ / D \geq_g Ex\}$.
- $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \setminus \{Ex \in \mathcal{B}^+ / D \geq_g Ex\}$.

Cette méthode réalise une couverture itérative des exemples positifs de l'ensemble d'apprentissage en ajoutant séquentiellement, tant que cela est nécessaire, une règle provenant de la généralisation d'un exemple positif non encore couvert (le noyau). $\mathcal{B}^+$ (resp. $\mathcal{B}^-$) dénote l'ensemble des exemples (resp. contre-exemples) de $\mathcal{B}$. $G(E|\mathcal{E}, m)$ dénote l'étoile du noyau E, de borne m, relativement à l'ensemble d'instances $\mathcal{E}$, c'est-à-dire les m meilleures descriptions (suivant un critère de préférence donné) généralisant l'exemple E qui ne couvrent aucun élément de $\mathcal{E}$.

Croissance d'une étoile

La méthode de l'étoile évoque un parcours ascendant du graphe de généralisation, puisque chaque noyau est généralisé en un ensemble de m hypothèses. Un exemple en est donné par le système SIA (décrit chapitre 2 section 2.2.3), dont le processus de généralisation est basé sur un algorithme évolutionnaire. La généralisation du noyau peut également être obtenue à l'aide d'une méthode localement descendante, comme dans le cas de la famille des systèmes AQ [Kaufman et al.99].

Ces systèmes génèrent en effet chaque étoile en développant les spécialisations d'un modèle initialement vide par ajout des caractéristiques du noyau : il s'agit donc d'une spécialisation dirigée par les données (ici, un exemple positif). De plus, contrairement à la plupart des systèmes existants qui implantent un critère de préférence pré-determiné, AQ permet à l'utilisateur de le définir (afin qu'il soit adapté au problème) à l'aide d'un langage codant une séquence (dénotée LEF) de couples *(critère, tolérance)*, comme par exemple :

$$LEF = < (couverture, 10\%), (cohérence, 100\%) >$$

De la couverture itérative en général

La caractéristique permettant de classer la méthodologie de l'étoile parmi les méthodes ascendantes est la généralisation d'un noyau en un ensemble d'hypothèses le couvrant.

Le fait qu'il puisse être nécessaire de répéter plusieurs fois le processus n'a, par contre, aucune implication quand à la direction de la recherche. Un exemple en est donné par le système CN2 [Clark et al.89], qui ne diffère d'AQ – du point de vue de la stratégie de recherche – que par l'absence de noyau : l'algorithme génère l'une après l'autre des règles en ajoutant à chaque étape le test jugé le plus discriminant, et en maintenant une liste des m meilleurs candidats. Ce mécanisme, très voisin de celui mis en œuvre par C4.5 pour construire un arbre de décision (cf. section 1.3.5), classe bien évidemment CN2 parmi les méthodes descendantes.

1.3.5 Apprendre c'est discriminer

Diviser pour régner

L'apprentissage par spécialisation consiste à choisir un modèle initial général, puis à le préciser en discriminant de plus en plus finement entre les différentes classes. La méthode diviser pour régner , qui est utilisée entre autres dans de nombreux algorithmes d'induction d'arbres de décision, est l'approche la plus répandue de ce type de technique ; elle consiste à construire incrémentalement le modèle de décision en partitionnant l'ensemble d'apprentissage en composantes que l'on espère plus homogènes (figure 1.8).

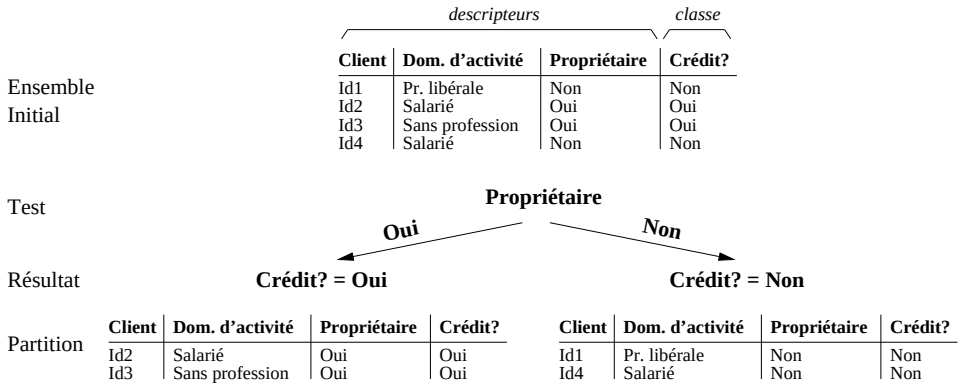


FIG. 1.8: L'approche diviser pour régner.

Ce principe se combine élégamment à la structure récursive des arbres de décision pour former l'algorithme générique de construction ci-dessous.

`/* Algorithme récursif de construction d'un arbre de décision. */`

`Initialement l'arbre est réduit à sa racine` R `qui contient tous les exemples, classes confondues.`

`Tant qu'il reste une feuille` F `dans l'arbre dont les exemples appartiennent à plusieurs classes distinctes`[a] `:`

- `Choisir un test` T `non encore utilisé dans les nœuds-pères de l'arbre.`
- `Ajouter à` F `autant de branches que` T `a de valeurs possibles.` F `devient un nœud de l'arbre comportant le test` T`.`
- `Propager les exemples dans les feuilles correspondantes issues de` T`.`

[a] Dans l'exemple détaillé ci-après, le critère d'arrêt est plus précisément : Tant qu'il existe un test dont le gain d'information est strictement positif .

Un arbre de décision est une structure arborescente permettant de déterminer la valeur de la classe d'un exemple en fonction de tests portant sur la valeur des attributs qui le composent. Par exemple, la structure de la figure 1.8, composée d'un test portant sur la valeur de l'attribut **Propriétaire**, de deux arcs correspondant aux valuations **Oui** et **Non** de cet attribut auxquels sont attachées des feuilles étiquetées respectivement **Crédit ? = Oui** et **Crédit ? = Non**, correspond à un arbre de décision de profondeur 1 pour la classe **Crédit ?**.

La mesure du gain d'information

Le choix du test T à utiliser afin de partitionner à chaque étape de la création de l'arbre l'ensemble d'apprentissage est crucial. Dans le cas des systèmes d'ordre 0, les étapes de spécialisation correspondent à l'ajout de contraintes portant sur les valeurs des attributs. Typiquement, les discriminations sur les attributs symboliques sont faites en développant autant de sous-arbres qu'il existe de valeurs possibles, alors que les discriminations sur les attributs numériques consistent à choisir une valeur réelle utilisée comme seuil (le point de coupure) pour générer une partition binaire. Le critère proposé par Quinlan, qu'il met en œuvre dans le système ID3 [Quinlan86], puis ses successeurs C4.5 [Quinlan93] et C5.0 est appelé gain d'information et est défini par :

$$Gain(E,T) = Entropie(E) - \sum_{i=1}^{|T|} \frac{|E_i|}{|E|}.Entropie(E_i)$$

où E (de cardinal $|E|$) correspond à l'ensemble d'exemples du nœud courant et T est le test utilisé pour partitionner E en $|T|$ sous-ensembles E_i. Cette heuristique, qui évalue la capacité du test à discriminer au mieux les exemples du point de vue de l'entropie des partitions candidates, est compatible avec le principe du rasoir d'Occam puisqu'on peut espérer favoriser la création d'arbres courts. L'entropie d'un ensemble E composé d'exemples de classes $c_i (i \in [1..|C|])$ est une mesure du désordre observé au sein de cet ensemble :

$$Entropie(E) = \sum_{i=1}^{|C|} -p_i.log_2(p_i)$$

où p_i correspond à la proportion des exemples de classe c_i dans E. Dans le cas où $p_i = 0$, on pose : $0.log_2(0) = lim_{x \to 0}\ x.log_2(x) = 0$. Par exemple, dans le cas booléen ($|C| = 2$), l'entropie est minimale (nulle) lorsque l'ensemble considéré est composé d'exemples d'une même classe, et au contraire maximale (égale à l'unité) lorsque les classes sont équi-réparties ($p_1 = p_2 = \frac{1}{2}$).

Post-traitement et extensions

L'obtention d'un ensemble de règles à partir d'un arbre de décision est particulièrement aisée puisqu'il suffit de parcourir l'arbre, chaque chemin menant de la racine à une feuille constituant une règle dont la partie conditionnelle est composée de la conjonction des couples $(Test, Résultat)$ des nœuds et arcs du chemin, la conclusion portant sur la classe

de la feuille [Quinlan87]. Bien qu'a priori négligeable, l'étape de génération des règles à partir de l'arbre de décision est composée en pratique d'un certain nombre d'opérations de simplification supplémentaires (post-élagage, regroupement, etc.) qui constituent en fait la partie la plus coûteuse et complexe de l'algorithme d'induction C4.5. Le lecteur intéressé pourra consulter [Rakotomalala97, Zighed et al.00] qui présentent et comparent de nombreuses approches en se plaçant dans le cadre général des graphes d'induction.

1.3.6 De l'importance de la direction de la recherche

Doit-on préférer les méthodes ascendantes ou descendantes ? Bien que le théorème de conservation porte à croire qu'il n'y a pas de réponse dans l'absolu à cette question, il est tentant d'analyser le choix guidant la recherche conduite par les méthodes présentées afin d'imaginer les cas favorables et défavorables. En ce qui concerne les méthodes basées sur la couverture itérative des exemples (section 1.3.4), et celles qui opèrent par partitionnement des données (section 1.3.5), il est possible de dresser le tableau suivant :

Méthode	Opérateur	Critère d'évaluation
De type AQ	Construction d'une étoile	Fonction de la couverture des données
De type ID3	Ajout d'un test portant sur un attribut	Fonction de la partition des données

Considérons alors les deux problèmes d'apprentissage supervisé **C1** et **C2** détaillés figure 1.9.

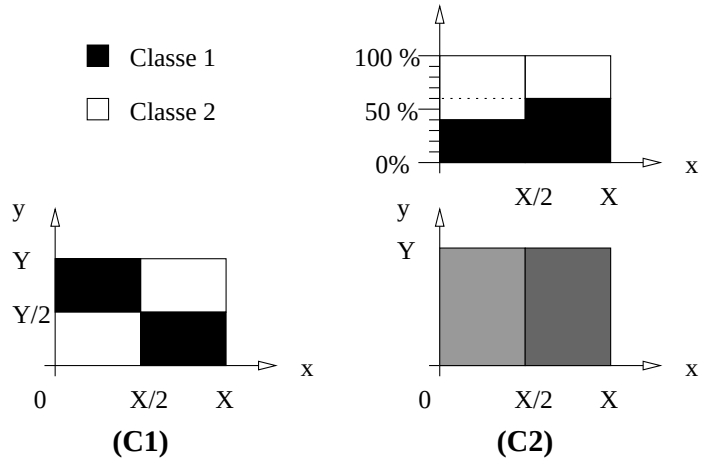


FIG. 1.9: Adaptation de la direction de la recherche aux problèmes d'apprentissage.

Dans les deux cas les exemples peuvent être décrits à l'aide de deux attributs numériques $x \in [0, X]$ et $y \in [0, Y]$; et il est facile, en appliquant les règles détaillées ci-dessous, de générer un nombre quelconque d'exemples distincts formant un ensemble d'apprentissage équilibré.

- Problème **C1**
  $(\mathcal{R}_{C1_1})$: $(x \in [0, X/2]) \wedge (y \in [0, Y/2]) \rightarrow Classe\ 2$
  $(\mathcal{R}_{C1_2})$: $(x \in [X/2, X]) \wedge (y \in [0, Y/2]) \rightarrow Classe\ 1$
  $(\mathcal{R}_{C1_3})$: $(x \in [0, X/2]) \wedge (y \in [Y/2, Y]) \rightarrow Classe\ 1$
  $(\mathcal{R}_{C1_4})$: $(x \in [X/2, X]) \wedge (y \in [Y/2, Y]) \rightarrow Classe\ 2$

- Problème **C2**
  $(\mathcal{R}_{C2_1})$: $(x \in [0, X/2]) \rightarrow Classe\ 1\ \ (pr\ 0,4)\ \ sinon\ Classe\ 2$
  $(\mathcal{R}_{C2_2})$: $(x \in [X/2, X]) \rightarrow Classe\ 1\ \ (pr\ 0,6)\ \ sinon\ Classe\ 2$

Cependant, malgré l'apparente trivialité de ces deux problèmes, les deux types d'algorithmes évoqués ci-dessus auront des comportements très différents.

- Dans le cas du problème **C1**, une recherche basée sur la généralisation sera typiquement amenée à développer un noyau dans chacun des quatre sous-espaces, ce qui conduira aisément aux quatre règles $\{\mathcal{R}_{C1_1}, ..., \mathcal{R}_{C1_4}\}$. Par contre, une recherche basée sur une méthode de spécialisation utilisera un critère (comme l'entropie) destiné à déterminer un point de coupure sur l'un des descripteurs numériques x ou y, de telle sorte que ce dernier sépare au mieux l'espace entre les différentes classes. Or, quelque soit le point de coupure envisagé en x ou en y, la partition de l'espace obtenue conserve une proportion identique d'exemples de classe 0 ou 1 ; une telle méthode est alors incapable de décider comment orienter sa recherche.

- Pour ce qui est du problème **C2**, les rôles sont inversés : une méthode de partitionnement (spécialisation), basant son fonctionnement sur l'analyse de la répartition relative des classes peut déterminer un point de coupure optimum en $x = X/2$. Par contre, dans le cas de la recherche par généralisation, il sera difficile de développer un noyau puisque l'environnement immédiat comporte dans le meilleur des cas 40% de contre-exemples.

1.3.7 Autres méthodes

Bien que la plupart des méthodes d'AS soient monotones (ascendantes ou descendantes), certains travaux se sont attachés à explorer le graphe formé par $(\mathcal{L}_h, \leq_g)$ à la fois à partir des modèles les plus spécifiques, et à partir du modèle le plus général : on parle alors de méthode mixte. À titre d'exemple, cette section présente l'espace des versions, ainsi que la méthode ascendante résultant de sa combinaison avec la méthodologie de l'étoile : l'espace des versions disjonctif.

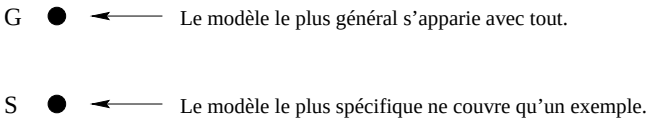
L'espace des versions

L'espace des versions, proposé par T.M. Mitchell [Mitchell77, Mitchell82] est un exemple de méthode d'apprentissage mixte (ascendante et descendante) qui traite incrémentalement un flot d'exemples positifs et négatifs représentés par un nombre fixe d'attributs, sous l'hypothèse que ces derniers ne sont pas bruités, et que la définition recherchée ne s'exprime pas sous forme disjonctive. L'algorithme travaille explicitement sur le treillis associé à

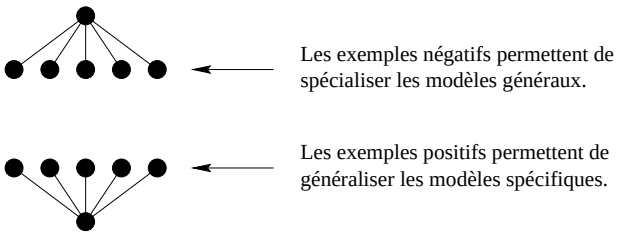
la relation de généralité qui structure l'espace de recherche en développant deux arbres complémentaires, l'un composé des modèles sur-spécifiques, et le second composé des modèles sur généraux. En pratique, ce processus peut se réduire au maintien des ensembles S et G mémorisant respectivement les modèles maximalement spécifiques et les modèles maximalement généraux par rapport aux exemples observés. L'espace des versions fonctionne schématiquement de la manière suivante [Mitchell97, Winston92] :

`/* Algorithme de l'espace des versions */`

1. `Chaque arbre est initialement réduit à un nœud.`

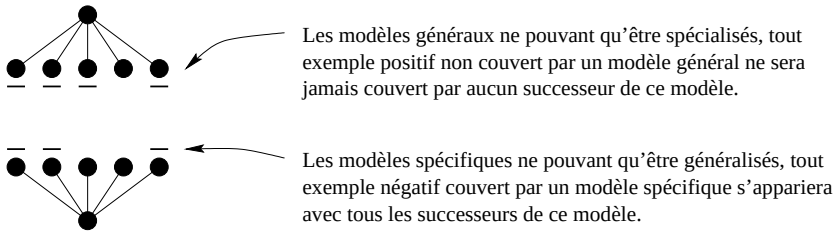


2. `Si l'exemple fourni est positif, généraliser tous les modèles spécifiques afin qu'ils recouvrent ce nouvel exemple. Réciproquement, si l'exemple fourni est négatif, spécialiser tous les modèles généraux afin d'empêcher leur appariement avec cet exemple.`



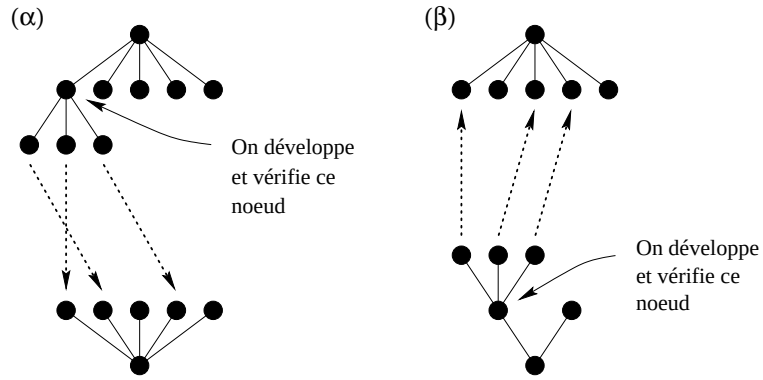
   `Les modifications opérées pour générer les nouveaux modèles spécifiques (resp. généraux) doivent être minimales.`

3. `Les exemples positifs élaguent les modèles généraux. Réciproquement, les exemples négatifs élaguent les modèles spécifiques.`

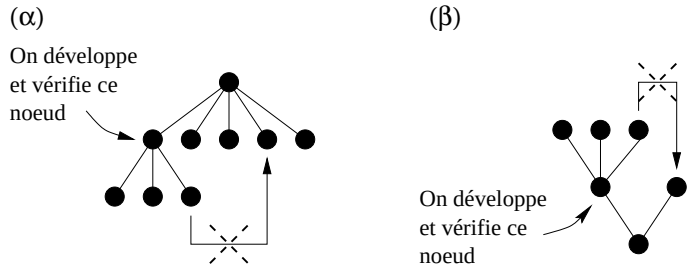


4. `Lors du développement d'un nœud, il faut également vérifier :`

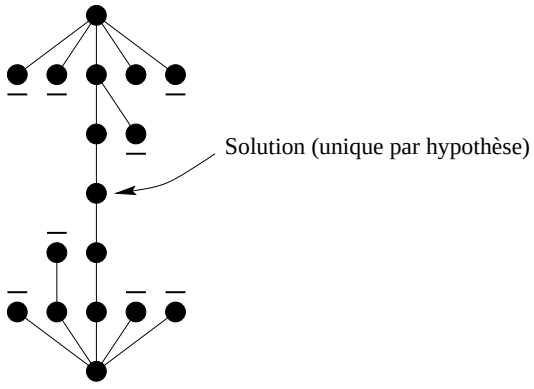
   4.a. `Que toute spécialisation est la généralisation d'un modèle spécifique` (α). `Réciproquement, toute généralisation doit être la spécialisation d'un modèle général` (β).



4.b. Qu'aucune spécialisation n'est la spécialisation d'un autre modèle général (α). Réciproquement, aucune généralisation ne peut être la généralisation d'un autre modèle spécifique (β).



5. L'algorithme termine lorsque les deux arbres ont convergé.



L'algorithme de l'espace des versions est caractérisé par :

- Un aspect dynamique provenant du traitement incrémental du flot des exemples. Il est ainsi possible de classer certaines instances avant que les structures aient convergé vers le modèle solution.

- Un traitement élégant et symétrique des exemples positifs et négatifs.

- Une complexité dans le pire des cas pour la taille de G exponentielle en le nombre d'exemples d'apprentissage et le nombre d'attributs (cf. [Haussler88] p. 185).
- La recherche bidirectionnelle dans le treillis des modèles candidats permet la visualisation du concept de nuance critique (*near-miss*). Un contre-exemple est un *near-miss* s'il est maximalement spécifique (c'est-à-dire situé sur la frontière du treillis séparant G de S).
- Un autre concept fondamental mis en jeu dans cet algorithme est celui de *moindre généralisé*. Le moindre généralisé de deux exemples Ex_1 et Ex_2 est le plus petit élément du treillis de généralisation couvrant ces deux exemples. Dans de nombreux langages, deux exemples quelconques ont plusieurs moindres généralisés.
- Une mise en œuvre difficile en pratique puisque le bruit ou l'absence de modèle non disjonctif dans le langage-cible empêchent la convergence de G et S.

L'espace des versions disjonctif

M. Sebag a étendu la méthode de l'espace des versions au cas de l'apprentissage de concepts disjonctifs à partir de données éventuellement bruitées [Sebag94, Sebag96]. L'approche proposée, appelée espace des versions disjonctif, consiste à construire la théorie $\mathcal{H}$ correspondant à la disjonction des étoiles $G(Ex)$ obtenues pour chaque exemple Ex de la classe-cible par rapport à tous les contre-exemples Ce :

$$\mathcal{H} = \bigvee_{Ex \in \mathcal{B}} G(Ex)$$

$$G(Ex) = \bigwedge_{Ce \in \mathcal{B}} C(Ex, Ce)$$

Le terme $C(Ex, Ce)$, qui permet de discriminer l'exemple Ex du contre-exemple Ce, correspond à la contrainte résultant de la disjonction des tests discriminants les plus généraux relatifs à chaque attribut :

$$C(Ex, Ce) = \bigvee_{a \in \mathcal{A}} T_a(Ex, Ce)$$

Par exemple, les instances :

	Forme	Taille	Couleur	Classe
Ex	Cercle	3	rouge	A
Ce	Triangle	12	rouge	B

sont discriminées par la contrainte :

$$C(Ex, Ce) = [Forme = Cercle] \vee [Taille < 12]$$

dans le cas d'un langage où les sélecteurs pour les attributs nominaux et numériques sont respectivement $=$ et $<,>$. Le sélecteur $\neq$, dont l'utilisation conduirait à discriminer

uniquement sur les valeurs prises par les contre-exemples, est de peu d'intêret. Le coût de construction de $\mathcal{H}$ est de $p.n^2$, p correspondant au nombre d'attributs et n au nombre d'exemples.

L'algorithme de l'espace des versions disjonctif est caractérisé par :

- La possibilité de caractériser des concepts disjonctifs. La combinaison de la méthode de l'étoile et de la construction des ensembles G rappelle une méthode des plus proches voisins.
- La possibilité de traiter des bases bruitées puisque l'utilisateur peut relaxer le modèle $\mathcal{H}$ en réglant un paramètre ϵ qui est un entier positif dénotant le nombre de contre-exemples contraignant l'étoile qu'une instance est autorisée à couvrir.
- Le contrôle du risque de sur-généralisation via un paramètre $M \geq 1$ qui spécifie le nombre minimum de tests qu'une instance doit vérifier pour ne pas couvrir un contre-exemple. Ce paramètre, tout comme ϵ, peut être ajusté *a posteriori* sans nécessiter de nouvel apprentissage.
- Une complexité dans le pire des cas pour la construction de $\mathcal{H}$ (ou la classification de nouvelles instances) polynomiale en le nombre d'exemples d'apprentissage et le nombre d'attributs : la perte de l'incrémentalité est compensée par le fait que l'algorithme se révèle en pratique extrêmement rapide.
- Un modèle $\mathcal{H}$ qui, bien que symbolique, est peu intelligible puisqu'il s'agit d'une disjonction de conjonctions de disjonctions d'un grand nombre de tests.

1.4 Biais et Apprentissage

Les paragraphes précédents ont adopté une approche descriptive destinée à présenter différentes méthodes d'apprentissage. Une autre manière de les différencier consiste à se placer du point de vue du théorème de conservation et à examiner les différents paramètres (implicites ou explicites) orientant les choix faits par l'algorithme : on introduit alors la notion de biais.

1.4.1 Définition et caractérisation des biais

Une définition couramment admise du concept de biais est donnée dans [Utgoff86] :

Définition 1.7 *Biais*
À l'exception des exemples et contre-exemples du concept-cible, tous les facteurs influençant la sélection d'une hypothèse sont des biais.

En effet, la définition antérieure, proposée dans [Mitchell80] (*Tout critère permettant de choisir parmi deux généralisations autre que la cohérence observée sur l'ensemble d'apprentissage est un biais.*) a le défaut de poser que le critère d'évaluation des hypothèses

(ici la cohérence sur la base d'apprentissage) n'est pas un biais. Or ce critère ne peut être défini dans l'absolu pour n'importe quel contexte : en considérant, par exemple, le cas d'une base bruitée, le critère de cohérence aura tendance à guider l'algorithme d'apprentissage vers des hypothèses sur-spécifiques.

Il est possible de classer les biais en trois grandes catégories [Nedellec et al.96] :

Les biais de langage sont constitués par :

- la définition syntaxique du langage $\mathcal{L}_h$ de représentation des hypothèses ;
- toutes les contraintes posées avant l'exploration, comme par exemple la taille maximale des règles ;
- la théorie du domaine, dont la raison d'être est de modifier le comportement de l'algorithme d'apprentissage en fournissant des informations qui amélioreront ses performances.

Les biais de recherche comprennent :

- les critères de préférence, comme l'ordre d'examen des solutions candidates (qui dépend, entre autres, de la relation de généralité considérée) ou encore le critère de validation intermédiaire (qui attribue un score aux définitions et dépend de la fonction de couverture, et donc de la relation de généralité) ;
- les critères restrictifs, qui élaguent dynamiquement au cours de la recherche des portions de l'espace des hypothèses en fonction des résultats antérieurs ;
- la théorie du domaine, qui permet parfois à l'utilisateur de spécifier ses propres critères de préférence qui peuvent être pris en compte au niveau de la génération des hypothèses candidates ou être intégrés dans le critère de validation intermédiaire.

Les biais de validation recouvrent :

- le critère de validation procédant, dans le cas de l'apprentissage de concepts disjonctifs, à l'acceptation d'une règle au sein de l'ensemble solution ;
- le critère d'arrêt de l'algorithme.

1.4.2 Un apprentissage non biaisé

Une première problématique suivant naturellement la présentation de la notion de biais concerne la création et l'étude d'un système d'apprentissage idéal , au sens ou il n'aurait aucun *a priori* sur le monde et ne reposerait par conséquent sur aucun biais. Tom M. Mitchell [Mitchell80, Mitchell97] s'est proposé d'étudier la construction et les propriétés d'un tel système, et a fourni un argument basé sur un raisonnement procédant en deux temps :

1. Une procédure d'apprentissage non biaisée implique tout d'abord l'absence de tout biais de langage : en particulier, $\mathcal{L}_h$ doit être suffisamment expressif pour permettre la caractérisation de n'importe quel sous-ensemble d'exemples $\mathcal{E} \subseteq \mathcal{L}_e$. De plus, puisqu'il n'y a aucun moyen pour choisir entre deux formules de $\mathcal{L}_h$ qui auraient des couvertures identiques sur $\mathcal{L}_e$, nous supposerons que $\mathcal{L}_h$ n'est pas trop expressif, et donc que toute formule de $\mathcal{L}_h$ peut être identifiée de manière unique par le sous-ensemble de $\mathcal{L}_e$ qu'elle couvre, d'où la relation :

$$|\mathcal{L}_h| = 2^{|\mathcal{L}_e|}$$

2. En ce qui concerne l'algorithme de recherche, l'absence de biais implique une méthode similaire à l'espace des versions, c'est-à-dire une méthode maintenant l'ensemble S de tous les modèles maximalement spécifiques et l'ensemble G de tous les modèles maximalement généraux (puisqu'il n'existe aucun moyen de choisir entre deux éléments de S ou deux éléments de G). Il suffit alors de remarquer que la borne S ne contient qu'un unique élément correspondant obligatoirement à la disjonction de tous les exemples d'apprentissage du concept-cible, alors que la borne G correspond à la négation de la disjonction de tous les contre-exemples du concept-cible :

$$S = \bigvee_{Ex \in \mathcal{B}^+} Ex \; ; \quad G = \neg(\bigvee_{Ce \in \mathcal{B}^-} Ce)$$

   Ainsi, un système d'apprentissage non biaisé est incapable de faire le moindre saut inductif, puisque les seules instances qu'il pourra classer sont les exemples et les contre-exemples de la base d'apprentissage[3].

En conséquence, toute tâche d'apprentissage présuppose la mise en œuvre implicite ou explicite de biais. Dans le cas contraire, la méthode obtenue n'est capable que d'apprendre par cœur les exemples d'apprentissage : on obtient un simple algorithme de mémorisation, dont le seul apport – dans le meilleur des cas – est une compression d'information sans généralisation.

1.4.3 Les biais en pratique

Quelques exemples de biais utiles

Puisque les systèmes d'apprentissage non biaisés sont incapables de faire le saut inductif nécessaire au classement d'instances inconnues, les biais sont une composante indispensable de toute tâche d'apprentissage. De plus, la caractérisation des biais mis en œuvre par un algorithme d'apprentissage donné permet de préciser le type de généralisation que ce

[3]Une autre manière de s'en convaincre consiste à considérer la loi de conservation de la précision en généralisation : pour tout exemple E n'appartenant pas à la base d'apprentissage, il existe exactement autant d'hypothèses $\mathcal{H}$ correctes et complètes sur $\mathcal{B}$ contenant E que d'hypothèses correctes et complètes sur $\mathcal{B}$ ne contenant pas E. Il est donc impossible d'attribuer une classe à E en l'absence de biais.

dernier va pouvoir opérer. En pratique, certains biais peuvent être justifiés par le contexte d'utilisation de l'outil d'apprentissage :

Les biais relatifs au domaine d'application concernent toutes les connaissances provenant des parties amont et aval de la chaîne du processus décisionnel. Par exemple, l'estimation du bruit, ainsi que toutes les connaissances relatives au processus de récolte et de représentation des données proviennent de la partie amont de la chaîne du processus décisionnel. À l'oppposé, la prise en compte du coût des décisions est un biais lié à la partie aval de cette même chaîne. Typiquement, si le coût d'une erreur dans le cas d'une instance positive du concept-cible est plus (resp. moins) important que le coût d'une erreur dans le cas d'une instance négative, alors il convient de préférer les définitions générales (resp. spécifiques).

Les connaissances du domaine peuvent jouer un rôle très utile en tant que biais permettant à l'utilisateur de contraindre l'espace de recherche ou de spécifier certaines propriétés ou structures sur les données. Par exemple, dans le cas d'une application dans le domaine de la chimie moléculaire (la base `Mutagenesis` décrite en annexe B, section B.4), il est possible de représenter les molécules comme un assemblage élémentaire d'atomes et de liaisons. Cependant, la spécification de structures (comme certains cycles aromatiques) améliore le plus souvent la précision des règles apprises. En effet, bien qu'aucune nouvelle connaissance n'ait été ajoutée – la structure d'un cycle aromatique peut se retrouver à partir des descriptions des atomes et des liaisons –, la réduction du coût de représentation de ces concepts facilite leur prise en compte lors de l'apprentissage.

Les biais directement liés à l'utilisateur sont centrés autour de l'intelligibilité des hypothèses : puisqu'une des caractéristiques d'un système d'AS est de fournir un résultat compréhensible par un être humain (ce dernier peut alors bénéficier au mieux des connaissances apprises et en contrôler la validité), il est légitime de chercher à maximiser ce critère. Bien que cette notion soit très difficile à spécifier en termes informatiques, et par conséquent encore plus difficile à mesurer, [Kodratoff94] remarque que de très nombreux travaux en AS y font référence (sans pour autant l'expliciter). Un exemple d'étude de cette notion est fourni par [Sommer95]. Ce travail s'appuie sur des constats en linguistique et en psychologie pour proposer de nombreux critères sémantiques et syntaxiques entrant en compte dans la qualité d'une théorie.

Les biais liés au système sont axés autour des contraintes temporelles et matérielles relatives à la tâche à accomplir, qu'il s'agisse de limiter la durée d'un apprentissage ou de restreindre l'espace de recherche à cause de considérations liées à la mémoire disponible ou à la complexité – pour un problème donné – de l'exploration de cet espace.

Une catégorie particulière de biais, relative au domaine d'application du système d'apprentissage mais à notre connaissance jamais mentionnée en dépit de son importance croissante, regroupe toutes les contraintes d'ordre juridique, éthique ou déontologique.

Par exemple, en France, la loi du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés, dont l'une des conséquences est la mise en place de la commission nationale de l'informatique et des libertés (CNIL), prévoit des dispositions visant à imposer une réflexion humaine lorsqu'une décision est prise à la suite d'un traitement informatique et à informer les intéressés des critères retenus dans le cadre d'actes administratifs [Encyclopædia Universalis].

Le principe du rasoir d'Occam

D'autres biais peuvent également relever de principes généraux. Ainsi, le principe de parcimonie, également connu sous le nom de rasoir d'Occam (attribué à William d'Occam, vers 1330 ap. J.-C. Pluritas non est ponenda sine necessitate) est très utilisé dans les sciences expérimentales telles que la physique et, bien sûr, l'AS. Il stipule que les causes ne doivent pas être multipliées si elles ne sont pas utiles , ce qui correspond au biais suivant :

> Lorsqu'il est nécessaire de choisir parmi des hypothèses expliquant des observations, et lorsqu'aucun critère (correction, complétude, ...) ne permet de départager ces hypothèses, alors il est préférable de choisir la plus simple.

[Blumer et al.87] montre que, sous certaines hypothèses très générales, le rasoir d'Occam permet de sélectionner des modèles ayant une probabilité importante de bien se comporter en généralisation. Ce principe se retrouve également dans des résultats récents issus des statistiques théoriques. Par exemple, le principe de contrôle de la capacité mis en œuvre par les machines à vecteurs de support (SVMs, [Vapnik98]) peut être résumé de la manière suivante :

- Informellement, la capacité d'une machine correspond à son aptitude à caractériser un ensemble d'apprentissage quelconque sans erreur. Soient $f_1(\alpha)$, $f_2(\alpha)$, ... des classes de machines de capacité croissante.

- Alors, pour une tâche d'apprentissage donnée, la meilleure performance en généralisation est obtenue au point d'équilibre entre la précision sur l'ensemble d'apprentissage (à maximiser) d'une part, et la capacité de la classe de la machine (à minimiser) d'autre part.

1.5 Apprentissage en logique du premier ordre

La plupart des outils de *data mining* commercialisés à l'heure actuelle qui mettent en œuvre des techniques d'AS travaillent sur des données mises sous forme tabulaire et produisent des règles de type `Si` *`Condition`* `Alors` *`Conclusion`* où *`Condition`* est une conjonction de tests portant sur la valeur des descripteurs et *`Conclusion`* la classe reconnue. On dit alors que $\mathcal{L}_h$ est un langage de la logique propositionnelle (ou logique d'ordre 0). Cette section est consacrée à la programmation logique inductive (PLI, [Muggleton90, Muggleton92, Muggleton et al.94, Lavrac et al.99]) qui regroupe l'ensemble des techniques d'inférence produisant des programmes logiques à partir d'exemples. Une première partie rappelle les spécificités du langage et analyse le gain d'expressivité au regard de l'accroissement de la complexité liée à la manipulation des formules. La seconde partie présente quelques techniques classiques de PLI et souligne les points communs partagés avec les techniques attribut-valeur.

1.5.1 Le langage de la logique relationnelle

Expressivité du langage

Le langage de la logique relationnelle (dont la définition formelle figure en annexe A, section A.2.2), est caractérisé par un vocabulaire bien plus riche que la logique propositionnelle puisqu'il permet :

- de distinguer les objets (les constantes) des propriétés (les prédicats) qui les caractérisent, comme dans l'expression :

$$parent(sébastien, guy)$$

- de définir des relations générales portant sur un nombre quelconque d'objets à l'aide de symboles de variables (associés aux quantificateurs $\forall$ et $\exists$) :

$$\forall X \ \forall Y \ (parent(X,Y) \rightarrow enfant(Y,X))$$

  Cette formule, qui est une clause de Horn, est également notée :

$$enfant(Y,X) \leftarrow parent(X,Y).$$

- de combiner les relations afin de former des concepts de complexité croissante, comme par exemple :

$$grand\_parent(X,Y) \leftarrow parent(X,Z) \wedge parent(Z,Y).$$

  ou encore :

$$\left[\begin{array}{l} ancêtre(X,Y) \leftarrow parent(X,Y). \\ ancêtre(X,Y) \leftarrow parent(X,Z) \wedge ancêtre(Z,Y). \end{array} \right.$$

  qui définit un concept (ancêtre) mettant en jeu une chaîne relationnelle dont la taille ne peut être bornée ;

- de représenter, à l'aide des symboles de fonction, des ensembles infinis (dénombrables) de symboles. Il est par exemple possible de définir l'ensemble $\aleph$ des entiers naturels en représentant l'axiomatique de Peano à l'aide de la constante *zéro* et de la fonction successeur, d'arité 1, notée $s()$:

$$\left[\begin{array}{l} entier\_naturel(zéro) \leftarrow . \\ entier\_naturel(s(X)) \leftarrow entier\_naturel(X). \end{array} \right.$$

  D'autres notions élémentaires d'arithmétique sur les entiers, telles que la relation d'ordre $\geq$ et le concept d'addition peuvent être définis au moyen de la logique relationnelle :

$$\left[\begin{array}{l} sup\_eg(X, zéro) \leftarrow . \\ sup\_eg(s(X), s(Y)) \leftarrow sup\_eg(X,Y). \end{array} \right.$$

$$\left[\begin{array}{l} somme(X, zéro, X) \leftarrow . \\ somme(X, s(Y), s(Z)) \leftarrow somme(X,Y,Z). \end{array} \right.$$

Traitement automatique

En contrepartie de sa grande expressivité, la logique relationnelle a le défaut d'être bien plus coûteuse sur le plan calculatoire que la logique attribut-valeur. Par exemple, considérons le problème de la validation logique d'une hypothèse [Muggleton et al.94] :

Étant donnés les ensembles de clauses $\mathcal{B}^+$, $\mathcal{B}^-$, $\mathcal{T}h$ et H correspondant respectivement à la base d'apprentissage (exemples et contre-exemples), à la théorie du domaine et à un modèle, déterminer si :

$$\forall Ex \in \mathcal{B}^+, H \wedge \mathcal{T}h \models Ex$$

$$\forall Ce \in \mathcal{B}^-, H \wedge \mathcal{T}h \not\models Ce$$

Ce problème, qui revient à résoudre plusieurs fois le problème de la satisfiabilité d'un ensemble de clauses, est décidable dans le cas de la logique d'ordre 0 mais indécidable en logique d'ordre 1 [SS88]. En pratique, les systèmes d'apprentissage relationnel résolvent cette difficulté en travaillant sur des sous-ensembles du langage de la logique d'ordre 1. Les restrictions les plus courantes comprennent :

- des hypothèses sous forme de clauses de Horn ;
- la suppression des symboles de fonctions ;
- l'interdiction de générer des définitions récursives.

Malgré ces restrictions, le test d'implication (noté $\vdash$) qui se substitue à l'implication logique indécidable (notée $\models$) est NP-complet. Ce phénomène[4] est illustré intuitivement en considérant l'hypothèse et l'exemple suivants :

$$H : C(I) \leftarrow P(I, X, Y) \wedge P(I, Y, Z) \wedge P(I, Z, T).$$

$$E : C(e1) \leftarrow P(e1, a, b) \wedge P(e1, c, a) \wedge P(e1, d, c).$$

Ces deux clauses ne comportent pas de symbole de fonction et ne mettent en jeu aucun mécanisme d'auto-référence. De plus, seule l'hypothèse comporte des variables. Ces formules sont donc l'expression la plus simple possible de la notion de concept relationnel : des variables et des constantes lient entre eux des prédicats.

Cependant, le test $E \vdash H$, qui revient à déterminer quels symboles de E associer à quelles variables de H, peut nécessiter de multiples essais. Raisonnons sur l'exemple ci-dessus. Puisque les deux têtes de clause doivent s'apparier, il vient immédiatement que la substitution $\theta_1 = \{I/e1\}$ est contenue dans la solution. Cependant, il n'est pas possible de déterminer *a priori* les instanciations des variables X, Y, Z et T. Choisir par exemple $\theta_2 = \{I/e1, X/a, Y/b\}$ conduit à une impasse car l'hypothèse devient alors :

$$H\theta_2 = C(e1) \leftarrow P(e1, a, b) \wedge P(e1, b, Z) \wedge P(e1, Z, T).$$

[4] La définition de $\vdash$ dans le cadre de $\mathcal{SIAO}1$, l'étude de sa complexité et la construction d'une procédure de calcul efficace sont l'objet du chapitre 3.

et il n'existe pas dans E d'atome couvert par $P(e1, b, Z)$. Les variables X, Y, Z et T sont dites existentielles car elles apparaissent uniquement dans le corps de la clause ; elles sont à l'origine de la NP-complétude du test $\vdash$.

La complexité exponentielle du test de couverture (en fonction du nombre de littéraux de l'hypothèse, voir chapitre 3) étant intrinsèque à la nature relationnelle des données, il n'est pas surprenant que tous les systèmes de PLI fassent appel en pratique à différents biais qui sont autant de restrictions destinées à gérer le dilemme coût de l'induction / richesse du langage .

1.5.2 La programmation logique inductive

Cette section présente très succintement quelques systèmes de PLI représentatifs du domaine afin de situer $\mathcal{SIAO}1$ (ainsi que les approches fondées sur les méthodes évolutionnaires détaillées dans le chapitre 2) parmi les autres travaux qui, bien que mettant en œuvre des méthodes différentes, ont pour objectif de résoudre des problèmes similaires. Afin de rester concis, les biais dont le but est de restreindre la complexité et donc l'expressivité du langage (comme par exemple le langage des modes de PROGOL ou l'ij-détermination de GOLEM) ne sont pas détaillés. Le lecteur intéressé consultera les références.

GOLEM [Muggleton et al.90] est un système adoptant une stratégie de recherche ascendante fondée sur une adaptation du moindre généralisé en présence d'une théorie du domaine. Étant donnée une théorie du domaine $\mathcal{T}h$ composée de clauses définies, GOLEM considère l'ensemble M_h (appelé modèle de Herbrand h-facile) de tous les atomes clos déductibles de $\mathcal{T}h$ en au plus h étapes de résolution. Cette reformulation de la théorie a l'avantage de transformer un modèle potentiellement infini en un modèle de taille bornée qui est toujours calculable.

La stratégie de recherche est fondée sur la méthode de couverture itérative. L'hypothèse ajoutée à chaque étape est construite comme moindre généralisée du plus grand nombre possible d'exemples positifs non couverts. Une difficulté relative au calcul du moindre généralisé de deux clauses C_1 et C_2 de la logique d'ordre 1 vient du fait que la taille de ce dernier est bornée par $|C_1|.|C_2|$. La taille du moindre généralisé croît donc exponentiellement ; dans le cas de n exemples, elle peut atteindre $|M_h|^n + 1$. Afin de résoudre cette difficulté, GOLEM restreint $\mathcal{L}_h$ aux clauses ij-déterminées : la taille du plus grand moindre généralisé ij-déterminé est en effet bornée par une fonction polynomiale.

Une autre catégorie de systèmes de PLI ascendants, dont CIGOL [Muggleton et al.88] et ITOU [Rouveirol91] font partie, est fondée sur des opérateurs dits d'inversion de la résolution [Muggleton et al.92]. Plus récemment, le système CILGG proposé par J.-U. Kietz, permet l'induction de concepts relationnels avec une complexité polynomiale. Ce résultat est obtenu à l'aide d'un opérateur de presque moindre généralisé garantissant une borne à la complexité des clauses générées.

FOIL [Quinlan et al.93, Quinlan et al.95] est un système descendant. Les entrées sont constituées des exemples et contre-exemples du prédicat-cible, et d'une théorie du

domaine définie en extension. La description en extension d'une théorie consiste à énumérer pour chaque symbole de prédicat introduit les ensembles de constantes qui vérifient la relation. La théorie du domaine prise en compte par FOIL est, comme dans le cas de GOLEM, de taille finie.

La stratégie de recherche est très proche de celle du système d'ordre 0 CN2 (cf. section 1.3.4). En effet, FOIL apprend les clauses l'une après l'autre, chaque clause étant construite par ajout successif de littéraux. Le critère de choix du littéral à ajouter est le gain d'information, également choisi par Quinlan pour son système ID3 (décrit section 1.3.5). L'opérateur de raffinement est construit de manière à exclure les littéraux qui occasionnent des appels récursifs sans fin ou qui ne satisfont pas des contraintes liées aux types des arguments ou aux liens entre variables.

PROGOL [Muggleton95] est, comme FOIL, un système descendant apprenant les clauses les unes après les autres. Il introduit le langage des modes, qui permet à l'utilisateur de déclarer des biais très précis concernant les formes possibles des littéraux insérés. Pour chaque exemple, PROGOL développe la clause la plus spécifique respectant les contraintes des modes, notée $\perp_i$. Une recherche de type A^* parmi les clauses couvrant $\perp_i$ est alors réalisée, ce qui donne à cet algorithme une certaine ressemblance avec AQ (cf. section 1.3.4).

TILDE [Blockeel et al.98, Blockeel et al.99] est un système construit sur le modèle de C4.5 qui génère des arbres de décision logiques dont les nœuds sont des conjonctions de littéraux et dont les feuilles sont étiquetées par une classe. L'expressivité du langage de représentation correspond à un programme PROLOG (pouvant contenir des *cuts*) dont les têtes de clauses sont des prédicats d'arité 0. Ce contexte, connu sous le nom d' apprentissage à partir d'interprétations et fondé sur l'hypothèse de localité de l'information, permet l'induction efficace de concepts relationnels non récursifs.

REMO [Zucker et al.96] et son successeur REPART [Zucker et al.98] ont introduit une technique, appelée propositionalisation, qui consiste à reformuler des descriptions en ordre 1 en une représentation en ordre 0. La composante relationnelle de la logique d'ordre 1 provoque un éclatement de chaque exemple sur plusieurs lignes (ie. exemples) de la représentation d'ordre 0 puisqu'à chaque ligne correspond un appariement possible de la structure relationnelle. Ainsi, la projection sur une représentation en ordre 0 d'une représentation en ordre 1 spécifie un problème d'apprentissage particulier, qualifié de multi-instances : la solution recherchée doit couvrir au moins une ligne parmi celles associées à chaque exemple, et aucune des lignes associées aux contre-exemples. [Alphonse et al.99] a résolu l'inconvénient majeur de ces méthodes, lié à la taille de la reformulation en ordre 0. En effet, le nombre d'appariements étant exponentiel en fonction du nombre de littéraux des hypothèses, la taille du problème reformulé est le principal facteur limitant. La solution proposée, appelée propositionalisation sélective, consiste à alterner les phases de reformulation avec les étapes d'apprentissage. Par exemple, dans le cas des méthodes descendantes, seules les lignes correspondant aux hypothèses maximalement générales discriminant les contre-exemples doivent être mémorisées. L'utilisation incrémentale des

contre-exemples permet donc de maintenir une structure bien plus petite, ce qui se traduit expérimentalement par un élagage considérable de l'espace des appariements candidats et un gain en temps également très important.

STILL [Sebag et al.97a] est fondé sur la notion d'appariement stochastique afin de permettre l'induction d'hypothèses relationnelles de taille quelconque avec un coût polynomial. L'appariement stochastique consiste, après une propositionalisation fondée sur un patron, à échantillonner l'espace des appariements en fonction des ressources disponibles plutôt que de procéder à une recherche exhaustive (dont le coût est exponentiel en fonction de la taille des hypothèses). Le désavantage de la méthode vient du fait que le test d'appariement est incorrect et incomplet puisqu'un échec du test peut venir soit du fait qu'aucun appariement n'est possible entre l'hypothèse et l'exemple, soit du fait que l'échantillonnage n'a pas permis de trouver l'appariement solution. L'utilisateur peut contrôler le dilemme coût de l'induction / qualité des hypothèses en paramétrant le nombre d'échantillons. Plus l'échantillon est important, plus l'incertitude du test diminue.

1.6 Récapitulatif

Ce chapitre a permi d'effectuer un tour d'horizon de l'apprentissage automatique symbolique supervisé (AS), qui est le cadre dans lequel se situe notre travail. De nombreux arguments ont été développés :

- L'AS est une discipline récente, motivée par de nombreux besoins, par exemple en ingéniérie des connaissances (IC) ou encore en extraction des connaissances à partir des données (ECD).
- L'AS peut être formulé comme un problème d'optimisation particulier, portant sur des définitions. L'espace de recherche est structuré par une relation d'ordre partiel, appelée relation de généralité. Les stratégies de recherche les plus courantes exploitent cette relation.
- Une autre particularité de l'AS est l'application du principe d'induction, qui consiste à généraliser des observations à des cas inconnus. Une condition indispensable à la généralisation est la mise en œuvre de biais permettant de choisir une définition parmi plusieurs hypothèses candidates de qualité équivalente sur la base d'apprentissage.
- Certains domaines d'application mettent en jeu des données fortement structurées. La programmation logique inductive (PLI) étudie l'induction de concepts exprimés dans le langage de la logique du premier ordre. Si l'expressivité du langage est un atout de ces méthodes, l'accroissement de la complexité nécessite la mise en œuvre de biais parfois très restrictifs.

Chapitre 2

Algorithmes d'Évolution pour l'AS

Ce chapitre a pour ambition de présenter brièvement les algorithmes d'évolution ainsi que les principaux travaux concernant leur application dans le cadre de l'apprentissage automatique symbolique supervisé. En conséquence, une première partie est consacrée à la description des grandes classes de méthodes évolutionnaires que sont les algorithmes génétiques, les stratégies d'évolution, la programmation évolutionnaire et la programmation génétique. Toutes ces classes sont présentées comme des variations d'un même algorithme générique de simulation de l'évolution de populations d'individus. La seconde partie, dédiée à l'application de ces algorithmes aux problèmes d'apprentissage automatique symbolique supervisé, décrit les principales approches qui proviennent des algorithmes génétiques ainsi que de la programmation génétique. Elle est complétée par une présentation plus approfondie de travaux spécifiques, particulièrement importants ou représentatifs, auxquels il est fait référence au cours de cette thèse.

2.1 Les algorithmes d'évolution

2.1.1 Introduction

Naissance du domaine

Dans les années 60, plusieurs chercheurs travaillant indépendamment en Allemagne, sur la côte Est et sur la côte Ouest des US ont proposé et étudié des méthodes d'optimisation basées sur les mécanismes de l'évolution biologique darwinienne. Des trois écoles originelles ont résulté les trois familles d'algorithmes [Spears et al.93, Schoenauer et al.97] :

1. Les algorithmes génétiques développés par J.H. Holland à Ann Arbor dans le Michigan sont issus de travaux consacrés à l'étude de l'évolution du patrimoine génétique d'une population. Le modèle canonique étudié se fonde sur une représentation des individus par des chromosomes codés sous la forme de chaînes binaires de taille fixe.
2. Les stratégies d'évolution, développées par I. Rechenberg et H.-P. Schwefel à la TUB (Technische Universität Berlin), proviennent de travaux d'optimisation appliqués dans divers domaines d'ingéniérie. Contrairement aux algorithmes génétiques, les

stratégies d'évolution sont principalement utilisées dans le but d'optimiser des paramètres réels.

3. La programmation évolutionnaire, développée par L.J. Fogel à San Diego en Californie est issue de travaux liés à l'intelligence artificielle : cette méthode opérait en effet initialement sur des populations d'automates d'états finis, dont l'adaptation était mesurée par leur capacité à prédire leur environnement.

Une description des aspects biologiques de l'évolution et de la reproduction des êtres vivants qui ont conduit au Darwinisme moléculaire inspirant ces travaux peut être trouvée dans [Back96].

Un algorithme d'évolution générique

Ces méthodes ont de nombreux points en commun. Elles opèrent toutes sur des ensembles d'objets (des vecteurs binaires ou réels, des automates, etc.) qui seront appelés par analogie avec la source d'inspiration biologique population d'individus . Chaque individu, modélisé par son patrimoine génétique, peut être considéré d'au moins deux points de vue :

- le point de vue génotypique, qui décrit l'individu d'après sa représentation interne (son code génétique brut) ;
- le point de vue phénotypique, qui correspond à l'expression du code génétique dans l'environnement biologique (l'organisme obtenu).

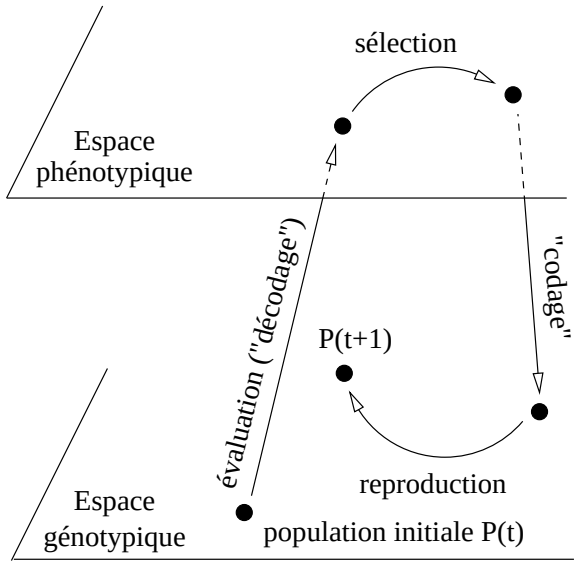


FIG. 2.1: Interaction génotype/phénotype au cours du processus d'évolution.

La figure 2.1, qui reprend un schéma proposé initialement par R.C. Lewontin [Atmar92], montre que l'évolution, tout comme les méthodes qui s'en sont inspirées, est un processus qui se déroule en alternance sur les plans génotypique et phénotypique. Contraire-

ment à des théories de l'évolution comme le Lamarckisme, le Darwinisme implique que le phénotype ne modifie pas le génotype (c'est-à-dire que l'être vivant ne peut modifier son code génétique au cours de son existence) : le flux de l'information génétique ne comporte aucun retour-arrière et seuls le principe de survie des plus adaptés et les modifications moléculaires opérant sur les chromosomes lors de la reproduction des parents déterminent (en première approximation) l'évolution du patrimoine génétique de la population.

Le squelette commun à tous les algorithmes d'évolution est composé d'une boucle générique qui correspond au déroulement des générations biologiques (cf. figure 2.2).

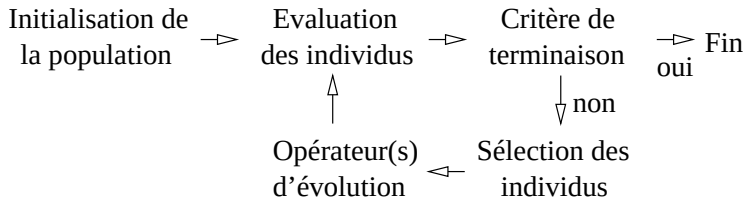


FIG. 2.2: Algorithme d'évolution générique.

Bien que le détail du fonctionnement de l'évolution varie d'une méthode à l'autre, les principaux opérateurs appliqués dans la boucle générique (qui sont la plupart du temps probabilistes) peuvent être classés en trois catégories :

- L'opérateur d'évaluation (appelé également fonction de qualité), qui attribue un score aux individus. Ce score mesure leur adaptation à l'environnement. Dans le cas général de l'optimisation (maximisation) d'une fonction f, le score de l'individu I est simplement $f(I)$.
- L'opérateur de sélection, qui applique le principe darwiniste selon lequel plus un individu est adapté à son environnement, plus sa probabilité de survivre et de se reproduire est élevée.
- Les opérateurs de reproduction, qui travaillent sur le code génétique. En modifiant les chromosomes, ils permettent la génération de nouveaux individus. De nombreux algorithmes d'évolution se placent dans le cadre des algorithmes génétiques en utilisant un opérateur de mutation et un opérateur de recombinaison. Cependant, le nombre et la fonction des opérateurs peuvent varier suivant les méthodes et les objectifs poursuivis.

Deux autres opérations sont nécessaires à l'implantation du processus de simulation :

- l'initialisation de la population au tout début du processus ;
- le test de terminaison qui a lieu à chaque itération de la boucle d'évolution.

Algorithmes d'évolution et optimisation

Il existe une très forte similitude entre le processus d'évolution tel qu'il a été décrit de manière générique ci-dessus et un processus d'optimisation. En effet, la pression exercée par l'opérateur de sélection à travers l'opérateur d'évaluation peut être considérée comme un procédé de maximisation de la fonction de qualité ; l'algorithme évolutionnaire correspond alors à un processus d'optimisation de cette fonction. Cependant, les algorithmes évolutionnaires se différencient des méthodes d'optimisation classiques puisque :

1. la gestion d'une population d'individus permet l'exploration simultanée de plusieurs points de l'espace de recherche ;
2. la fonction de qualité est utilisée uniquement lors de l'étape d'évaluation des individus afin d'obtenir leur score.

Actuellement, la très grande majorité des travaux concernant les algorithmes d'évolution traite de leur étude ou de leur application en tant que procédures d'optimisation. En pratique, ces derniers sont particulièrement recommandés lorsque le problème à résoudre présente une ou plusieurs des caractéristiques suivantes :

1. la fonction à optimiser ne dispose pas des qualités de régularité (telles que la continuité, la dérivabilité, ...) requises par les approches mathématiques classiques ;
2. le problème est bruité, dynamique, multimodal (cf. les techniques de niches écologiques ou les fonctions de partage de la qualité (*sharing*) [Goldberg et al.87]), multi-critères [Zitzler99] ou encore défini sur des espaces de dimension élevée ;
3. la fonction à optimiser présente certaines qualités mathématiques, cependant les heuristiques de recherche classiques ne donnent pas de résultats satisfaisants. Par exemple, bien que les problèmes de type *longpath* [Horn et al.94] puissent être résolus par un algorithme de montée (*hill climbing*), le fait que le nombre de pas nécessaire soit exponentiel en fonction de la dimension de l'espace de recherche rend ce dernier inutilisable en pratique.

Comme le remarque [Back96], même dans les cas où le problème d'optimisation est globalement intraitable, la découverte par les algorithmes d'évolution de solutions de meilleure qualité que celles obtenues par les méthodes traditionnelles dans de nombreuses applications pratiques est également une raison majeure de leur succès.

Les restrictions concernant l'utilisation des algorithmes d'évolution sont les suivantes :

– Du fait de leur nature stochastique d'une part et de l'utilisation d'une population d'autre part, ces méthodes nécessitent une puissance de calcul importante. En particulier, il est préférable que la fonction de qualité, qui est utilisée un très grand nombre de fois, soit rapidement calculable[1].

[1] Cette condition n'est pas impérative : par exemple [Periaux et al.95] applique des techniques évolutionnaires à des problèmes d'optimisation de formes en aérodynamique qui nécessitent pour chaque structure des calculs extrêmement complexes (génération d'un réseau maillé modélisant la structure à partir de la représentation initiale sous forme de points de contrôle de courbes splines puis évaluation de la structure ainsi discrétisée par résolution des équations d'Euler associées).

– Bien que certaines classes d'algorithmes évolutionnaires comme la programmation génétique soient très souples quand à la structure syntaxique de l'espace de recherche, la plupart des méthodes travaillent sur des vecteurs de booléens ou de réels. Cette contrainte, commune à la totalité des méthodes d'optimisation classiques, est un inconvénient lorsque le domaine d'application nécessite la manipulation de structures complexes, comme par exemple en Apprentissage Symbolique.

De plus ces techniques ont, comme beaucoup d'autres, l'inconvénient de ne pouvoir garantir en pratique la découverte de l'optimum (ou des optima), ni de pouvoir les reconnaître dans le cas où ces derniers auraient été découverts. En effet les résultats théoriques, lorsqu'ils existent (convergence à l'infini, taille adéquate de la population) ont souvent un coût trop important pour pouvoir être utilisés dans des conditions réelles.

Les sous-sections qui suivent sont consacrées à la présentation des principales méthodes évolutionnaires.

2.1.2 Les algorithmes génétiques

Historique

Les algorithmes génétiques sont probablement les plus connus des algorithmes d'évolution. Leur forme initiale (dite canonique) est dûe aux travaux de Holland, résumés en 1975 dans le livre *Adaptation in natural and artificial systems* [Holland75], qui a servi de référence à l'ensemble des travaux du domaine. Bien qu'un des buts initiaux de son travail ait été la simulation et l'étude des systèmes biologiques, Holland a poursuivi un objectif bien plus général en développant une théorie des systèmes adaptatifs, capables d'apprentissage, interagissant avec leur environnement à l'aide de détecteurs et d'effecteurs (la section 2.2.1 est consacrée aux systèmes de classeurs issus de ces travaux). De plus, Holland s'est également intéressé à l'application des algorithmes génétiques à la résolution de problèmes réels.

Description de l'algorithme génétique canonique

L'algorithme génétique canonique (AGC) opère sur une population P de n individus. Chaque individu I^k $(1 \leq k \leq n)$ est composé d'un chromosome qui est une chaîne binaire de longueur fixe l : $I^k = \{i_1^k, ..., i_l^k\}$. La population initiale P_0 est générée aléatoirement. Le déroulement d'une génération, qui permet d'obtenir la population P_{g+1} à partir de la population P_g est le suivant :

Évaluation et sélection
Soit $f : I \longrightarrow \Re^{+*}$ la fonction d'évaluation et P_g la population courante ; le premier opérateur appliqué est l'opérateur de sélection. Ce dernier génère une population P'_g en choisissant aléatoirement n individus de P_g, où la probabilité de sélectionner un

individu I^k de P_g est :

$$p_s(I^k) = \frac{f(I^k)}{\sum_{j=1}^{n} f(I^j)}$$

Cet opérateur, non déterministe et conforme au principe darwiniste de la sélection naturelle des individus les plus adaptés est appelé sélection proportionnelle à la qualité .

Reproduction

Par analogie avec le monde biologique, les individus sélectionnés sont appelés parents. Les opérateurs modélisant la reproduction sexuée (la mutation, notée $\mu : I \to I$ et le croisement, noté $\times : I^2 \to I^2$, voir figure 2.3) permettent d'obtenir le matériel génétique des individus de la population suivante (les enfants). Le croisement à un

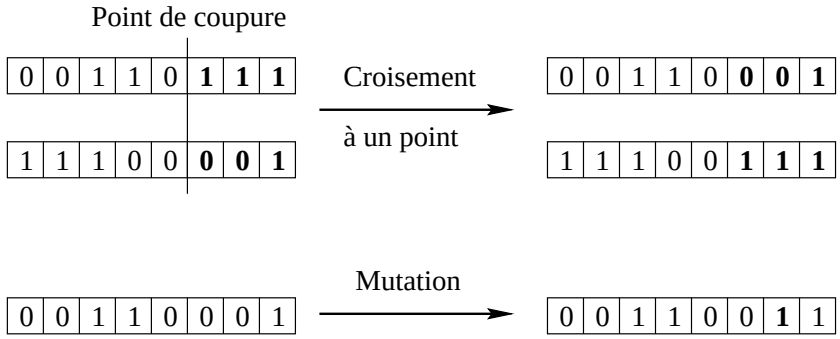


FIG. 2.3: Opérateurs de reproduction de l'AGC.

point tire un point de coupure uniformément dans $[1, l-1]$ puis échange les sous-chaînes correspondantes. Il est appliqué avec une probabilité p_c à l'ensemble des individus de P'_g qui ont été préalablement groupés par paires. La population ainsi obtenue est notée P''_g. La mutation consiste à changer chacun des bits des individus de P''_g avec une probabilité p_m (supposée peu importante), ce qui fournit P_{g+1}.

La figure 2.4 détaille l'évolution sur une génération d'une population de $n = 4$ individus comportant $l = 5$ bits. La fonction de qualité utilisée ici est $f(I^k) = (\sum_{j=1}^{l} i_j^k.2^{l-j})^2$. En pratique, les populations contiennent plusieurs dizaines à plusieurs milliers d'individus.

Une explication intuitive du fonctionnement général de l'AGC a été proposée par Holland et Goldberg [Goldberg89]. La sélection duplique les meilleurs individus avec d'autant plus de force que la pression sélective, définie par le rapport $\frac{f_{max}}{\overline{f}}$, où f_{max} dénote la qualité du meilleur individu de P et $\overline{f}$ la qualité moyenne, est importante. Le croisement procède à un échange d'informations entre deux chromosomes et permet, avec la sélection, la propagation des blocs de bits garantissant une qualité élevée des individus. Il est donc typiquement appliqué avec une probabilité assez importante ($p_c = \frac{1}{2}$ par exemple). Par contre la mutation, qui ne sert qu'à maintenir une certaine diversité génétique et garantir la possibilité d'explorer tout point de l'espace de recherche, est appliquée avec une probabilité

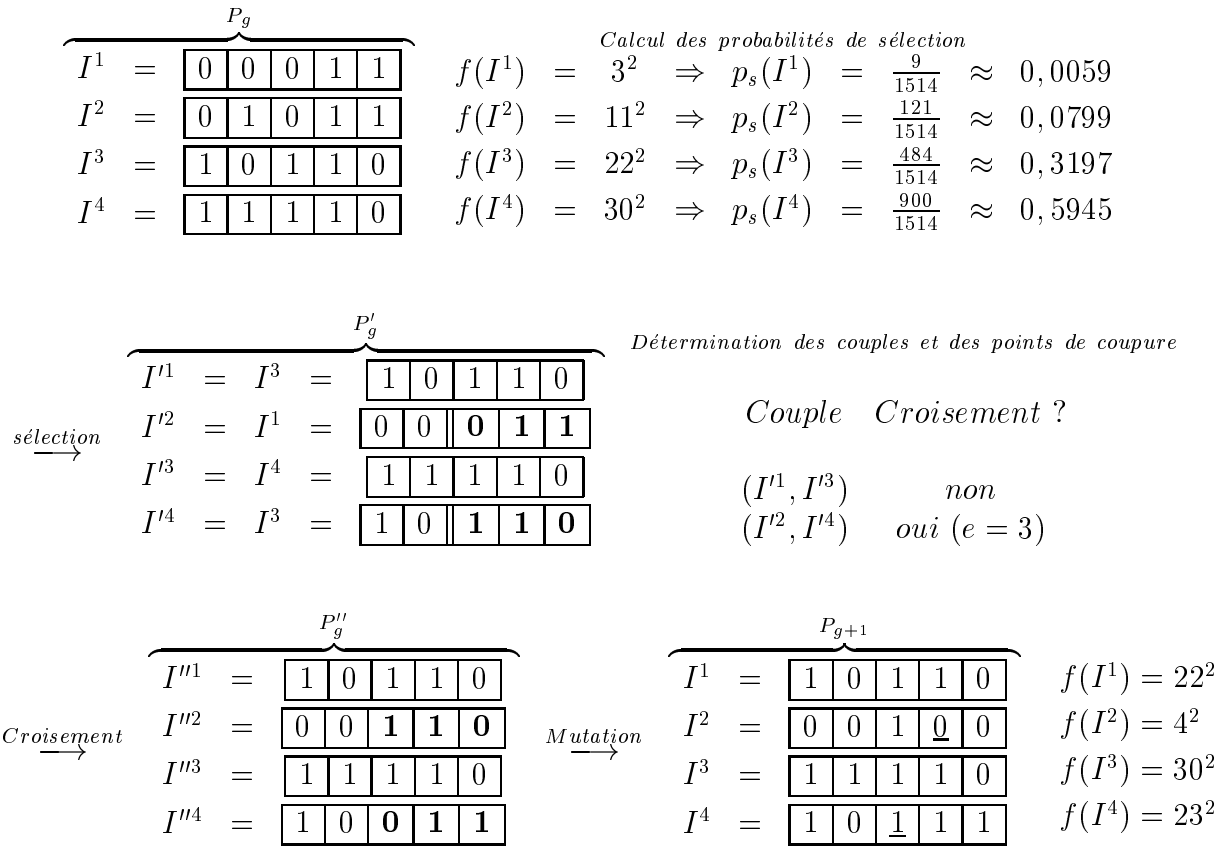


FIG. 2.4: Déroulement d'un cycle de l'algorithme génétique canonique.

très faible ($p_m = \frac{1}{1000}$ par exemple) afin d'éviter que l'algorithme ne puisse plus conserver et exploiter les individus prometteurs et finisse par se comporter comme une marche aléatoire.

Cette explication, qui valorise l'effet de l'opérateur de croisement (principal apport initial de Holland par rapport aux autres méthodes évolutionnaires), est l'objet de discussions et a été souvent remise en question. Par exemple [Jones95] critique la méthode de validation consistant à comparer un AGC classique avec un AGC sans croisement. Une nouvelle méthode, basée sur le remplacement du croisement classique par un croisement aléatoire, montre que dans de nombreux cas le croisement classique, apparemment utile, peut être remplacé avantageusement par un opérateur de macromutation. Cet argument remet en cause l'analyse justifiant l'intérêt du croisement par leur capacité à permettre un échange d'informations entre individus de la population. D'autre part, elle légitimise les méthodes évolutionnaires telles que les stratégies d'évolution, décrites section 2.1.3.

Analyse de l'algorithme génétique canonique

L'analyse faite par Holland de l'AGC à l'aide des schémas est le premier résultat théorique faisant référence dans le domaine et ayant donné lieu à de très nombreuses investigations.

Un schéma est un vecteur (noté H) de $\{0,1,*\}^l$, que l'on utilise afin de caractériser la similarité d'ensembles d'individus ; le symbole $*$ correspondant indifféremment à 0 ou à 1. Par exemple le schéma $H = \underline{1} * * * \underline{0} *$, qui comporte deux bits définis (soulignés), couvre entre autres les individus $\underline{1}0100\underline{0}$ et $\underline{1}1100\underline{1}$. Il est aisé de démontrer d'une part qu'il existe 3^l schémas distincts définis sur l'espace des chaînes binaires de taille l, et d'autre part qu'un individu donné appartient à $2^l - 1$ schémas.

Définition 2.1 *Ordre d'un schéma*
L'ordre d'un schéma H, noté $o(H)$, est le nombre de bits définis que comporte ce schéma.

Définition 2.2 *Longueur d'un schéma*
La longueur d'un schéma H, notée $\delta(H)$, correspond à la distance entre la première et la dernière position définie de ce schéma.

Par exemple, avec $H = 1 * * * 0 *$, on a : $o(H) = 2$ et $\delta(H) = 4$.

Soit $m(H,g)$ le nombre d'exemplaires du schéma H présents à la génération g dans la population et $f(H,g)$ (resp. $\overline{f}(g)$) la moyenne des qualités des individus couverts par H (resp. la moyenne des qualités des individus de la population) à la génération g. L'espérence du nombre d'exemplaires du schéma H présents à la génération $g+1$ dans la population est donc : $E(m(H,g+1)) = m(H,g).\frac{f(H,g)}{\overline{f}(g)}$ si l'on ne considère que l'action de l'opérateur de sélection, qui multiplie les instances des schémas proportionnellement à leur force relative. Comme d'autre part la probabilité de survie d'un schéma H à un croisement à un point appliqué avec la probabilité p_c est $p_{s\times 1pt} = 1 - p_c.\frac{\delta(H)}{l-1}$ et que la probabilité de survie de H à la mutation modifiant chaque bit avec la probabilité p_m est de $p_{s\mu} = (1-p_m)^{o(H)} \approx 1 - o(H).p_m$ (pour $p_m \ll 1$) ; il est possible de minorer $E(m(H,g+1))$ en ne comptabilisant que les schémas dupliqués sous réserve qu'ils n'ont pas été détruits par l'application des opérateurs génétiques et en ignorant ceux qui sont éventuellement créés lors de la génération de nouveaux individus. En approximant $p_{s\times 1pt}.p_{s\mu}$ qui est de la forme $(1-x).(1-y)$ par $(1-x-y)$, on obtient le résultat du théorème 2.1.

Théorème 2.1 *(théorème des schémas)*
Soit H un schéma présent en $m(H,g)$ exemplaires dans P_g. L'espérance du nombre de représentants de H dans la population à la génération suivante de l'AGC vérifie :

$$E(m(H,g+1)) \geq m(H,g).\frac{f(H,g)}{\overline{f}(g)}.(1 - p_c.\frac{\delta(H)}{l-1} - o(H).p_m)$$

Le théorème des schémas a suscité de nombreux travaux :

L'hypothèse des blocs de construction
définit ces derniers comme les schémas de qualité supérieure à la moyenne ($\frac{f(H,g)}{\overline{f}(g)} > 1$), courts ($\delta(H)$ petit) et comprenant peu de bits définis ($o(H)$ petit). Puisque le théorème des schémas prévoit une croissance exponentielle des blocs de construction au sein de la population, cette hypothèse postule (avec de nombreuses confirmations empiriques) qu'une performance quasi-optimale peut être effectivement obtenue par combinaison de schémas très performants, courts, et d'ordre faible.

Le parallélisme implicite
fait référence à un résultat démontré par Holland estimant (sous certaines hypothèses relatives, entre autres, à la taille de la population) que le nombre de schémas pris en compte utilement à chaque génération par un algorithme génétique travaillant sur une population de n individus est de l'ordre de $\Theta(n^3)$. Ce résultat conduit Holland à conseiller l'utilisation d'un alphabet minimal (c'est-à-dire binaire), argumentant que le nombre de schémas diminue lorsque la taille de l'alphabet définissant les valeurs potentielles des gènes augmente. Cette argumentation est toutefois fortement combattue : [Antonisse89] précise la notion de schéma sur les alphabets non binaires et montre que leur nombre ne diminue pas avec la taille de l'alphabet.

La caractérisation des problèmes trompeurs
(du point de vue du théorème des schémas) découle de la notion de compétition entre schémas. Informellement, une fonction est dite trompeuse lorsque les blocs de construction courts et d'ordre faible se combinent en des blocs plus longs et d'ordre élevé sub-optimaux [Goldberg87].

Enfin, la stratégie d'allocation exponentielle des ressources aux meilleurs schémas est justifiée par Holland sur l'étude du problème du bandit manchot à deux (puis k) bras puisqu'il démontre que la stratégie d'allocation optimale consiste à donner un nombre d'essais légèrement plus qu'exponentiel au meilleur bras observé. De ce point de vue, l'AGC résoud le dilemme exploration versus exploitation puisqu'il maintient un équilibre quasi-optimal du point de vue de la maximisation des qualités cumulées des individus générés entre l'effort consenti à l'exploration de nouvelles régions de l'espace de recherche et l'exploitation des régions connues les plus prometteuses[2]. Cet argument est repris dans [De jong92] pour souligner la différence entre l'objectif d'une procédure d'optimisation classique et ce qu'optimise réellement l'AGC.

[2]Contrairement à ce que l'on pourrait croire, le résultat concernant l'optimalité de la stratégie de l'AGC pour la maximisation des qualités cumulées des individus (quelle que soit la fonction de qualité considérée) n'est pas en opposition avec le théorème de conservation (cf. section 1.2.3). En effet, le maximum de la qualité cumulée des individus ne peut s'exprimer comme une fonction Φ de l'ensemble ordonné d_m des m solutions distinctes examinées par l'AGC puisque d_m ne permet pas de comptabiliser les évaluations multiples.

Quelques extensions

Compte tenu du nombre et de la variété des travaux concernant les algorithmes génétiques, toute tentative de résumer ces derniers dans un espace raisonnable est vouée à l'échec. Les quelques lignes suivantes son destinées à donner un avant-goût de certains travaux afin de permettre un premier élargissement du champ de vision du lecteur intéressé.

Premières variations sur l'algorithme génétique canonique

De très nombreuses déclinaisons des opérateurs génétiques ont été proposées.

La classique sélection proportionnelle à la qualité mise en œuvre dans l'AGC peut être complétée par des techniques de changement d'échelle, c'est-à-dire l'application de transformations du type $f(I) \rightarrow \alpha.f(I)+\beta$ afin de garantir une pression sélective donnée. Les autres opérateurs de sélection sont la sélection par tournoi de k individus, qui consiste à choisir pour chaque ajout le meilleur individu parmi k déterminés aléatoirement dans la population, et la sélection par le rang où la probabilité de sélectionner un individu ne dépend plus de sa qualité mais d'une fonction (typiquement linéaire) de son rang dans la population. Chacun des opérateurs de sélection décrit ci-dessus peut être utilisé de manière incrémentale (*steady state*) ou générationnelle : contrairement à la sélection générationnelle, une sélection incrémentale ne remplace qu'une partie (typiquement un individu) de la population à chaque génération. D'un point de vue plus théorique, [Baker87] analyse les opérateurs les plus connus et propose deux opérateurs de complexité $\Theta(n)$ minimisant les erreurs d'échantillonnage afin d'augmenter l'adéquation entre le modèle du comportement de l'AG dérivé du théorème des schémas et la réalité expérimentale. D'un point de vue pratique, les variations les plus populaires comprennent la sélection élitiste, qui garantit la présence des k meilleurs individus d'une génération sur l'autre (habituellement $k = 1$), et la sélection par élevage (breeding), qui utilise un schéma classique (roulette, tournoi, ...) n'opérant que sur les k meilleurs individus (typiquement 10% de la population).

En ce qui concerne les croisements, l'opérateur utilisant un point de coupure a été généralisé tout d'abord au croisement à n points de coupure (déterminés aléatoirement), puis au croisement uniforme [Syswerda89]. Ce dernier a l'avantage de supprimer toute notion de localité dans les blocs de construction puisqu'il échange les gènes des parents avec une probabilité fixe p_e. La capacité du croisement uniforme de détruire les schémas ne dépend plus alors que de l'ordre des schémas, et non des positions des bits définis.

Quelques autres travaux théoriques

Les vertus explicatives et prédictives du théorème des schémas ont montré de nombreuses limites qui ont motivé d'autres travaux :

- il a pu être constaté que des fonctions comme *royalroad* [Mitchell et al.92], construites pour être facilement optimisables par les algorithmes génétiques sont en réalité plus difficiles que prévu ;
- réciproquement, certaines fonctions trompeuses se sont révélées en pratique faciles à optimiser pour les algorithmes génétiques ;

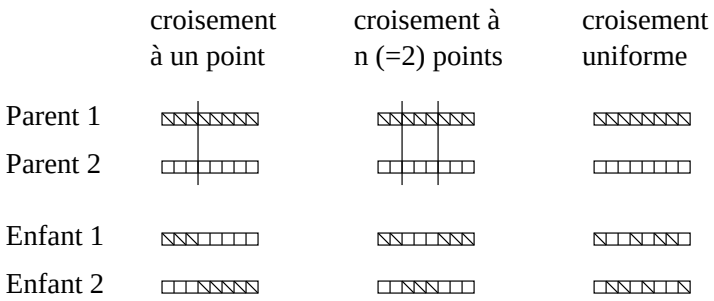


FIG. 2.5: Principaux croisements sur les chromosomes binaires de taille fixe.

- le théorème donne une borne inférieure au nombre de schémas dupliqués en négligeant les effets positifs de la mutation et du croisement (l'effet disruptif est majoré et la possibilité que le schéma soit généré à nouveau ignorée) ;
- puisque la qualité moyenne des individus de la population augmente lorsque l'algorithme génétique progresse, l'hypothèse d'un ratio $\frac{f(H,g)}{\overline{f}(g)}$ constant qui garantirait la duplication exponentielle des bons schémas est peu réaliste ;
- les erreurs d'échantillonnage liées à l'utilisation de populations de petite taille (par rapport à la taille de l'espace de recherche) ainsi que la variance parfois importante de la qualité des individus couvrant un schéma donné peuvent causer des divergences entre les comportements attendu et réel [Schaffer et al.91].

Une première direction de recherche concerne la poursuite de l'analyse basée sur le théorème des schémas : ainsi N.J. Radcliffe en a proposé une version très générale dans le cas d'un alphabet et d'opérateurs quelconques en étendant la notion de schéma à la notion de classe d'équivalence (appelée forma) induite par une relation d'équivalence définie sur les individus. Cette analyse est complétée par l'énoncé de principes caractérisant d'une part ce qu'est une bonne représentation du point de vue des formae, et d'autre part l'adéquation du couple (représentation, opérateurs) à l'algorithme d'évolution [Radcliffe91]. Ces travaux ont également permis en pratique la proposition et l'analyse d'opérateurs travaillant sur des objets tels que les multi-ensembles [Radcliffe92].

Une seconde direction de recherche concerne l'étude de la complexité génétique dérivée de la notion de problème trompeur. Parmi les nombreux résultats, on notera [Wilson91] qui montre qu'il existe des fonctions AG-faciles qui ne sont pas faciles pour les méthodes de gradient, [Venturini95] qui montre que la réciproque est également vraie, et [Whitley91] qui montre que les fonctions AG-difficiles sont également difficiles pour les méthodes de montée (*hill-climbing*).

Une dernière direction s'affranchit du cadre théorique de Holland. En effet, de très nombreuses méthodes ont été appliquées à la modélisation des algorithmes génétiques, qu'il s'agisse de simples droites de régression (cf. [Grefenstette95] où les droites $F_e = \alpha + \beta.F_p$ et $\sigma_e(F_p) = \sigma_\alpha + \sigma_\beta.F_p$ modélisent la qualité des enfants (F_e) à partir de celle des parents (F_p), ce qui permet une simulation rapide du comportement de l'algorithme basée sur une population virtuelle d'individus représentés uniquement par leur qualité), de modèles empruntés à la physique statistique (afin d'étudier sur le plan macroscopique

l'évolution de grandeurs telles que la qualité moyenne des individus), ou encore de modèles exacts comme les chaînes de Markov [Vose92] (il est possible de remarquer dans ce dernier cas que chaque population étant un état du système, le modèle associé à l'évolution d'une population de taille finie comporte $C_{2^l-1}^{n+2^l-1} \approx (\frac{2^l}{n}e)^n$ états, ce qui rend ce type de modélisation particulièrement complexe).

2.1.3 Les stratégies d'évolution

Historique

Les stratégies d'évolution [Rechenberg73] ont été conçues conjointement par Bienert, Rechenberg et Schwefel, ce dernier ayant procédé aux premières implantations destinées aux applications à des problèmes d'optimisation numérique. Contrairement aux algorithmes génétiques, cette classe de méthodes ne prétend pas modéliser un quelconque processus biologique : elle met en œuvre des processus itératifs simples, définis et analysés rigoureusement, et destinés dès leur origine à la résolution de problèmes d'optimisation définis dans $\Re^n$. Les paragraphes suivants décrivent par ordre d'apparition chronologique les trois classes de stratégies qui fondent ce domaine.

La stratégie $(1+1)-ES$

La première stratégie, notée $(1+1)-ES$, travaillait sur un unique vecteur de n réels auquel était appliqué une mutation, c'est-à-dire une perturbation de chaque coefficient d'écart-type fixe et de moyenne nulle. Le résultat obtenu était comparé au parent, et le meilleur des deux individus était conservé. Les premiers travaux théoriques de Rechenberg concernant la convergence de la stratégie $(1+1)-ES$ sur deux types de fonctions lui permirent de proposer la règle dite du seuil des 1/5, qui estime à 1/5 le ratio ρ_s optimal de mutations réussies. En conséquence, lorsque $\rho_s > 1/5$, il faut augmenter l'écart-type σ de l'opérateur de mutation alors que lorsque $\rho_s < 1/5$, il faut diminuer σ. De plus Schwefel obtint, dans le cas où le modèle à optimiser est une sphère, une valeur théorique pour le coefficient de modification de σ : $c = 0,817$[3].

La stratégie $(\mu+1)-ES$

Un second schéma de sélection, introduisant la notion de population et l'opérateur de recombinaison fut ensuite étudié : la stratégie $(\mu+1)-ES,\ \mu > 1$, qui correspond à la sélection incrémentale (*steady-state*) des algorithmes génétiques, consiste à recombiner à chaque étape les μ parents pour obtenir un individu qui, après mutation, remplace le pire

[3]Plus précisément, la règle exacte donnée par Schwefel est la suivante : Toutes les n mutations, compter combien de succès ont été observés au cours des $10.n$ dernières mutations. Si ce nombre est inférieur à $2.n$, alors $\sigma \leftarrow 0,85 \times \sigma$. Si ce nombre est supérieur à $2.n$, alors $\sigma \leftarrow \sigma/0,85$. La constante c a en effet été corrigée de 0,817 à 0,85 afin de s'affranchir de la spécificité du modèle sphérique, qui a tendance à nécessiter un pas d'adaptation assez important.

parent de la population s'il lui est supérieur. La mutation et l'ajustement des écarts-types proviennent de la (1+1)-ES stratégie, alors que la recombinaison (notée rec, qui est vue comme la composition d'un opérateur de sélection des parents noté co et d'un opérateur de combinaison des parents noté re : $rec = re \circ co$) peut appartenir à l'un des quatre types ω suivants :

$\omega = 0$ *Aucune recombinaison*
On a : $co : I^\mu \to I$ (sélection par tirage aléatoire) et $re : I \to I = Id$.

$\omega = 1$ *Recombinaison globale intermédiaire*
On a : $co : I^\mu \to I^\mu = Id$ (aucune sélection) et $re : I^\mu \to I$ où chaque coefficient du vecteur résultat correspond à la valeur moyenne des coefficients des parents.

$\omega = 2$ *Recombinaison locale intermédiaire*
On a : $co : I^\mu \to I^2$ (sélection aléatoire sexuée) et $re : I^2 \to I$ où chaque coefficient c_i du vecteur résultat est obtenu à partir des coefficients c_i^1 et c_i^2 de ses parents par : $c_i = u.c_i^1 + (1-u).c_i^2$ avec $u \in]0,1[$ (en général $u = \frac{1}{2}$).

$\omega = 3$ *Recombinaison discrète*
On a : $co : I^\mu \to I^\mu = Id$ (aucune sélection) et $re : I^\mu \to I$ où chaque coefficient du vecteur résultat est déterminé par tirage aléatoire parmi les μ coefficients correspondant des parents.

Les stratégies $(\mu + \lambda) - ES$ et $(\mu, \lambda) - ES$

Finalement, les stratégies $(\mu + \lambda) - ES$ et $(\mu, \lambda) - ES$, introduites par Schwefel, correspondent respectivement soit à la survie des μ meilleurs individus provenant de l'union des μ parents et λ enfants obtenus par recombinaison puis mutation, soit à la sélection des $\mu < \lambda$ meilleurs enfants en remplacement de tous les parents. Hormis le schéma de sélection, une différence fondamentale avec les stratégies précédentes provient du fait que l'individu manipulé comporte non seulement les n coefficients à optimiser, mais également les paramètres de l'opérateur de mutation : les écarts-types s'auto-adaptent au cours du déroulement de l'algorithme. Plus formellement, un individu $\vec{a}$ est défini par :

$$\vec{a} = (\vec{x}, \vec{\sigma}, \vec{\alpha}) \in I = \Re^n \times \Re_+^{n_\sigma} \times [-\pi, \pi]^{n_\alpha}$$

avec : $n_\sigma \in [1..n]$ et $n_\alpha = (n - n_\sigma/2).(n_\sigma - 1)$. $\vec{x} = (x_1, ..., x_n)$ est le vecteur des coefficients à optimiser, $\vec{\sigma} = (\sigma_1, ..., \sigma_{n_\sigma})$ est le vecteur des variances associées à chaque coefficient, et $\vec{\alpha} = (\alpha_1, ..., \alpha_{n_\alpha})$ correspond au vecteur des covariances entre chaque couple de coefficients (il s'agit donc d'une demi-matrice). Afin de limiter la taille du codage d'un individu ainsi que le coût des calculs, le coefficient n_σ peut être inférieur à n ; par exemple, choisir pour n quelconque :

- $n_\sigma = 1$ et $n_\alpha = 0$ permet de spécifier une stratégie simple mettant en œuvre un unique écart-type pour tous les coefficients de $\vec{x}$;
- $n_\sigma = n$ et $n_\alpha = 0$ spécifie une stratégie contrôlant un écart-type spécifique à chaque coefficient ;

– $n_\sigma = 2$ et $n_\alpha = n - 1$ correspond à une stratégie dont le premier coefficient est associé à un écart-type spécifique σ_1, σ_2 étant l'écart-type associé à tous les autres coefficients.

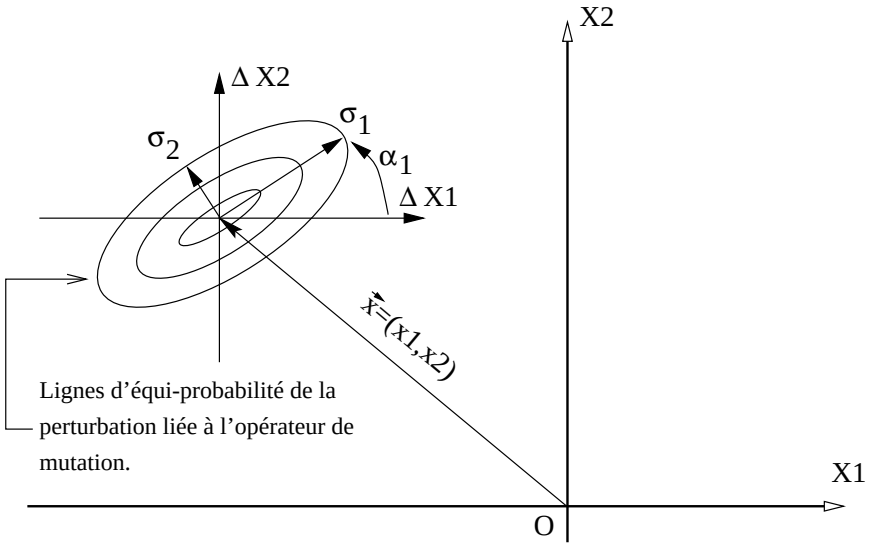


FIG. 2.6: Mutation auto-adaptative des stratégie $(\mu, \lambda) - ES$ et $(\mu + \lambda) - ES$ pour $n = 2$.

La figure 2.6 illustre les différentes composantes de $\vec{a}$ dans un espace à $n = 2$ dimensions avec $n_\sigma = 2$.

En notant $N(\xi, \sigma^2)$ un tirage donné suivant la distribution normale de moyenne ξ et de variance σ^2, chaque nouvel indice $N_i(\xi, \sigma^2)$ dénotant un nouveau tirage, et étant donnés les paramètres τ, τ' et β, il est possible de définir la mutation de l'individu $\vec{a}$ (notée $mut(\vec{a})$) comme la composée de trois opérateurs modifiant chacun une composante de $\vec{a}$: $mut(\vec{a}) = mu_x(\vec{x}, mu_\sigma(\vec{\sigma}), mu_\alpha(\vec{\alpha}))$, avec :

– $mu_\sigma(\vec{\sigma}) = (\sigma_1.exp(\tau'.N(0,1) + \tau.N_1(0,1)), ..., \sigma_{n_\sigma}.exp(\tau'.N(0,1) + \tau.N_{n_\sigma}(0,1)))$;

– $mu_\alpha(\vec{\alpha}) = (\alpha_1 + \beta.N_1(0,1)\ [-\pi, \pi]^4, ..., \alpha_{n_\alpha} + \beta.N_{n_\alpha}(0,1)\ [-\pi, \pi])$;

– $mu_x(\vec{x}) = (x_1 + cor_1(\vec{\sigma}, \vec{\alpha}), ..., x_n + cor_n(\vec{\sigma}, \vec{\alpha}))$.

$cor_i(\vec{\sigma}, \vec{\alpha})$ dénote une composante aléatoire obtenue par tirage suivant une distribution normale comportant éventuellement une partie corrélée (cf. figure 2.6, le lecteur trouvera de plus amples détails dans [Back96]). Schwefel conseille d'utiliser : $\tau \approx (\sqrt{2\sqrt{n}})^{-1}$, $\tau' \approx (\sqrt{2.n})^{-1}$ et $\beta \approx 0,0873$ ($\approx 5^o$).

La recombinaison réutilise les versions définies par $\omega \in [0..3]$ pour la $(\mu + 1) - ES$ sur chacun des vecteurs $\vec{x}$, $\vec{\sigma}$ et $\vec{\alpha}$: on note $(\omega_1, n_1, \omega_2, n_2, \omega_3, n_3)$ la recombinaison résultant de

[4]Tous les angles étant à valeur dans $[-\pi, \pi]$, en cas de dépassement une réduction modulo π est effectuée.

l'application de ω_1 à $\vec{x}$, ω_2 à $\vec{\sigma}$ et ω_3 à $\vec{\alpha}$, chacune des composantes opérant respectivement sur n_1, n_2 et n_3 individus.

Suite à de nombreuses expérimentations, Bäck et Schwefel conseillent :

- l'utilisation de la $(\mu, \lambda) - ES$;
- un choix de $\mu > 15$ et d'un ratio $\lambda/\mu \approx 7$;
- l'utilisation de la $(3, \mu, 2, \mu, 0, 1)$-recombinaison.

2.1.4 La programmation évolutionnaire

La programmation évolutionnaire, introduite dès 1962 par Fogel, se place dans la lignée des nombreux travaux de l'époque consacrés à la programmation automatique. Prenant acte des difficultés rencontrées par Friedberg qui essaie en 1958 de générer des programmes très simples en langage machine (ce dernier introduit les notions clé de mutation et d'évaluation sous les formes respectives d'un opérateur permutant deux instructions du programme ou modifiant aléatoirement l'une des instructions d'une part, et un *success number* mesurant l'adéquation des instructions afin de contrôler le taux de mutation d'autre part), Fogel a préféré orienter sa recherche vers la prédiction séquentielle de symboles à l'aide d'automates d'états finis représentés par des tables de transitions. Comme pour ce qui concerne les travaux de Friedberg, la mutation opère une modification aléatoire de l'individu (la table des transitions), et la sélection correspond à la variante mise en œuvre dans la stratégie d'évolution $(1 + 1) - ES$. Un individu est évalué par le pourcentage de lettres correctes de la séquence qu'il réussit à prédire. De même que les premières stratégies d'évolution, et contrairement aux algorithmes génétiques, la programmation évolutionnaire ne met pas en œuvre de mécanisme de recombinaison.

L'ampleur des difficultés rencontrées ainsi que les raisons de l'abandon dans les années 70 de la programmation évolutionnaire ont été diversement appréciées et analysées : selon D.E. Goldberg, l'ampleur de ce qu'il présente comme un échec fut tel qu'il desservit également le développement des algorithmes génétiques à leurs débuts : *The rejection of this work by the artificial intelligence community, more than any other single factor, was responsible for the widespread skepticism faced by more schema-friendly genetic algorithms of the late 1960s and mid-1970s.* ([Goldberg89], p.105), et la raison des difficultés rencontrées par Fogel provient, toujours d'après lui, de l'absence de croisement : *The evolutionary programming of Fogel (.../...) was insufficiently powerful to search other than small problem spaces quickly. We turn away from these and similar studies that ignore the fundamental importance of structure recombination (.../...)* ([Goldberg89], p.106). L'analyse de T. Bäck est beaucoup plus nuancée : les difficultés rencontrées par Friedberg auraient causé en partie l'ignorance des travaux de Fogel, et d'autre part ces derniers seraient venus trop tôt compte tenu des puissances de calcul extrêmement modestes disponibles à l'époque ([Back96], p. 59). Vers la fin des années 80, D.B. Fogel reprit les travaux de son père dans le but de les appliquer aux problèmes d'optimisation de paramètres continus : les méthodes proposées sont très similaires aux stratégies d'évolution puisqu'elles mettent en place des mécanismes d'auto-adaptation basés sur l'ajustement des variances et des

covariances des paramètres.

2.1.5 La programmation génétique

Historique

La programmation génétique (PG) a été proposée par J.R. Koza [Koza89, Koza92] dans le but de travailler sur des structures directement utilisables par la machine et (si possible après simplification) compréhensibles par l'homme. En particulier, ces travaux permettent la levée des contraintes de représentation (codage des individus sous forme de chaînes binaires de taille fixe) rencontrées dans le cadre de la version canonique des algorithmes génétiques. Bien qu'il soit possible de trouver les racines de la programmation génétique d'une part dans la programmation évolutionnaire quand à sa problématique, et d'autre part dans le langage `TB` [Cramer85] quand à la représentation utilisée, Koza est reconnu comme le fondateur de ce domaine car il en a posé le cadre (en insistant à la fois sur les similitudes et les différences entre PG et AG) et a contribué à son développement par des études systématiques sur un très grand nombre de problèmes. Plutôt que d'apprendre des automates d'états finis, les travaux de Koza se sont initialement portés vers le langage LISP pour les raisons suivantes :

- tous les objets LISP (programmes et données) sont des S-expressions, c'est-à-dire des listes syntaxiquement correctes qui peuvent également être représentées sous forme d'arbres ;
- LISP n'est pas un langage typé et par conséquent impose peu de contraintes sur les opérateurs manipulant les programmes, du moment que des contraintes de base telles que le respect de l'arité des fonctions sont vérifiées ;
- la manipulation des listes est extrêmement aisée ; de plus les programmes obtenus peuvent être immédiatement évalués (aucun mécanisme de compilation n'est nécessaire).

Espace de recherche

En pratique, l'utilisateur doit spécifier :

- l'ensemble T des terminaux (c'est-à-dire les symboles de constantes et de variables du langage) ;
- l'ensemble F des fonctions primitives : $F = \bigcup_{i=1}^{n} f_i$ est l'ensemble des symboles de fonction ; on notera $|f_i|$ l'arité de la fonction f_i ;
- la profondeur maximale des arbres (il s'agit de l'un des paramètres de l'algorithme, cf. par exemple [Koza92] p. 114).

Afin de permettre les générations stochastiques d'expressions et les recombinaisons, ces ensembles doivent vérifier l'hypothèse de clôture, spécifiant que toute combinaison syntaxiquement correcte d'éléments de F et T est correcte et évaluable. Par exemple, dans le cas d'un problème relatif à la découverte d'une loi numérique liant trois variables a, b et c, l'utilisateur pourrait choisir $F = \{+, -, *, /, sin, cos, \sqrt{}\}$ et $T = \{a, b, c, \Re\}$ où $\Re$ dénote l'ensemble des réels pouvant être représentés sur la machine. Afin de respecter la propriété de clôture, certaines des fonctions devront être étendues : $x \to \sqrt{x}$ sera ainsi remplacée par $x \to \sqrt{|x|}$ et $(x, y) \to x/y$ sera définie en $y = 0$ en posant $x/0 = 1$.

Opérateurs

Le croisement et la mutation, qui proviennent des algorithmes génétiques, ont été adaptés aux structures arborescentes. Comme le montre la figure 2.7, le croisement consiste à échanger deux sous-arbres appartenant chacun à un parent : si la propriété de clôture est respectée, les deux arbres résultant seront bien des S-expressions sémantiquement valides. Les points de coupure (représentés par des traits gras) sont déterminés suivant une loi favorisant les liens de profondeur faible. La mutation (qui est volontairement omise par

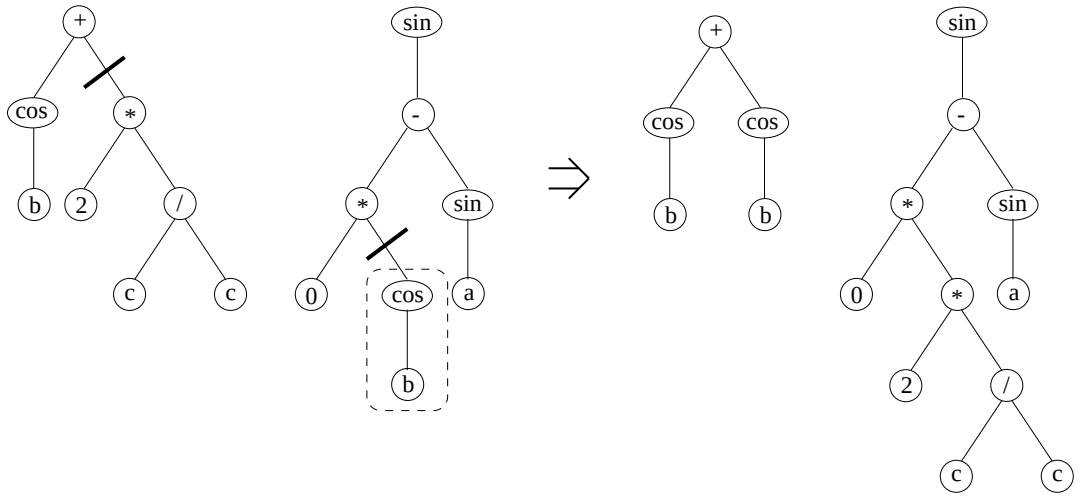


FIG. 2.7: Exemple de croisement en programmation génétique.

Koza dans un grand nombre d'expérimentations) modifie localement un individu en remplaçant un nœud choisi au hasard par un sous-arbre généré aléatoirement, en changeant un symbole de fonction par un autre de même arité ou en modifiant un symbole terminal.

De nombreuses expérimentations sont à l'origine de certains choix concernant les autres opérateurs d'évolution :

- Koza préconise l'utilisation d'un schéma de sélection *ad-hoc* copiant sans modification 10% des individus choisis suivant une probabilité proportionnelle à leur qualité. Expérimentalement, les sélections peu sensibles à la performance absolue comme par exemple la sélection par tournoi, basée uniquement sur l'ordre des individus,

donnent également de bons résultats.

– La variance de la qualité des individus générés aléatoirement lors de la création de la population initiale pouvant être très importante, un mécanisme appelé décimation empêche dans un premier temps la duplication des meilleurs individus afin d'éviter que ces derniers ne provoquent la convergence prématurée de l'algorithme.

Comportement expérimental

La quasi-totalité des travaux concernant la programmation génétique étant de nature applicative et expérimentale, peu de résultats théoriques sont disponibles (la plupart s'attachent à reprendre le théorème des schémas). Cependant, il est possible de dégager expérimentalement trois grandes caractéristiques de cette approche.

La première a trait à la sensibilité de l'algorithme au choix des fonctions primitives ainsi qu'à d'autres paramètres liés à la taille de l'espace de recherche, comme par exemple la profondeur maximale autorisée des arbres. Par exemple [BT et al.96], qui procède à une étude systématique pour évaluer et comprendre le rôle des paramètres-clé en programmation génétique dans le domaine de la classification par induction d'hypersurfaces, remarque que l'algorithme est très sensible à l'ajout de nouvelles primitives, et encore plus au choix de la fonction de qualité (les tests ont porté sur un critère entropique et une fonction de coût quadratique). Par contre, l'ajout de variables non pertinentes, la complexité de la formule-cible ou encore l'introduction de constantes ont peu d'incidence sur la qualité des résultats.

La seconde caractéristique concerne l'augmentation de la taille des individus au fur et à mesure du déroulement de l'évolution, par opposition aux autres méthodes qui travaillent sur des structures de taille fixe. Cet accroissement de la complexité des individus est la source de nombreux inconvénients que ce soit du point de vue de la quantité de mémoire requise, du temps d'évaluation des individus ou de la compréhensibilité pour l'utilisateur ; il a conduit au développement de plusieurs techniques allant de la manipulation directe du langage machine à la compression des arbres à l'aide de graphes acycliques orientés. Cette dernière méthode, combinée à la mémorisation au niveau des nœuds du graphe des résultats des évaluations, s'est révélée extrêmement efficace en espace et en temps [Ehrenburg96].

La dernière caractéristique de la programmation génétique, fortement liée à l'augmentation de la taille des arbres, est la création dynamique d'introns. Par analogie avec son homologue biologique, l'intron artificiel correspond à un segment non codant du chromosome et permettrait d'atténuer l'aspect destructeur du croisement (la figure 2.7 en donne un exemple délimité en pointillés). De ce point de vue, l'accroissement de la taille des arbres (qui correspond expérimentalement à l'apparition d'introns), correspondrait à un ajustement dynamique des structures destiné à favoriser la conservation des parties performantes. Cette hypothèse a été validée expérimentalement : [Haynes96] montre

que la destruction des introns par l'utilisation d'une procédure de simplification des arbres provoque une dégradation très importante des performances dûe à la convergence prématurée de l'algorithme, alors que la diminution du taux des individus simplifiés permet de revenir à la performance optimale (lorsque ce taux est nul). En conséquence, la procédure de simplification d'arbres à l'aide de règles de réécriture de type `(* 0 X)` $\Rightarrow$ `0` décrite par Koza [Koza92] ne doit être appliquée que pour des raisons d'intelligibilité une fois le résultat final obtenu, et non au cours de l'évolution.

Quelques variations

Deux extensions fort utiles de la méthode de base doivent être mentionnées. La première concerne la suppression de l'hypothèse de clôture à l'aide d'un typage fort permettant également la prise en compte des types génériques [Montana95]. Cette extension a été complétée dans [Haynes et al.96] par la gestion des hiérarchies de types et des types polymorphes. La seconde extension concerne la découverte automatique de procédures (ADFs, [Koza92, Koza94]) qui sont autant de blocs de construction intermédiaires facilitant la compréhension. Les procédures sont particulièrement importantes lorsque la solution du problème a une forte structure hiérarchique : il n'est pas envisageable d'augmenter la taille maximale des arbres dans une proportion qui permettrait la découverte de la fonction développée car la taille de l'espace de recherche serait alors beaucoup trop importante.

2.2 Algorithmes d'évolution et apprentissage automatique

Comme l'a montré le chapitre 1, l'apprentissage automatique supervisé peut être formulé comme un problème d'optimisation dans l'espace des règles. Puisque d'autre part les algorithmes d'évolution se sont avérés utiles et performants dans le domaine de l'optimisation, il n'est pas étonnant que de nombreux systèmes d'apprentissage basés sur ces algorithmes aient vu le jour. La grande majorité des travaux du domaine sont initialement apparus dans la communauté des algorithmes génétiques dont le champ d'investigation était particulièrement large ; le développement peu important de la programmation évolutionnaire, la spécification du cadre d'application des stratégies d'évolution aux problèmes d'optimisation de paramètres réels, et finalement la relative adaptation du codage binaire à la représentation de règles symboliques ont également contribué à cet état de fait. L'apparition (bien plus tard) de la programmation génétique, dont le biais de représentation est encore plus adapté à l'expression de formules complexes, a été la source de nombreux autres travaux dans le domaine de l'apprentissage.

Historiquement, deux courants correspondant à deux manières d'envisager l'apprentissage à l'aide des algorithmes génétiques ont vu le jour à la même période, et définissent de fait une dichotomie encore utilisée pour caractériser les systèmes [Venturini96] :

- L'approche de Michigan a été proposée par Holland [Holland86] et considère que l'ensemble de la population représente une base de connaissances, chaque individu correspondant à une règle particulière. Le résultat de l'apprentissage est alors un sous-ensemble d'individus de la population. Cette approche a donné naissance aux systèmes de classeurs.

- L'approche de Pittsburgh [Smith83] utilise au contraire l'algorithme génétique en se conformant au cadre des nombreux travaux appliqués à l'optimisation : le meilleur individu issu de l'évolution doit fournir la connaissance recherchée. En conséquence, dans le cas où le concept-cible peut être disjonctif, chaque individu code un ensemble de règles.

Les deux sections ci-dessous sont respectivement consacrées à la description des systèmes de classeurs (approche de Michigan), puis du système GABIL qui est l'un des représentants les plus connus de l'approche de Pittsburgh. La troisième section traite des travaux concernant la programmation génétique, et les deux dernières décrivent SIA ainsi que REGAL et G-Net.

2.2.1 Approche de Michigan - Les systèmes de classeurs

La problématique de l'animat

Les systèmes de classeurs (SC), [Holland86] sont des systèmes d'apprentissage automatique incrémentaux qui apprennent dynamiquement des règles de comportement sous forme de chaînes binaires. Ils ont été développés afin de répondre au problème de l'apprentissage adaptatif en environnement dynamique : le système est capable de s'auto-modifier au cours du temps afin de mieux s'adapter à son environnement. Les SC se différencient donc du cadre classique de l'apprentissage supervisé par plusieurs points :

- aucun oracle (ou professeur) n'est disponible pour fournir le comportement souhaité : l'apprentissage est non supervisé ;

- l'environnement sur lequel agit le système (et dont il reçoit des stimuli) peut varier : l'apprentissage, qui se fait par renforcement, doit être également adaptatif.

On parle alors de la problématique de l'animat [Wilson85, Wilson87], qui désigne un robot autonome dont la raison d'exister est la survie dans son environnement [Venturini94b, Wilson99].

Structure

Comme le montre la figure 2.8, un système de classeurs est constitué :

- D'une liste de messages (chaînes binaires de taille fixe) d'entrée et de sortie.

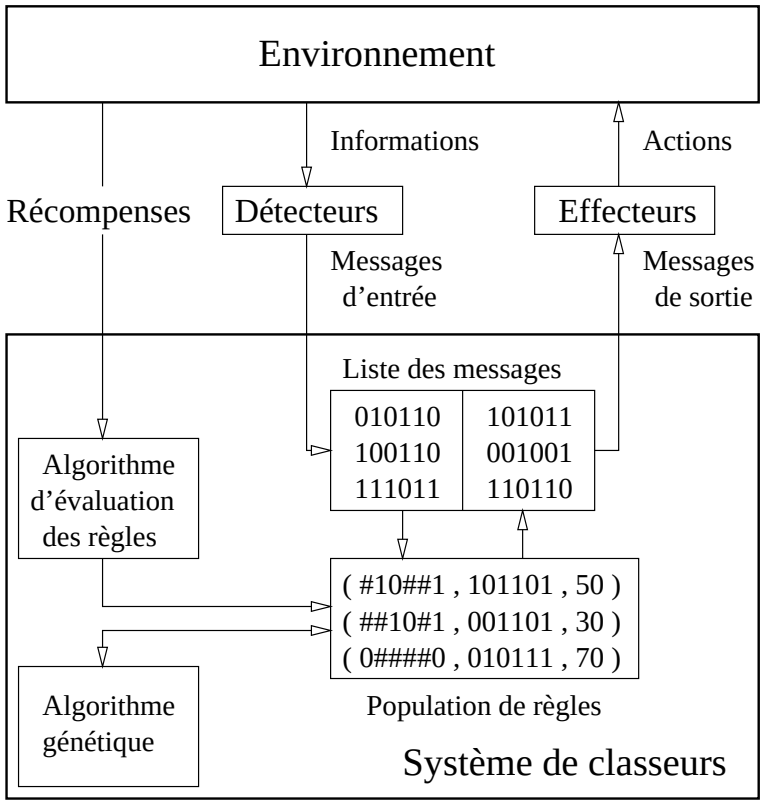


FIG. 2.8: Contexte d'application et structure interne d'un système de classeurs.

- D'une population de règles, appelées classeurs, de la forme "`( condition, action, force )`" où `condition` est une chaîne de taille fixe définie sur l'alphabet {`0,1,#`}, `action` est une chaîne binaire de taille fixe, et `force` est un nombre réel. Le caractère `#`, qui peut apparaître dans la partie `condition` du classeur, joue le même rôle que `*` pour les schémas.
- D'un algorithme d'évaluation de la qualité des règles répartissant les récompenses (ou les pénalités) associées aux actions entreprises.
- D'un algorithme génétique optimisant la population de règles afin d'améliorer et d'adapter constamment les comportements.

Fonctionnement général

Le cycle de fonctionnement d'un SC débute par la réception puis l'ajout dans la liste correspondante des messages provenant des détecteurs. Ces messages peuvent déclencher des classeurs dont la partie `action`, ajoutée dans la liste des messages, peut elle-même déclencher d'autres classeurs : contrairement à des méthodes telles que les systèmes experts, l'activation des règles a lieu en parallèle. Une fois que le système de gestion des messages a convergé vers un état stable (c'est-à-dire lorsque plus aucun classeur ne peut être déclenché), les messages restants sont envoyés aux effecteurs, et la récompense (ou la

pénalité) provenant de l'environnement est partagée par les classeurs concernés. L'apprentissage, destiné à introduire de nouveaux classeurs (potentiellement plus performants), est réalisé par un algorithme génétique soit à intervalles réguliers, soit lorsque les performances du système déclinent. Afin de ne pas modifier radicalement la base de règles au risque de dégrader les performances du SC, l'algorithme ne remplace qu'une partie de la population. L'évaluation des classeurs est fondée sur leur force.

Algorithme d'allocation des crédits

La méthode initiale, connue sous le nom de *bucket brigade*, a été proposée par Holland. Le SC est vu comme un système économique dans lequel les classeurs achètent et vendent de l'information : à chaque étape un classeur paie, en cédant une partie de sa force à l'objet l'ayant déclenché, le droit de placer un message dans la liste suivant la loi :

$$S_i(t+1) = S_i(t) + R_i(t) - P_i(t) - T_i(t)$$

où S_i dénote la force du classeur i considéré, R_i les récompenses reçues des classeurs déclenchés par i ou reçues de l'environnement, $P_i = \alpha.S_i(t)$ la force cédée par i dans le cas où il a été déclenché et a émis un message. $T_i(t) = \beta.S_i(t)$ correspond à un terme facultatif (une taxe) destiné à pénaliser les règles improductives. La figure 2.9 détaille

Classeurs	Liste des Messages	Gachettes	Enchères	Forces	Liste des Messages	Gachettes	Enchères	Forces	Liste des Messages	Gachettes	Enchères	Forces	Liste des Messages finale	Forces après renforcement
$(11\#, 001, 50)$	111	×	5	45	001			55	000			55		55
$(0\#1, 000, 50)$				50		×	5	45	101			50		50
$(00\#, 101, 50)$				50		×	5	45		×	5	40	101	65
Environnement				5										

FIG. 2.9: Déroulement d'un cycle de l'algorithme *bucket brigade*.

le fonctionnement de l'algorithmes *bucket brigade* sur une population de trois classeurs, avec un coefficient d'enchères $\alpha = 0,1$ et un taux d'imposition $\beta = 0$. Supposons que les détecteurs aient reçu de l'environnement un stimulus qui se traduise par le dépôt du message 111 dans la liste des messages. Puisque 111 ne vérifie que la première des trois conditions 11#, 0#1, et 00#, seul le premier classeur est activé (ce qui est représenté par une croix en face de ce dernier dans la colonne Gachettes). Le premier classeur paie donc $50 \times 0,1 = 5$ unités de force à l'environnement afin de placer son message (001) dans la liste des messages du tour suivant. Au second tour, l'unique message 001 active deux classeurs qui rémunèrent chacun de 5 unités le classeur émetteur. La propagation-consommation des messages se poursuit jusqu'à ce qu'il ne reste plus aucun classeur à activer. Le message restant (101) est alors envoyé aux effecteurs. En supposant que l'action sur l'environnement se traduise par une récompense de 25 unités, cette dernière sera attribuée au classeur qui avait posté ce message : le troisième classeur voit donc sa force augmenter de 40 à 65.

2.2.2 Approche de Pittsburgh - GABIL

GABIL [De jong et al.93, De jong et al.91] est un exemple récent de système suivant l'approche de Pittsburgh. Ce dernier apprend un concept booléen représenté sous la forme d'un ensemble disjonctif de règles propositionnelles. L'algorithme de recherche est en tout point similaire à un algorithme génétique, sauf en ce qui concerne le codage des individus : si celui-ci est bien binaire, la nécessité de pouvoir représenter des concepts disjonctifs (c'est-à-dire une disjonction d'un nombre quelconque de règles) a conduit à l'utilisation de chaînes binaires de taille variable.

Représentation des formules

La figure 2.10 détaille le génotype et le phénotype d'un individu composé de $r = 2$ règles dans le cas d'un domaine comprenant $n = 2$ attributs $Taille$ et $Couleur$ à valeur dans $\{petit, grand\}$ et $\{rouge, bleu, vert\}$.

Taille	*Couleur*	*Taille*	*Couleur*	*Classe*
1 0	0 1 1	0 1	1 1 1	1

$$Si\ [(Taille = petit) \wedge (Couleur = bleu \vee vert)] \vee [(Taille = grand)]\ Alors\ Classe = 1$$

FIG. 2.10: Représentation d'un individu dans GABIL.

À chaque attribut A_i est associée une sous-chaîne $a_i = [b_1, ..., b_{|A_i|}]$ dont chaque bit correspond à l'une des $|A_i|$ valeurs possibles. Un bit à 0 dénote l'absence de la valeur correspondante dans la disjonction, et 1 sa présence. Lorsque tous les bits sont à 0 ou à 1, le test portant sur l'attribut correspondant est abandonné. Toutes les règles R_j partagent le même patron P_R, résultant de la concaténation dans un ordre fixé des sous-chaînes de tous les attributs : $P_R = [a_1, ..., a_n]$. Pour finir, l'individu est obtenu en concaténant aux règles qui le composent un bit codant sa classe.

Opérateurs génétiques

La mutation correspondait initialement à l'opérateur canonique procédant par modification d'un bit avec la probabilité p_m. Elle a été ensuite déclinée sous deux formes implantant deux règles provenant de [Michalski84]. La mutation à tendance généralisatrice opère, lorsqu'elle est déclenchée, un changement de bit de 0 à 1 avec une probabilité de $0,75$, et le changement inverse avec une probabilité de $0,25$ seulement. Elle correspond à une version probabiliste des règles d'ajout et de suppression d'alternative. La mutation par abandon choisit un attribut dont la majorité des bits sont à 1 et remplace tous les bits restant par des 1 : il s'agit donc de l'opérateur de généralisation abandonnant une condition.

Contrairement au croisement à un point standard, l'opérateur de recombinaison de GABIL

permet l'ajout ou la suppression de règles : un point de coupure est déterminé sur chacun des parents de telle sorte que les deux points soient sémantiquement compatibles (c'est-à-dire situés entre les mêmes attributs), puis les deux sous-chaînes sont échangées (figure 2.11). Bien que ce mécanisme garantisse localement (ie. lors de l'application de l'opérateur) la conservation de l'espace mémoire, la possibilité de dupliquer lors de la sélection les individus les plus longs peut occasionner, tout comme dans le cas de la programmation génétique, une augmentation de l'espace mémoire nécessaire au déroulement du processus d'apprentissage.

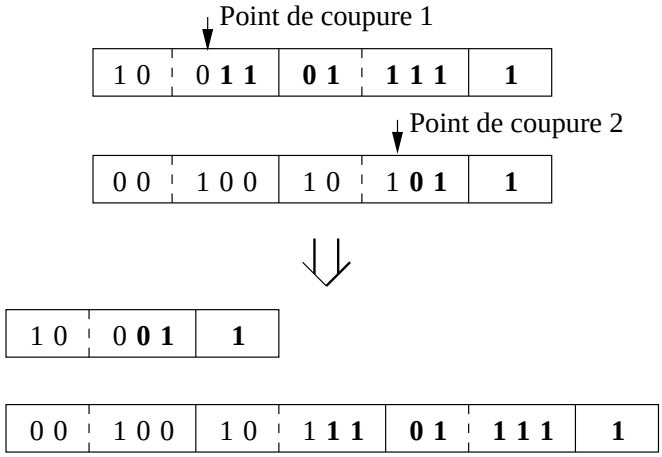


FIG. 2.11: Opérateur de recombinaison de GABIL.

Fonction de qualité

Soit p le pourcentage d'exemples correctement classés par l'individu I ; la fonction de qualité est définie par :

$$f(I) = p^2$$

Ce choix minimaliste provient de la volonté de combiner de manière très simple les aspects de cohérence et de complétude sans introduire de biais liés à la longueur des individus ou à leur complexité.

2.2.3 SIA

SIA (Symbolic Inductive Algorithm) [Venturini93, Venturini94a, Venturini94b] constitue le point de référence initial à partir duquel $\mathcal{SIAO}1$ a été développé. Ce système, dont la première application concerne l'analyse de données judiciaires, a été construit avec les objectifs suivants :

- analyser des données réelles : l'expert veut obtenir des connaissances intelligibles à partir de données pouvant être bruitées ;
- certains descripteurs sont symboliques et d'autres numériques ;

– les exemples peuvent être de taille variable (ils contiennent de nombreuses valeurs inconnues).

Ce système se distingue par son codage non binaire, puisqu'à chaque attribut est associé un gène représentant le domaine de généralisation. Cette représentation à l'avantage de permettre des traitements de haut niveau provenant non du cadre de l'optimisation dans un espace binaire mais du domaine plus spécifique de l'apprentissage supervisé symbolique et numérique. Ainsi, l'opérateur de mutation, qui est ici un opérateur de généralisation, peut utiliser des connaissances du domaine sous la forme de hiérarchies sur les constantes symboliques.

D'autre part, SIA met à la disposition de l'utilisateur (qui est supposé être un expert du domaine et non un expert en analyse de données) un petit nombre de paramètres simples lui permettant d'adapter l'algorithme à ses besoins : outre les hiérarchies reflétant les connaissances du domaine, il est possible de pondérer les exemples et de spécifier des biais intelligibles tels que la préférence de règles plutôt générales ou plutôt efficaces ainsi qu'une estimation du taux de bruit de la base.

Représentation des formules

Le codage des règles est réalisé en associant à chaque attribut A_i, $i \in [1, n]$, un gène pouvant représenter ses différentes généralisations dans le langage-cible. Ces dernières comprennent l'ensemble des intervalles convexes couvrant le noyau dans le cas des attributs numériques et les successeurs dans la hiérarchie des constantes ($*$ compris) dans le cas des attributs symboliques. À chaque attribut numérique est ainsi associée une paire de réels (le plus petit intervalle le contenant), et à chaque attribut symbolique une chaîne de caractères. SIA apprend des règles de la forme $r = (\mathbf{a}, C, f)$ où $\mathbf{a} = \{a_1, ..., a_n\}$ est un vecteur codant les domaines des attributs, C est la classe sur laquelle conclut la règle, et f sa force :

$$\begin{array}{rcll} & & \text{Si} & \text{NBPERS} \in [1.5,2.5] \\ r & = & \text{et} & \text{NATIO} = \text{française} \\ & & \text{alors} & \text{MOTCLASS} = \text{sans suite} \quad (f = 207) \end{array}$$

Chaque exemple est de la forme $ex = (\mathbf{e}, C_L, p)$ où $\mathbf{e}$ est un vecteur codant les valeurs des attributs, C_L est la liste des classes auxquelles ex appartient et p est le poids de l'exemple.

Traitement des valeurs inconnues

Dans la perspective d'adéquation de l'algorithme aux problèmes du monde réel, SIA offre un mécanisme sophistiqué de traitement des valeurs inconnues en les subdivisant en trois types :

– Une valeur est dite manquante (et dénotée par ?) lorsque la valeur de l'attribut considéré est définie mais n'a pas été reportée dans la base. Par exemple, dans le cas d'une application médicale, il est possible que l'état du patient ou la durée de

certains examens rendent impossible l'obtention d'une partie des données au moment où le système d'aide à la décision doit fournir sa réponse.

- Une valeur indifférente (notée *) est une valeur manquante dont l'omission est volontaire de la part de l'analyste qui postule que la valeur de l'attribut n'a aucune importance dans le cas de l'instance décrite. Par exemple, il est possible de supposer que la nationalité d'une personne ayant commis de nombreux crimes n'aura aucune importance sur la décision du tribunal, ce qui évite à l'expert de rechercher l'information dans le dossier.

- Les valeurs non définies (notées #) correspondent aux cas où la valeur de l'attribut n'a aucune signification pour l'instance considérée. Par exemple, l'attribut codant la nationalité des dossiers sans auteur n'a aucune signification et n'est donc pas défini pour de telles instances.

L'importance de la distinction faite au niveau sémantique entre les différents types de valeurs inconnues provient du constat que les algorithmes qui prennent en compte de telles valeurs transforment la base d'apprentissage[5] à l'aide de stratégies qui induisent toutes un certain nombre d'inconvénients (voir tableau 2.1).

Par opposition, la méthode mise en œuvre dans SIA consiste à :

- conserver, sans aucune modification, les valeurs non définies dans la base ;

- inclure la sémantique des valeurs non définies dans la fonction d'appariement ;

- s'aider de cette même sémantique lorsqu'une règle est générée à partir d'un noyau contenant des valeurs non définies.

En conséquence, l'appariement est restreint aux attributs connus ainsi qu'au symbole * (qui couvre tout symbole et n'est couvert que par lui-même). ? et # ne figurent pas dans les règles et ne peuvent être couverts que par *. La création des noyaux est détaillée au paragraphe décrivant les opérateurs génétiques.

Traitement des valeurs numériques

Les valeurs numériques sont conservées telles quelles dans les exemples. De plus, aucun pré-traitement (discrétisation, analyse statistique) n'est utilisé pour restreindre l'espace de recherche : il existe autant de seuils définissant les domaines qu'il y a de valeurs numériques, et le processus d'apprentissage se charge de découvrir quelles sont les valeurs significatives. Si $\{v_1, ..., v_m\}$ sont les valeurs rencontrées pour un attribut numérique, alors

[5] Ils peuvent le faire de manière indirecte par un traitement qui est équivalent à une réécriture. Par exemple C4.5 adopte une stratégie dynamique qui est une variante du pré-traitement remplacer ? par la valeur la plus probable dans l'échantillon d'apprentissage complétée par une diminution du poids de l'exemple considéré afin de minorer son influence lors du calcul de l'entropie.

Stratégie	**Conséquence**
Supprimer les instances contenant des valeurs non définies.	Perte d'informations. De plus certaines bases peuvent comporter au moins une valeur non définie par instance.
Remplacer ? par la valeur la plus probable.	La règle "Si NBPERS = ? et NATIO = française ..." peut être transformée en "Si NBPERS = 0 et NATIO = française ...", d'où un problème de sémantique.
Traiter ? comme un nouveau symbole de constante définie.	La signification d'une règle telle que "Si NBPERS = ? alors ..." (si elle existe) dépend du domaine et de la technique de récolte des données. De plus, cette stratégie ne peut être appliquée au cas des attributs numériques.
Remplacer * par toutes les valeurs possibles.	Le coût en espace dû à la duplication des instances ainsi que leur poids peuvent rendre l'apprentissage impossible (n valeurs indifférentes de modalité m_i causent un facteur de duplication de $\prod_{i=1}^{n} m_i$).
Répartir les exemples suivant # [Brunet93].	Un arbre d'héritage d'attributs doit pouvoir être défini sur le domaine. La segmentation peut conduire à des résultats contradictoires sur les sous-ensembles ainsi qu'à une perte de confiance statistique.

TAB. 2.1: Inconvénients des stratégies classiques de traitement des valeurs non définies.

les bornes potentielles associées sont : $\{-\infty, \frac{v_1+v_2}{2}, ..., \frac{v_{m-1}+v_m}{2}, +\infty\}$.

Fonction de qualité

La qualité d'une règle R est définie par :

$$Q(R) = max(\frac{BC(R) - \alpha.MC(R) + \beta.g(R) + \gamma.e(R)}{BC_{max}}, 0)$$

Elle croît (resp. décroît) proportionnellement à la somme des poids, notée BC (resp. MC), des exemples bien classés (resp. mal classés) par R. BC_{max} correspond à la somme des poids des exemples de la classe sur laquelle conclut R. $g(R)$ est une mesure de la généralité de R à valeur dans $[0, 1]$ de la forme $g(R) = \frac{1}{n}\sum_{i=1}^{n} g_i$, où g_i dénote la généralité du domaine a_i de R relatif à l'attribut A_i. $e(R)$ mesure l'explicabilité de R pour l'utilisateur qui peut définir pour ce faire des poids $p_1, ..., p_n$ caractérisant la préférence accordée aux attributs.

Les propriétés de la fonction de qualité sont reliées aux trois paramètres α, β et γ ajustables par l'utilisateur :

- Si β et γ sont négligeables devant 1, on peut montrer [Venturini94b] que la cohérence de R est nécessairement supérieure à $\frac{\alpha}{\alpha+1}$. L'expert du domaine peut ainsi paramétrer

α afin d'obtenir des règles parfaitement cohérentes (en choisissant $\alpha > |\mathcal{B}|$) ou adapter cette contrainte afin de prendre en compte le bruit (par exemple $\alpha = 9$ force SIA à n'accepter que des définitions dont la cohérence est supérieure à 90%).

- β, qui est un paramètre à valeur dans $\{-10^{-3}; 0; 10^{-3}\}$ permet de spécifier un biais de généralité ($\beta = 10^{-3}$), de spécificité ($\beta = -10^{-3}$), ou de neutralité en ce qui concerne la généralité syntaxique des règles.

- γ ($0 \leq \gamma << 1$), permet de pondérer l'importance de la mesure d'explicabilité.

Tant que β et γ sont négligeables devant 1, le critère principal d'évaluation des règles est leur cohérence (sous la forme $BC - \alpha.MC$). Il est également important de noter que le paramètre α ne fait que fixer un seuil de cohérence minimal, mais que le système peut fournir des règles dont la cohérence est supérieure à ce seuil.

Opérateurs génétiques

Hormis le schéma de sélection, les opérateurs génétiques sont au nombre de trois : la création, la mutation et le croisement.

L'opérateur de création génère une règle R_{init} aussi spécifique que possible couvrant un exemple ex (le noyau). Pour ce faire, les attributs de ex sont transformés comme suit :

Attributs symboliques
: La valeur d'un attribut symbolique est recopiée si elle est définie. Dans le cas où elle vaut `#`, `*` ou encore `?`, elle est abandonnée.

Attributs numériques
: Dans le cas où la valeur de l'attribut numérique est définie, on génère le plus petit intervalle la contenant. Le traitement concernant les valeurs non définies reste le même.

L'opérateur de mutation, qui s'applique sur une règle R, généralise un gène déterminé aléatoirement. Dans le cas où le gène sélectionné code un attribut symbolique, l'opérateur peut utiliser la hiérarchie définie sur le domaine. En ce qui concerne les attributs numériques, les bornes des intervalles peuvent être étendues. De plus, dans les deux cas, il est également possible d'abandonner la condition portant sur l'attribut.

L'opérateur de croisement correspond au croisement uniforme défini dans [Syswerda89], avec une probabilité d'échanger deux gènes égale à $\frac{1}{2}$. Bien que les règles générées par cet opérateur puissent être incomparables, plus générales ou plus spécifiques que leurs parents, la propriété de conservation de la couverture du noyau ex à partir duquel R_{init} est générée est toujours vérifiée. En effet, les domaines a_i de tous les individus couvrent (par construction) la valeur e_i correspondante de ex.

Évolution des règles

Les règles sont apprises successivement suivant le principe de couverture itérative décrit chapitre 1, section 1.3.4. L'algorithme de développement du noyau procède par évolution d'une population P de 50 règles. La population initiale est générée en insérant des règles obtenues par mutation de R_{init} de telle sorte que seules les règles dont la qualité est supérieure à celle de $Q(R_{init})$ sont insérées dans P. Lorsque la taille maximale est atteinte SIA fait évoluer la population de règles de la manière suivante :

1. Appliquer l'un des opérateurs ci-dessous. La sélection (si nécessaire) des règles fournies en argument de l'opérateur se fait par tirage uniforme dans la population.
   - Création (prob. par défaut de 10%) : généraliser R_{init} pour obtenir une règle R'.
   - Mutation (prob. par défaut de 80%) : sélectionner une règle et appliquer la mutation pour obtenir R'.
   - Croisement (prob. par défaut de 10%) : sélectionner deux règles et appliquer le croisement pour obtenir R' et R''.
2. Insérer R' (resp. R'') dans la population seulement si elle n'y figure pas déjà et :
   - si R' a été engendrée par mutation à partir d'une règle R, alors R' remplace R dans la population seulement si $Q(R') > Q(R)$;
   - sinon R' (resp. R'') n'est insérée que si sa qualité est supérieure à celle de la plus mauvaise règle R_{pire} de la population. Dans ce cas R' (resp. R'') remplace R_{pire}.
3. Si la meilleure règle de la population est restée inchangée depuis Nb_{max} itérations, alors rendre la (les) meilleure(s) règles de $P \cup R_{init}$, sinon aller en 1.

Plusieurs algorithmes de filtrage sont proposés dans [Venturini94b] afin de supprimer les règles redondantes ou bruitées, ce qui permet d'obtenir l'ensemble de règles $\mathcal{R}$ final.

Classification

La classification d'une nouvelle instance ex à partir de la base de règles $\mathcal{R}$ se fait au moyen d'une mesure de distance $d(ex, R)$. Plus précisément, si $\mathcal{R}^*$ est l'ensemble des règles de $\mathcal{R}$ dont la distance à $ex = ([e_1, ..., e_n], C_L, p)$ est minimale, alors ex est de la classe de la règle $R^* = ([a_1, ..., a_n], C, f) \in \mathcal{R}^*$ dont la qualité est la plus élevée. La distance est de la forme : $d(ex, R) = \frac{1}{n(r)} \cdot \sqrt{\sum_{i=1}^{n} d_i^2}$ où $n(R)$ est le nombre de conditions non abandonnées de R^* et d_i est définie par :

- si $cond_i$ a été abandonnée dans R^*, alors $d_i = 0$;
- si A_i est symbolique et si $e_i \in a_i$ alors $d_i = 0$ sinon $d_i = 1$;
- si A_i est numérique et $[B, B']$ sont les bornes de a_i, alors si $e_i \in [B, B']$ alors $d_i = 0$, si $e_i > B'$ alors $d_i = \frac{e_i - B'}{max_i - min_i}$ et si $e_i < B'$ alors $d_i = \frac{B - e_i}{max_i - min_i}$.

2.2.4 REGAL et G-Net

Le système REGAL [Angelano et al.97, Neri97] présenté en détail ci-dessous, est une évolution du système GA-SMART [Giordana et al.92]. L'objectif de ces systèmes, qui font référence en apprentissage supervisé basé sur les algorithmes génétiques, est d'apprendre des formules de la logique relationnelle étendue aux disjonctions internes. Pour ce faire, les auteurs ont développé :

- un codage des formules adapté aux chaînes de bits de taille fixe ;
- des opérateurs spécifiques aux problèmes d'apprentissage supervisé de concepts disjonctifs (les croisements dirigés généralisant et spécialisant, ainsi que l'opérateur de sélection basé sur le suffrage universel).

De plus, l'algorithme peut être réparti sur plusieurs machines, et une version dédiée aux grappes de stations de travail a été élaborée (il s'agit du système G-Net [Angelano et al.98, Lo bello98]).

Représentation des formules

Contrairement aux langages d'ordre 0 définis sur des attributs à valeurs discrètes et dont le codage des formules correspond naturellement aux chaînes binaires de taille fixe utilisées par l'AGC, la représentation des formules de la logique relationnelle est plus difficile à mettre en œuvre :

- les formules de la logique relationnelle peuvent être de taille quelconque ;
- le langage de la logique relationnelle introduit des quantificateurs et des symboles de variable ;
- les domaines d'application mettant en jeu de nombreuses valeurs numériques font croître la taille des chromosomes.

La solution adoptée par G-NET consiste à limiter l'espace de recherche à l'aide d'une formule générique Φ, fournie par l'utilisateur. Cette formule générique est alors décomposée en :

- une partie fonctionnelle typée (éventuellement fonctionnelle typée et relationnelle) notée Φ_{LP} (*learnable predicates*) ;
- une partie purement relationnelle Φ_{CP} (*constraint predicates*), éventuellement vide.

Par exemple, la formule générique $\Phi = Color(X, \texttt{col}) \wedge Distance(X, Y, \texttt{dist}) \wedge Follow(X, Y)$ (dont la partie fonctionnelle typée est marquée en caractères d'imprimerie) peut être décomposée en $\Phi_{LP} = Color(X, \texttt{col}) \wedge Distance(X, Y, \texttt{dist})$ et $\Phi_{CP} = Follow(X, Y)$. Il est alors possible, connaissant les domaines des différents types du langage, de coder la partie fonctionnelle sur une chaîne de bits de taille fixe en utilisant un mécanisme similaire à GABIL. Chaque bit du chromosome code ainsi la présence (`1`) ou l'absence (`0`) d'un

symbole de constante donné dans une disjonction interne. La partie contrainte, qui n'est pas modifiée par l'algorithme, ne figure pas dans le codage du chromosome.

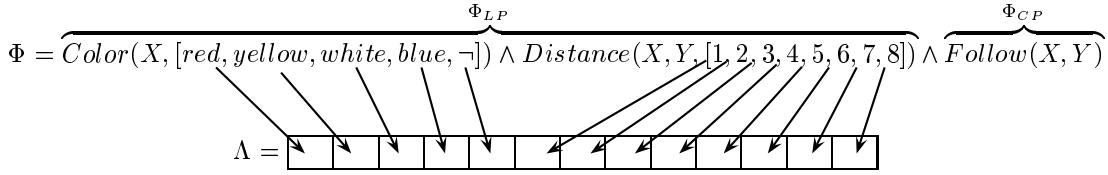


FIG. 2.12: Codage de la formule générique $\Phi = \Phi_{LP} \wedge \Phi_{CP}$ dans une chaîne binaire de taille fixe Λ [Giordana et al.97].

L'ensemble des constantes d'un type donné pouvant être très important, l'utilisateur a la possibilité de définir des disjonctions internes incomplètes : ces dernières ne comportent que certaines constantes choisies, ce qui diminue d'autant la taille des chromosomes. Les prédicats contenant de telles disjonctions sont également dits incomplets (*incomplete predicates*, cf. [Neri97] page 71).

De plus, un marqueur noté $\neg$ code la négation interne, ce qui permet d'étendre l'expressivité des disjonctions incomplètes puisqu'il est alors possible de faire référence par complément à des ensembles de symboles non codés dans la disjonction. Lorsque tous les bits d'une disjonction complète (y compris l'étoile) sont à 1, le prédicat correspondant est abandonné. Ainsi, la formule générique Φ dont le codage Λ est détaillé figure 2.12 permet de générer (entre autres) les formules φ_i correspondant aux chaînes $s(\varphi_i)$ suivantes :

Λ	=	red	yellow	white	blue	$\neg$	1	2	3	4	5	6	7	8
$s(\varphi_1)$	=	0	1	1	0	0	0	0	1	0	0	0	0	0
$s(\varphi_2)$	=	1	1	0	0	1	0	0	0	0	1	1	1	0
$s(\varphi_3)$	=	1	1	1	1	1	0	1	0	0	0	0	0	0

$\varphi_1 = Color(X, [yellow \vee white]) \wedge Distance(X, Y, [3]) \wedge Follow(X, Y)$
$\varphi_2 = Color(X, [\neg(white \vee blue)]) \wedge Distance(X, Y, [5 \vee 6 \vee 7]) \wedge Follow(X, Y)$
$\varphi_3 = Distance(X, Y, [2]) \wedge Follow(X, Y)$

Fonction de qualité

La qualité f d'une formule φ est définie empiriquement par :

$$f(\varphi) = (1 + A.z(\varphi)).e^{-w(\varphi)} + \varepsilon$$

où A est un paramètre (qui vaut $0,1$ par défaut) fourni par l'utilisateur ;

$$z(\varphi) = \frac{nombre\ de\ bits\ à\ 1}{longueur\ du\ chromosome}$$

est une mesure de la simplicité de l'hypothèse ; $w(\varphi)$ correspond au nombre d'instances négatives couvertes par φ et ε est une constante positive très petite.

Opérateurs génétiques

Les opérateurs de G-Net sont multiples : alors que certains (la mutation et les croisements classiques) proviennent du cadre des algorithmes génétiques binaires classiques, d'autres (les croisements dirigés, la création de noyaux et la sélection au suffrage universel) ont été développés spécifiquement pour la tâche d'apprentissage.

Les croisements sont au nombre de quatre : le croisement à deux points [Holland75], le croisement uniforme [Syswerda89], le croisement spécialisant et le croisement généralisant. Si les deux premiers sont bien connus, les deux croisements dirigés ainsi que la procédure de choix du croisement à appliquer ont été proposés pour le système GA-SMART [Giordana et al.92] dans le cadre de la représentation des disjonctions internes sous forme de chaînes de bits. Puisque le changement d'un bit de `0` à `1` (resp. de `1` à `0`) ajoute (resp. supprime) une constante dans une disjonction, l'application d'un ou-logique (resp. et-logique) sur les bits codant une disjonction aura pour effet de généraliser (resp. spécialiser) cette dernière. Le croisement généralisant (resp. spécialisant) consiste donc à appliquer un ou-logique (resp. et-logique) aux bits des deux parents qui correspondent à un ensemble δ de littéraux déterminés aléatoirement. Chaque individu est sélectionné pour un croisement avec une probabilité $p_c = 0,6$.

Lorsque deux individus φ_1 et φ_2 ont été déterminés, le choix d'un des quatre croisements se fait suivant la règle ci-dessous où p_u, p_{2pt}, p_s et p_g représentent repectivement la probabilité d'effectuer un croisement uniforme, un croisement à deux points, un croisement spécialisant et un croisement généralisant :

$$\begin{aligned} p_u &= (1 - a.f_n).b \\ p_{2pt} &= (1 - a.f_n).(1 - b) \\ p_s &= a.f_n.r \\ p_g &= a.f_n.(1 - r) \end{aligned}$$

a et b sont des paramètres à valeur dans $[0,1]$ ajustables par l'utilisateur et valant par défaut $0,7$ et $0,4$. a correspond à la préférence d'appliquer un croisement dirigé plutôt qu'un croisement classique et b la préférence au croisement uniforme plutôt qu'au croisement à deux points. f_n et r valent respectivement $\frac{f(\varphi_1)+f(\varphi_2)}{2.f_{max}}$ et $\frac{n^+(\varphi_1)+n^-(\varphi_1)+n^+(\varphi_2)+n^-(\varphi_2)}{(E+C)^2}$, où f dénote la fonction de qualité, f_{max} sa valeur maximale, $n^+(\varphi)$ (resp. $n^-(\varphi)$) le nombre d'exemples (resp. contre-exemples) couverts par l'hypothèse φ et $E = |\mathcal{E}|$ (resp. $C = |\mathcal{C}|$) le nombre d'exemples (resp. de contre-exemples) du concept-cible. Moins $f(\varphi_1)$ et $f(\varphi_2)$ sont importantes, plus f_n est petite et plus la probabilité de préférer un croisement classique à un croisement dirigé est grande. D'autre part, le ratio r est construit de telle sorte que la probabilité d'appliquer le croisement spécialisant soit proportionnelle à la somme des couvertures des deux hypothèses.

La mutation, qui correspond à la version classique de l'AGC, est appliquée avec un taux $p_m = 10^{-4}$. Elle est complétée par un opérateur de génération (le *seeding operator*), qui est également utilisé lors de la création de la population initiale. Cet opérateur crée une

formule φ, à partir d'un exemple positif ξ et à l'aide d'une chaîne de bits s (générée aléatoirement) en déterminant tout d'abord aléatoirement un appariement satisfaisant les contraintes de Φ_{CP} entre les littéraux de ξ et ceux du modèle générique Φ puis en généralisant s aussi peu que possible (c'est-à-dire en minimisant le nombre de bits de s qu'il faut mettre à 1) afin d'assurer qu'elle couvre ξ. L'opérateur de génération permet ainsi d'augmenter d'une part la couverture de l'espace de recherche en enrichissant la diversité génétique de la population et d'autre part la couverture des exemples à l'aide d'un mécanisme similaire à la création du noyau à partir d'une règle R_{init} qui est utilisé par SIA.

La sélection au suffrage universel

Le schéma de sélection, qui est l'un des apports majeurs de l'algorithme, est à la fois très original et très bien adapté à la problématique de l'optimisation multimodale dans le cadre de l'application des algorithmes génétiques à l'apprentissage supervisé de concepts. Il a été baptisé « sélection au suffrage universel » car à chaque génération ce sont des exemples du concept-cible qui votent pour les règles qui les couvrent ; de plus les exemples ont une probabilité identique d'être choisis et chacun dispose d'une voix. Ainsi, si M est la taille de la population A, et $0 < g \leq 1$ son taux de renouvellement, la sélection extrait avec replacement $g.M$ exemples ξ_i du concept-cible. Chaque exemple ξ_i choisit alors une règle au moyen d'un tirage de la roulette r_i constituée de toutes les règles de A couvrant ξ_i (la probabilité de tirer une règle φ_j de la roulette est proportionnelle à sa qualité $f(\varphi_j)$). Dans le cas où aucune règle de la population ne couvre ξ_i, une nouvelle formule est créée à l'aide de l'opérateur de génération. Contrairement aux techniques classiques de création de niches écologiques et de formation d'espèces, la sélection au suffrage universel présente les propriétés suivantes :

1. à tout moment de l'évolution, la couverture (stochastique) de l'ensemble d'apprentissage par les formules est garantie ;
2. il n'est pas nécessaire de définir une distance entre deux individus ;
3. la fonction de qualité est utilisée localement pour comparer des individus couvrant le même exemple.

Ces propriétés sont particulièrement adaptées au contexte de l'apprentissage supervisé de concepts disjonctifs. D'une part, la première leur est spécifique et ne peut être obtenue par une technique proposée dans le cadre plus général de l'optimisation. D'autre part, la seconde propriété est souhaitable lorsque le langage-cible est constitué de formules de la logique relationnelle puisque la définition d'une distance dans un tel langage est très difficile à élaborer. Enfin, la dernière propriété permet une évaluation des règles dirigée par la couverture locale.

Évolution des règles

L'architecture distribuée de REGAL comprend un ensemble de sous-populations (ou dèmes) $\{Pop_n\}$ distribuées sur les processeurs ainsi qu'un processus central appelé su-

perviseur. À chaque génération t, le dème Pop_n reçoit des autres dèmes un ensemble d'individus $A_{net}(t)$ tel que $|A_{net}(t)| = \mu.|A_n(t)|$ où μ est le taux de migration et $A_n(t)$ l'ensemble d'individus de la population courante. Chaque dème Pop_n applique alors localement la sélection par suffrage universel à l'aide d'un sous-ensemble spécifique $\mathcal{E}_n$ des exemples d'apprentissage sur $A_n(t)$ et $A_{net}(t)$. De plus, $\rho.|A_{net}(t)|$ individus (où ρ correspond au taux de reproduction extra-dème) sont sélectionnés aléatoirement parmi $A_{net}(t)$ afin de participer également à la reproduction. Enfin, une partie des enfants est insérée avec remplacement dans $A_n(t+1)$ alors qu'une autre partie est envoyée aux autres dèmes sur le réseau. Le meilleur individu (au sens d'une mesure $\pi(best_n, \mathcal{E}_n) = Pos_n.f(best_n)$ où Pos_n correspond à la couverture de $best_n$ sur les exemples du concept-cible) est alors envoyé au superviseur et la génération suivante peut débuter.

Le rôle du superviseur consiste à maintenir la liste des meilleures définitions courantes et à assigner à chaque dème un ensemble $\mathcal{E}_n$ adapté, c'est-à-dire permettant la caractérisation d'un sous-concept.

2.3 Récapitulatif - Motivation de $\mathcal{SIAO}1$

Ce chapitre a été l'occasion de présenter les algorithmes d'évolution, dont le fonctionnement général s'inspire de la simulation du développement de populations d'individus. En pratique, ces algorithmes ont connu un grand succès en tant que méthodes d'optimisation générales, car elles sont applicables à un très vaste ensemble de problèmes sans pour autant nécessiter d'investissement très important que ce soit sur le plan du développement du logiciel, ou sur le plan du coût d'adaptation de la méthode au problème à résoudre.

En particulier, les algorithmes d'évolution ont fait l'objet d'études et ont été appliqués avec succès dans le domaine de l'apprentissage automatique. Cependant, de par leur généralité même, une comparaison avec les approches dédiées à l'apprentissage automatique supervisé symbolique montre qu'un travail d'adaptation aux spécificités du problème est nécessaire. En effet :

- Du point de vue de l'expressivité de $\mathcal{L}_h$, il semble difficile de trouver un juste milieu entre les chromosomes binaires (resp. réels) de taille fixe mis en œuvre dans l'AGC (resp. les stratégies d'évolution) d'une part, et les arbres très peu contraints utilisés en programmation génétique d'autre part. Ce problème est particulièrement épineux dans le domaine de la logique du premier ordre, où il est nécessaire de trouver un compromis au dilemme provenant de la relation liant l'expressivité de $\mathcal{L}_h$ à la taille de l'espace de recherche d'une part et à la complexité calculatoire du test de couverture d'autre part.

- Du point de vue de la stratégie de recherche, les méthodes évolutionnaires sont pour la plupart caractérisées par un déplacement non dirigé, redondant et peu informé au sein de l'espace de recherche, alors que le théorème de conservation donne sur le plan théorique de bons arguments pour que l'algorithme de recherche exploite autant que possible toutes les connaissances spécifiques au domaine.

– Enfin, la problématique des bases de données de taille importante (qui se retrouve dans de nombreuses applications pratiques en fouille de données) doit également être prise en compte.

Le système $\mathcal{SIAO}1$, détaillé dans les chapitres suivants, propose – entre autres – une réponse originale à chacun des points évoqués ci-dessus :

– L'espace de recherche $\mathcal{L}_h$ résulte d'un compromis entre les formes fixes (comme les patrons prédéfinis par l'utilisateur) et les formes quelconques (provenant de la programmation génétique). Les hypothèses initiales sont en effet générées à l'aide d'un exemple (ou noyau), ce qui fixe une forme initiale. Celle-ci est alors modifiée au cours de la recherche, non seulement à l'aide d'opérateurs de généralisation classiques tels que l'abandon de prédicat, mais également à l'aide d'un opérateur de croisement générant un « presque moindre généralisé » de la définition courante et d'un exemple du concept-cible non couvert par cette définition. L'algorithme peut ainsi créer des formes à la fois pertinentes et totalement nouvelles, c'est-à-dire différentes des formes des exemples.

– En ce qui concerne la stratégie d'exploration, l'ordre partiel défini sur $\mathcal{L}_h$ est mis à profit puisque la recherche est dirigée (ascendante). De plus, la redondance est interdite par le schéma de sélection, ce qui permet également de diminuer le risque de convergence prématurée. Enfin, la recherche est informée puisque les opérateurs sont, autant que faire se peut, dirigés par les données : on espère ainsi améliorer l'efficacité de l'exploration par rapport aux systèmes reproduisant exactement le cadre évolutionnaire qui cherchent « en aveugle ».

– Les algorithmes d'évolution étant gourmands en puissance de calcul, de nombreux travaux concernent leur parallélisation. Cependant, alors que la totalité des travaux relatifs aux algorithmes d'évolution parallèles proposent de répartir si possible dynamiquement la charge de calcul (la plupart du temps en éparpillant la population sur différents processeurs), $\mathcal{SIAO}1$ répond également à la problématique de la répartition de la charge mémoire, lorsque cette dernière devient insuffisante pour permettre à chaque processeur de contenir en mémoire vive (RAM) l'ensemble de la base d'apprentissage.

Chapitre 3

Le biais de langage

Ce chapitre est dédié à l'étude du biais de langage de $\mathcal{SIAO}1$, en particulier du point de vue de la complexité du test de couverture[1]. En effet, comme l'a montré le chapitre 1 (section 1.5.1), la prise en compte du dilemme « coût de l'induction / richesse du langage » est fondamentale dans le contexte de l'apprentissage relationnel. La structure de l'argumentation est donc divisée en deux grandes parties interdépendantes.

La première partie est centrée autour du langage de représentation de $\mathcal{SIAO}1$. Elle débute par une présentation de la problématique et des résultats théoriques relatifs à la logique du premier ordre en général et au test de θ-subsomption utilisé par certaines méthodes de PLI en particulier. Une seconde section présente le langage de $\mathcal{SIAO}1$ ainsi que la relation de subsomption choisie pour structurer l'espace. L'expressivité du langage ainsi que la complexité du test de subsomption sont également analysées au regard des résultats obtenus dans le cadre de la logique du premier ordre.

La seconde partie concerne l'étude et l'implantation d'un test de subsomption optimisé. Après avoir montré l'intérêt d'un tel test pour $\mathcal{SIAO}1$ dans le cas de problèmes fortement relationnels, l'architecture générale de l'algorithme de subsomption est esquissée, chacune des trois étapes (le pré-traitement, la réduction des domaines et la résolution avec propagation) étant détaillée dans une section spécifique. Finalement, les performances du test de subsomption sont validées sur un problème fortement relationnel.

À l'issue de ce chapitre, le lecteur aura pris connaissance des structures et algorithmes de base sur lesquels le système $\mathcal{SIAO}1$ est construit. En particulier, les choix qui ont été faits au niveau des biais de langage (représentation des connaissances, théorie du domaine, typage, complexité du test de subsomption) auront été explicités.

[1] Nous nous conformons par la suite à l'usage du vocabulaire de la logique qui utilise le terme « subsomption » pour rendre compte des notions de couverture ou d'appariement de formules.

3.1 Le langage de représentation de $\mathcal{SIAO}1$

3.1.1 Contexte théorique

La donnée d'une relation d'ordre partiel large (cf. chapitre 1, section 1.3.3 et annexe A, section A.2) entre deux formules quelconques d'un langage est cruciale dans le cadre de l'apprentissage symbolique puisqu'elle permet :

- le test de couverture d'un exemple ou d'un contre-exemple du concept à apprendre par une définition candidate ;
- l'organisation de l'espace de recherche, ce qui ouvre de nombreuses possibilités d'exploitation algorithmique (apprentissage par généralisation, par spécialisation, espace des versions, élagage de l'espace ainsi structuré, ...) ;
- la définition d'une relation d'équivalence (puisqu'on peut poser $[x = y] \Leftrightarrow [(x \leq y) \wedge (y \leq x)]$), ce qui rend possible la définition de classes d'équivalences entre hypothèses : on peut alors par exemple choisir parmi différentes formules équivalentes celle qui sera la plus intelligible à l'utilisateur.

La θ-subsomption, définie par Robinson [Robinson65] et introduite dans le cadre de l'apprentissage par Plotkin [Plotkin70] est une relation vérifiant les propriétés ci-dessus. Les résultats énoncés dans cette section sont relatifs au cadre de la PLI : $\mathcal{L}_h$ est le langage de la logique du premier ordre (ou l'un de ses sous-ensembles), et les formules manipulées sont des clauses de Horn.

Définition 3.1 *Relation de θ-subsomption*
On dit qu'une clause D θ-subsume une clause E si et seulement si il existe une substitution θ telle que $D\theta \subseteq E$. On notera $D \vdash_\theta E$.

Exemple : Soient les clauses :

$D :\ C \leftarrow P(X,Y) \wedge P(Y,X).$
$E_1 :\ C \leftarrow P(a,b) \wedge P(b,a).$
$E_2 :\ C \leftarrow P(a,b) \wedge P(b,c).$
$E_3 :\ C \leftarrow P(a,a).$

On a :

$D \vdash_\theta E_1 \quad (\text{avec } \theta = \{X/a, Y/b\})$
$D \nvdash_\theta E_2$
$D \vdash_\theta E_3 \quad (\text{avec } \theta = \{X/a, Y/a\})$

Les raisons pour lesquelles cette relation (qui formalise l'intuition de la notion d'appariement décrite dans le paragraphe précédant) est tellement utilisée en PLI sont liées à ses nombreuses propriétés :

Propriété 3.1
La θ-subsomption est une relation correcte pour l'implication logique, i.e. $(D \vdash_\theta E) \Rightarrow (D \models E)$.

La preuve de cette propriété est immédiate de par la définition de la θ-subsomption puisque : $[\exists\theta \ / \ (D\theta \subseteq E)] \Rightarrow (D \models E)$. Bien qu'en toute généralité l'implication logique puisse être considérée comme le test d'appariement idéal (ce qui revient à dire que D s'apparie (ou subsume) E ssi $D \models E$), cette dernière a l'inconvénient majeur du point de vue pratique d'être indécidable.

Le principal intérêt de la θ-subsomption vient du fait que, contrairement à l'implication logique, cette relation est décidable.

Propriété 3.2
La θ-subsomption est une relation décidable sur l'ensemble des clauses.

Là encore, la preuve est immédiate puisqu'il suffit de tester la compatibilité d'un nombre fini de compositions de substitutions partielles. Dans le pire des cas (chaque littéral de D est compatible avec tous les littéraux de E), il existe $|E|^{|D|}$ manières d'associer à chaque littéral de D un littéral de E.

Puisque d'une part l'implication logique est indécidable sur l'ensemble des clauses et que d'autre part la θ-subsomption est décidable et correcte, il est impossible que la θ-subsomption soit complète pour l'implication logique : $(D \vdash_\theta E) \not\Leftarrow (D \models E)$, ce qu'illustre le contre-exemple suivant :

Contre-exemple :

Soient les deux clauses $D : pair(s(s(X)) \leftarrow pair(X)$ et $E : pair(s(s(s(s(X))))) \leftarrow pair(X)$. On a bien : $D \models E$, mais pas $D \vdash_\theta E$.

[Gottlob87] a caractérisé les deux cas d'incomplétude de la θ-subsomption, ce qui permet de formuler la propriété suivante :

Propriété 3.3
Soient D et E deux clauses. Si D n'est pas auto-référente et si E n'est pas une tautologie, alors la θ-subsomption est une relation correcte, i.e. $(D \vdash_\theta E) \Leftarrow (D \models E)$.

Ainsi, dans le cadre de l'apprentissage de définitions D non récursives grâce à une base d'apprentissage dont les exemples E ne sont pas des tautologies, la θ-subsomption peut être identifiée à l'implication logique.

Du point de vue opératoire, le premier résultat montrant que le calcul de $\vdash_\theta$ est NP-difficile remonte à [Baxter77]. Plus récemment, [Kietz et al.94] ont étendu ce résultat au cas où E est de taille fixe :

Propriété 3.4
Soient D et E deux clauses. Alors, le calcul de $D \vdash_\theta E$ est NP-difficile, même lorsque E est de taille fixe.

3.1.2 Représentation des hypothèses et des instances dans $\mathcal{SIAO}1$

Le langage de représentation des hypothèses de $\mathcal{SIAO}1$ (noté $\mathcal{L}_h$ par la suite) est un langage relationnel typé sans symboles de fonction qui intègre certaines caractéristiques du calcul des prédicats annotés (APC, [Michalski84]), du langage d'ordre 0 de SIA [Venturini94b], et du contexte de l'apprentissage à partir d'interprétations [De raedt97, Blockeel et al.99].

Définition de $\mathcal{L}_h$

Soient :

- Un ensemble fini $\mathcal{C}$ de symboles de classes (chaque symbole correspond à un prédicat d'arité nulle).
- Un ensemble fini $\mathcal{T}_n$ de types de constantes numériques.
- Un ensemble fini $\mathcal{T}_s$ de types de constantes symboliques ($\mathcal{T}_n \cap \mathcal{T}_s = \emptyset$).
- À chaque type de constante numérique (resp. symbolique) correspond un ensemble fini de constantes numériques (resp. symboliques). Chacun de ces ensembles comporte toujours les symboles ? (valeur manquante typée) et $*$ (valeur indifférente typée).
- Un ensemble fini de variables typées.
- Un ensemble fini $\mathcal{P}$ de prédicats typés ($\mathcal{C} \cap \mathcal{P} = \emptyset$). À chaque symbole de prédicat est associé sa signature constituée :
  - d'un entier positif a appelé arité, représentant le nombre d'arguments ;
  - de la liste des k types des arguments du prédicat.

Alors, toutes les formules de $\mathcal{L}_h$ sont de la forme :

$$cl \leftarrow a_1 \wedge a_2 \wedge ... \wedge a_n$$

où cl est un symbole de classe, et les a_i, appelés atomes, sont de la forme :

$$p(t_1, ..., t_k)$$

$p \in \mathcal{P}$ est un symbole de prédicat d'arité k, dont les arguments t_j (qui doivent respecter les contraintes de type relatives aux signatures des prédicats) peuvent être :

- une constante c ;
- une disjonction de constantes, appelée disjonction interne et notée $\{c_1 \vee ... \vee c_j\}$;
- dans le cas des constantes numériques, un intervalle noté $[c_1, c_2]$ (avec $c_1 \leq c_2$) ;
- une variable.

Définition de $\mathcal{L}_e$

$\mathcal{L}_e$ est un sous-ensemble strict de $\mathcal{L}_h$ destiné à représenter les exemples et contre-exemples du concept-cible qui sont par hypothèse des objets spécifiques (c'est-à-dire totalement instanciés). Pour cette raison, une formule de $\mathcal{L}_e$ ne comporte :

- aucune variable ;
- aucune disjonction interne ;
- aucun intervalle numérique.

Caractéristiques de $\mathcal{L}_h$

Le langage de représentation des hypothèses de $\mathcal{SIAO}1$ a, au regard des autres systèmes de PLI, les caractéristiques suivantes :

- $\mathcal{L}_h$ est un langage relationnel clausal :
  - dont toutes les variables sont existentielles ;
  - sans symboles de fonction ;
  - non récursif.

  Ces restrictions, qui garantissent la décidabilité du test d'implication ainsi que la localité des instances[2], définissent un cadre très proche de l'apprentissage par interprétations [De raedt97, Blockeel et al.99]. Ce cadre est particulièrement approprié à l'apprentissage de concepts à partir d'une base de données relationnelle. Dans ce cas, une relation de la base de données est typiquement associée à un prédicat, chaque n-uplet correspondant à un atome instancié. Cependant, alors que la base de données est organisée en tables (par relation), la base d'apprentissage est organisée par projection des relations sur les instances (on parle alors de saturation). La figure 3.1 montre la représentation de molécules à l'aide d'une base de données relationnelle, ainsi que la représentation dans le langage $\mathcal{L}_h$ correspondante.
- $\mathcal{L}_h$ permet d'étendre les domaines des arguments des prédicats d'une manière similaire à ce que propose Michalski dans le cadre du calcul des prédicats annotés (APC, [Michalski84]). Les extensions de domaine correspondent :
  - Pour les constantes symboliques, à une compression d'information puisque la clause $concept \leftarrow P(X, \{c_1 \vee ... \vee c_n\})$ est équivalente à l'ensemble de clauses $\{concept \leftarrow P(X, c_1), ..., concept \leftarrow P(X, c_n)\}$. La présence de d disjonctions internes comportant chacune n constantes correspond donc à n^d clauses.

[2]Chaque instance contient toute l'information descriptive la concernant. Les instances sont donc indépendantes les unes des autres.

Molécules	
Id_m	Classe
$m1$	$classe1$
$m2$	$classe2$
...	

Atomes		
Id_m	Id_a	Élément
$m1$	$a1$	o
$m1$	$a2$	s
$m1$	$a3$	o
$m2$	$a4$	c

Liaisons			
Id_m	Id_a	Id_a	Liaison
$m1$	$a1$	$a2$	$dative\_2e$
$m1$	$a2$	$a3$	$covalente\_4e$
...			

$$classe1 \leftarrow atome(a1, o) \wedge atome(a2, s) \wedge atome(a3, o) \wedge liaison(a1, a2, dative\_2e) \wedge liaison(a2, a3, covalente\_4e).$$
$$classe2 \leftarrow atome(a4, c) \wedge ...$$

FIG. 3.1: Représentations de données relationnelles.

- Pour les constantes numériques, outre la compression relative aux disjonctions internes, les intervalles impliquent la connaissance implicite de la relation d'ordre $\leq$ définie sur $\Re$.

Ces extensions permettent de faire bénéficier $\mathcal{SIAO}1$ des capacités de généralisation et de l'intelligibilité des systèmes opérant sur les représentations attribut-valeur. $\mathcal{SIAO}1$ peut donc être utilisé à la fois dans le cadre de problèmes relationnels relevant de la PLI et dans le cadre de problèmes plus classiques relevant de systèmes d'ordre 0.

– Il est possible de spécifier une théorie du domaine sous la forme de hiérarchies implantant la relation est-un sur des constantes de même type ou des prédicats de même signature. Par définition, le symbole (typé) * est au sommet de la hiérarchie de chaque type. Pour ce qui est des prédicats, le sommet de chaque hiérarchie est le prédicat vide (noté $\emptyset$), qui représente d'un point de vue logique l'abandon de l'atome correspondant. Un exemple de hiérarchie sur des constantes est donné par la figure 1.4 du chapitre 1.

– Pour finir, $\mathcal{L}_h$ bénéficie des symboles représentant les valeurs manquantes (?) ou indifférentes (*) dont l'utilité a été démontrée dans le cadre du système attribut-valeur SIA ([Venturini94b], voir également chapitre 2, section 2.2.3). Cependant, contrairement aux représentations attribut-valeur, les représentations relationnelles expriment la caractéristique non définie d'une valeur par l'abandon du prédicat associé, ce qui rend superflu l'existence d'un symbole spécifique (noté # dans le cas de SIA). Par exemple, considérons la représentation attribut-valeur suivante :

classe	identifiant	nb_roues	couleur	contenance_remorque	marque
voiture	$283LZZ75$	4	*	#	?

L'expert a observé une voiture immatriculée $283LZZ75$. Il sait que la couleur n'est pas une caractéristique discriminante pour la tâche de classification en cours : plutôt que de rechercher cette information, il la déclare donc indifférente (symbole *). D'autre part, puisqu'aucune remorque n'est attachée au véhicule, l'attribut contenance_remorque n'a aucune signification dans ce contexte, ce que dénote le symbole # associé aux valeurs non définies. Finalement, l'expert n'ayant pas con-

naissance de la marque de la voiture, l'attribut correspondant est noté manquant (symbole `?`). Le codage relationnel associé de l'objet décrit par cette table est donc :

$$voiture \leftarrow objet(283LZZ75) \wedge nb\_roues(283LZZ75, 4) \wedge couleur(283LZZ75, *) \wedge marque(283LZZ75, ?).$$

Typage et généralisation – types fonctionnels et relationnels

Le typage est souvent perçu comme une technique dont l'utilité se résume au contrôle de la cohérence d'expressions d'un langage. Cependant, dans le cas de $\mathcal{SIAO}1$, ce dernier est d'autant plus utile qu'il correspond à un biais de langage naturel (la notion de type est très intuitive) et souple (l'utilisateur segmente autant qu'il le désire les ensembles d'attributs). Dans ce cas, le typage ne sert pas uniquement au contrôle des données fournies au système d'apprentissage, puisqu'il permet de réduire la taille de l'espace de recherche. En effet, définir le type d'un objet consiste (entre autres) à lui associer un ensemble de valeurs potentielles. Par exemple, en supposant l'existence :

- d'un type `forme` à valeur dans $\{cercle, rectangle, triangle\}$;
- d'un type `couleur` à valeur dans $\{blanc, bleu, noir, rouge, vert\}$;
- d'un prédicat $Objet$ d'arité 3 et dont les arguments sont de type `obj_id`, `forme` et `couleur` ;

il est possible de restreindre les généralisations de l'atome $Objet(X, rectangle, rouge)$ aux formules vérifiant les règles de typage, ce qui permet d'élaguer des atomes tels que $Objet(X, \{rectangle \vee noir\}, rouge)$.

De plus, $\mathcal{SIAO}1$ enrichit le typage en spécifiant deux modalités (relationnelle et fonctionnelle) sur les types :

- Un type est dit relationnel si toutes les généralisations utiles des symboles de ce type peuvent être obtenues par variabilisation.
- Un type est dit fonctionnel si toutes les généralisations utiles des symboles de ce type peuvent être obtenues par extension de domaine.

En pratique, les données n'ayant qu'un aspect soit structurel, soit ensembliste, il est toujours possible de déterminer avec certitude le biais de langage correspondant. En particulier, lorsque les exemples proviennent d'une base de données relationnelle, les attributs structurants correspondent aux clés des tables (ce sont des constantes relationnelles), les autres valeurs étant des constantes fonctionnelles. Par exemple, dans le cas de la figure 3.1, les types `Id_m` et `Id_a` sont relationnels alors que les types `Élément` et `Liaison` sont fonctionnels.

3.1.3 La relation de subsomption sur $\mathcal{L}_h$

Le test de subsomption de $\mathcal{SIAO}1$, noté $\vdash$, est similaire au test de θ-subsomption puisqu'il est fondé sur la mise en correspondance de chaque littéral de l'hypothèse avec un littéral de l'exemple, sous réserve du respect des contraintes d'égalité imposées par les variables. Cependant, les particularités de $\mathcal{L}_e$ et $\mathcal{L}_h$ induisent certaines différences qui sont énumérées ci-après et examinées du point de vue de leur impact sur la complexité du test $\vdash$.

Une première différence provient de l'existence d'une théorie du domaine sous la forme de hiérarchies définies sur les constantes de même type et les prédicats de même signature. Par bonheur, la complexité du test de subsomption reste identique à celle du test de θ-subsomption puisque d'une part les règles d'instanciation des variables restent inchangées, et que d'autre part chaque appariement entre un littéral de l'hypothèse et un littéral de l'exemple génère au plus une substitution candidate. Plus formellement, en faisant les hypothèses très raisonnables :

- d'un coût borné par une constante k_1 pour la complexité du test de subsomption de deux symboles de constante ou de prédicat ;
- d'une borne k_2 à l'arité des prédicats ;

alors la complexité du test de subsomption entre l'hypothèse D et l'exemple E augmente (grossièrement[3]) dans le pire des cas de $k_1.k_2.|E|.|D|$, ce qui ne modifie pas la grandeur asymptotique de la complexité du test de subsomption, puisque (de par son caractère NP-difficile) cette dernière domine asymptotiquement toute fonction polynomiale de $|D|$ et de $|E|$.

La seconde différence a trait aux extensions de domaine (disjonctions internes et intervalles numériques). Là encore, la complexité globale du test de subsomption n'est pas modifiée. En effet, en remarquant que toute extension de domaine provient d'un type fonctionnel et ne s'apparie par conséquent qu'avec une constante ou une autre extension de domaine du même type, il découle :

- que chaque appariement entre un littéral de l'hypothèse et un littéral de l'exemple génère au plus une substitution candidate (aucun symbole de variable n'est mis en correspondance avec une extension de domaine) ;
- qu'en supposant bornée la taille des extensions de domaine, le test de compatibilité (qui revient à vérifier l'inclusion d'un ensemble dans un autre) est de complexité bornée par une constante k.

[3] C'est par exemple le cas pour l'algorithme proposé sections 3.2.2 et suivantes, puisqu'alors seule la complexité du pré-traitement correpondant au coût de création du tableau des contraintes est altérée. La complexité de la recherche n'est par contre pas affectée, le nombre de contraintes du tableau étant toujours borné par $|E|.|D|$.

En suivant le même raisonnement que précédemment, il découle donc que la grandeur asymptotique de la complexité du test de subsomption n'est pas modifiée.

La dernière différence entre $\vdash$ et $\vdash_\theta$ provient de la sémantique particulière associée aux symboles `*` et `?`, qui nécessite la spécification de règles d'appariement adaptées. En effet :

- Par définition, une valeur manquante est définie mais inconnue. Le symbole `?` doit donc être considéré comme une constante. Cependant, chaque occurrence de `?` correspond à une valeur inconnue potentiellement différente d'une autre occurrence du même symbole. La règle d'appariement de `?` se ramène ainsi à un simple mécanisme de réécriture :

  Chaque occurrence de `?` dans une formule de $\mathcal{L}_h$ est équivalente pour la relation de subsomption à l'introduction d'un nouveau symbole de constante.

  Exemples :

D	D *réécrite*	E	E *réécrite*	$D \overset{?}{\vdash} E$
$P(*,*)$	$P(*,*)$	$P(?,?)$	$P(c_1,c_2)$	*oui*
$P(a,A)$	$P(a,A)$	$P(a,?)$	$P(a,c_1)$	*oui*
$P(A,A)$	$P(A,A)$	$P(?,?)$	$P(c_1,c_2)$	*non*
$R(a)$	$R(a)$	$R(?)$	$R(c_1)$	*non*
$R(?)$	$R(c_1)$	$R(?)$	$R(c_2)$	*non*

  Un cas particulier à cette règle survient lorsque le test de subsomption est utilisé afin de simplifier une formule (comme dans le cas de l'algorithme `FASTCONDENSE` décrit section 3.2.1). Il est alors nécessaire de procéder à la réécriture de la formule avant duplication. Dans ce cas, la dernière ligne du tableau devient :

D	D *réécrite*	$E \subseteq D$	E *réécrite*	$D \overset{?}{\vdash} E$
$R(?)$	$R(c_1)$	$R(?)$	$R(c_1)$	*oui*

  Ce mécanisme permet également de garantir la réflexivité de la relation de subsomption.

- Une valeur est dite indifférente lorsque la constante à laquelle elle se rapporte doit être définie, sa valeur n'ayant par contre aucune importance pour l'exemple et la tâche de classification considérés. Soient par exemple les formules suivantes :

  - $(D_1) : oiseau \leftarrow animal(X) \wedge ailes(X,2) \wedge plumes(X,*)$ dont la signification est un animal ayant deux ailes et des plumes (dont la couleur est indifférente) est un oiseau .
    $(E_1) : oiseau \leftarrow animal(titi) \wedge ailes(titi,2) \wedge plumes(titi,jaunes)$, qui signifie l'oiseau titi est un animal ayant deux ailes et des plumes jaunes .
    Il vient naturellement : $D_1 \vdash E_1$.

– (D_2) : $oiseau \leftarrow animal(X) \wedge ailes(X,2) \wedge plumes(X, jaunes)$ signifie un animal ayant deux ailes et des plumes jaunes est un oiseau .
(E_2) : $oiseau \leftarrow animal(titi) \wedge ailes(titi,2) \wedge plumes(titi,*)$ signifie l'oiseau titi est un animal ayant deux ailes et des plumes. L'expert n'a pas pris la peine de mentionner la couleur des plumes car il sait que cette donnée est, dans le cas de *titi*, inutile à la tâche de classification.
Il vient également : $D_2 \vdash E_2$.

La règle d'appariement pour $*$ s'énonce alors :

> Toute occurrence du symbole $*$ dans une formule de $\mathcal{L}_h$ couvre et est couverte par tout symbole de même type.

Exemples :

D	E	$D \overset{?}{\vdash} E$
$P(a,*)$	$P(a,b)$	*oui*
$P(*,*)$	$P(a,b)$	*oui*
$P(a,A)$	$P(a,*)$	*oui*
$P(A,A)$	$P(*,*)$	*oui*
$R(a)$	$R(*)$	*oui*

En contrepartie de son intérêt pratique évident (possibilité de spécifier des connaissances portant sur la tâche de classification, facilitation de l'acquisition des données, etc.), la sémantique choisie pour le symbole $*$ a des répercussions sur le cadre théorique du couple $(\vdash, \mathcal{L}_h)$. Ces dernières obligent un raisonnement à deux niveaux afin de conserver la richesse et le bien fondé sémantique du langage d'une part, et une structure adaptée à la méthode de recherche d'autre part.

– *Au niveau de la sémantique de* $\mathcal{L}_h$, où opère le test de subsomption, les propriétés du symbole $*$ sont bien fondées. La validation des hypothèses par calcul de leur couverture et de leur cohérence se fait donc au moyen du test $\vdash$.

– *Au niveau de la structure de* $\mathcal{L}_h$, où opère la méthode de recherche, les propriétés du symbole $*$ ne sont pas bien fondées car la relation $\vdash$ n'est plus une relation d'ordre. Par exemple, les formules $F_1 : P(X,a)$ et $F_2 : P(*,*)$ vérifient $F_1 \vdash F_2$ et $F_2 \vdash F_1$. La relation $\vdash$ n'est donc plus antisymétrique, et un algorithme de recherche fondé uniquement sur cette relation peut être confronté à des problèmes de cycles dans le graphe d'exploration.

  Une solution consiste alors à considérer une relation $\vdash_2$ plus stricte que $\vdash$ (c'est-à-dire telle que $\forall F_1, F_2 \in \mathcal{L}_h, (F_1 \vdash_2 F_2) \Rightarrow (F_1 \vdash F_2)$), qui ait la propriété d'être une relation d'ordre. Cette relation est trivialement obtenue en posant la règle de réécriture suivante pour $*$:

  > Toute occurrence de $*$ dans une formule de $\mathcal{L}_h$ est équivalente pour la relation de subsomption $\vdash_2$ à l'introduction d'une nouvelle variable.

En développant des opérateurs unidirectionnels respectant la relation $\vdash_2$, l'algorithme de recherche est bien fondé (voir figure 3.2).

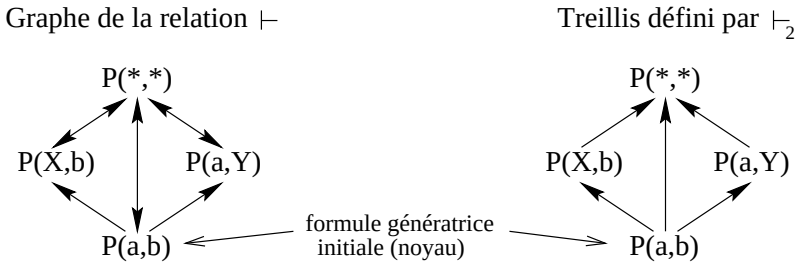


FIG. 3.2: La relation $\vdash_2$ génère un treillis sur le graphe défini par la relation $\vdash$, permettant ainsi une exploration efficace.

Récapitulatif

La première partie de ce chapitre a été l'occasion de décrire le langage de représentation des hypothèses et des instances de $\mathcal{SIAO}1$, que ce soit du point de vue de son expressivité ou de la complexité de la relation de subsomption. Puisque $\mathcal{L}_h$ ne permet pas de générer de définition auto-référente et que, par hypothèse, aucun exemple de la base d'apprentissage n'est une tautologie, le test de subsomption proposé est à la fois correct, complet, et décidable pour l'implication logique. Son principal inconvénient, malheureusement inhérent à la nature relationnelle du langage, est sa NP-complétude. La seconde partie de ce chapitre est donc consacrée à l'implantation d'un test optimisé pouvant opérer en un temps raisonnable sur des données complexes.

3.2 Implantation du test de subsomption

3.2.1 Raisons motivant l'optimisation

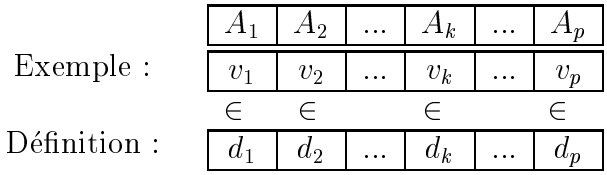
Importance et complexité du test de subsomption

Le test de subsomption est une composante cruciale de $\mathcal{SIAO}1$ puisqu'il est entre autres utilisé lors de toute évaluation d'un nouvel individu. Cependant, ce test est également un point critique de l'algorithme pour au moins deux raisons.

En premier lieu, *le test de subsomption doit être fait très souvent* : pour chaque évaluation d'un individu, le calcul de la couverture de ce dernier nécessite autant de tests qu'il y a d'exemples dans la base d'apprentissage. Ce facteur limitant, qui est commun aux algorithmes de type générer et évaluer procédant par évaluation de la couverture (parmi lesquels la plupart des algorithmes basés sur la méthodologie de l'étoile), peut être atténué

par l'utilisation de stratégies spécifiques (comme par exemple l'échantillonnage de la base d'apprentissage [Cochran77, Brezellec99], différentes techniques de court-circuit ([Augier et al.96] et chapitre 4 section 4.2.3)), ou bien encore traité par des méthodes de parallélisation (chapitre 5). Bien que ce chapitre ne soit pas consacré à ce premier facteur limitant, il est cependant important de remarquer qu'il n'est pas rare qu'en pratique plusieurs millions de tests de subsomption soient nécessaires pour un apprentissage.

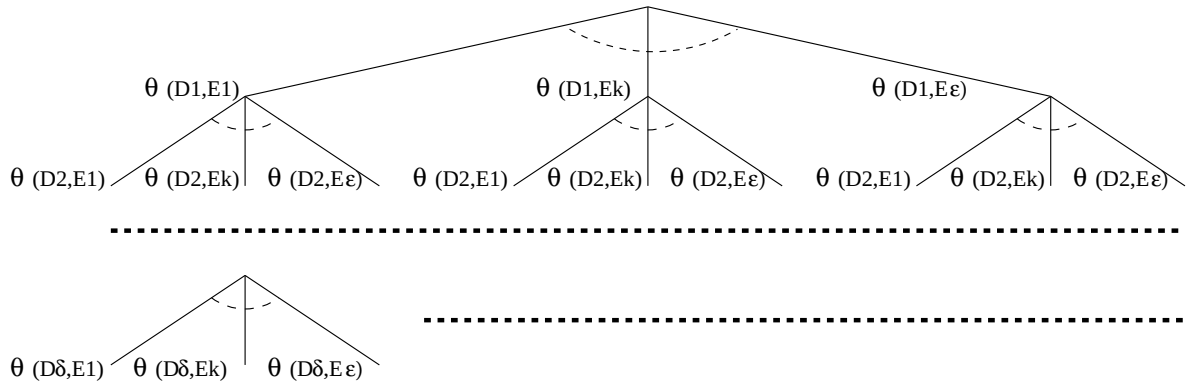
D'autre part, *le test de subsomption peut être extrêmement complexe*, ce facteur dépendant du langage utilisé. Dans le cas de la logique d'ordre 0, une définition et un exemple comportent le même nombre fixe p d'attributs A_i, et le test de subsomption consiste alors en une succession d'au plus p tests d'appartenance entre chaque valeur v_i de l'exemple et le domaine d_i correspondant de la définition (voir figure 3.3).



p tests d'appartenance de type $v_i \in d_i$ sont nécessaires dans le pire des cas.

FIG. 3.3: Complexité du test de subsomption en logique d'ordre 0.

La complexité du test de subsomption en logique d'ordre 0 est donc dans le pire des cas linéaire en fonction du nombre d'attributs de la définition (ou de l'exemple). Par contre, dans le cas du système $\mathcal{SIAO}1$, où l'on travaille dans un langage qui est un sous-ensemble des formules de la logique d'ordre un, la complexité du test de subsomption est exponentielle en fonction du nombre de littéraux de la définition. En effet, comme l'ont montré le chapitre 1 (section 1.5.1) et la section présentant le contexte théorique du test de θ-subsomption, en supposant que la définition D (resp. l'exemple E) soit sous la forme d'une conjonction $D_1 \wedge ... \wedge D_\delta$ (resp. $E_1 \wedge ... \wedge E_\varepsilon$) de δ (resp. ε) atomes ne comportant aucun symbole de fonction, le test de subsomption consiste à trouver pour chaque atome de D un atome de E compatible. De plus, comme les relations entre atomes occasionnées par les variables rendent ces choix interdépendants (les substitutions $\theta(D_i, E_j)$ doivent être compatibles entre elles), ε^δ essais sont nécessaires dans le cas d'un test de subsomption énumérant les différentes combinaisons possibles[4] (voir figure 3.4).



ε^δ tests de composition de substitutions sont nécessaires dans le pire des cas.

FIG. 3.4: Complexité du test de subsomption dans le cas de $\mathcal{SIAO}1$.

Cette section présente le test de θ-subsomption tel qu'il est mis en œuvre dans de nombreux systèmes de PLI et motive le développement d'un test de subsomption optimisé adapté au langage de $\mathcal{SIAO}1$.

[4]L'implantation habituelle de ce test, détaillée ci-après, est fondée sur une définition récursive avec possibilité de retour-arrière précoce. Bien que cette définition permette de réduire le nombre de tests de compatibilité dans le cas de certaines instances, la complexité dans le pire des cas est toujours ε^δ.

Le test de θ-subsomption canonique

Le test de θ-subsomption canonique de deux clauses $D : D_{tete} \leftarrow D_1 \wedge ... \wedge D_n$ et $E : E_{tete} \leftarrow E_1 \wedge ... \wedge E_p$ (dont on peut trouver une définition simple en PROLOG dans [Kietz et al.94]) construit récursivement une substitution solution à l'aide d'une fonction booléenne `apparie_atomes(`a_1`,`a_2`,`θ`)` rendant vrai si les atomes a_1 et a_2 s'apparient suivant une substitution θ.

```
/* Test de θ-subsomption canonique */

Fonction θ-subsomption(D,E)
Si apparie_atomes(D_tete,E_tete,θ)
Alors → apparie_corps({D_1 ∧ ... ∧ D_n}, {E_1 ∧ ... ∧ E_p}, {E_1 ∧ ... ∧ E_p}, θ)
Sinon → échec

Fonction apparie_corps(D_restant, E_restant, E_corps, θ)
Si D_restant = ∅ Alors → succès
Si E_restant = ∅ Alors → échec
Si (apparie_atomes(D_1,E_1,θ')) ∧ (θ,θ' sont compatibles)
     Alors Si apparie_corps({⋀_{i=2}^{n} D_i}, E_corps, E_corps, θ ∪ θ') → succès
     Sinon → apparie_corps(D_restant, {⋀_{j=2}^{p} E_j}, E_corps, θ)
```

Bien qu'il soit possible d'implanter le test $\vdash$ à l'aide d'une architecture similaire à celle du test $\vdash_\theta$ canonique, de nombreuses observations – réalisées dans un contexte expérimental – concernant le comportement de cette procédure motivent l'implantation d'un test optimisé.

Changement de complexité lors de l'aplatissement

Une première interrogation provient du fait que le test de θ-subsomption canonique a un comportement très différent suivant la syntaxe de la formule sur laquelle il s'applique, et ce même à sémantique identique. En considérant par exemple le domaine de la chimie moléculaire, dans lequel on dispose des prédicats *atome(identifiant,symbole)* et *liaison(identifiant,identifiant,type_de_la_liaison)*, on pourra représenter la molécule de dioxyde de soufre (de structure $|\overline{O} \leftarrow \overline{S} = O\rangle$) par la formule suivante :

$$atome(X_1, o) \wedge atome(X_2, s) \wedge atome(X_3, o) \wedge liaison(X_1, X_2, dative\_2e) \wedge liaison(X_2, X_3, covalente\_4e)$$

FIG. 3.5: Description relationnelle classique de la molécule de dioxyde de soufre.

Cependant la formule aplatie correspondante (qui est strictement équivalente) aura un

coût d'appariement (en nombre de retours-arrière) plus important puisque le test de θ-subsomption canonique, qui est totalement guidé par la syntaxe, fera des retours-arrière inutiles sur tous les prédicats ajoutés qui correspondent aux constantes aplaties.

$$atome(X_1, X_2) \wedge \boxed{o(X_2)} \wedge atome(X_3, X_4) \wedge \boxed{s(X_4)} \wedge atome(X_5, X_6) \wedge \boxed{o(X_6)} \wedge$$
$$liaison(X_1, X_3, X_7) \wedge \boxed{dative\_2e(X_7)} \wedge liaison(X_3, X_5, X_8) \wedge \boxed{covalente\_4e(X_8))}$$

(Les atomes encadrés correspondent aux constantes aplaties.)

FIG. 3.6: Description aplatie de la molécule de dioxyde de soufre.

Idéalement, un test de θ-subsomption optimisé devrait, en ne capturant que la complexité intrinsèque associée à l'aspect relationnel de la formule, utiliser une représentation rendant la recherche aussi indépendante que possible de la formulation syntaxique. En particulier, il serait intéressant que dans le cas de formules isomorphes sur le plan relationnel cette représentation rende la recherche indépendante de l'ordre des atomes ou de leur nombre.

Un problème provenant du domaine de la chimie moléculaire

Un second motif pour remettre en question le test de θ-subsomption canonique vient de l'observation de son comportement sur des formules fortement relationnelles. Par exemple la base *Mutagenesis* (voir [Debnath et al.91, Srinivasan et al.94], ainsi que l'annexe B, section B.4) décrit des molécules comportant typiquement entre 20 et 40 atomes : si les algorithmes d'apprentissage travaillant par spécialisation sont alors fortement ralentis, puisqu'en testant des hypothèses comportant 1, 2, 3, ... atomes sur une molécule comportant 40 atomes le coût de l'appariement sera dans le pire des cas de 40^1, 40^2, 40^3, ... ; il en va tout autrement d'un algorithme ascendant qui sera confronté, dès les premières variabilisations à un espace de recherche d'une taille dans le pire des cas de 40^{40}. Dans ce dernier cas, et contrairement aux observations concernant la variation de la complexité lors d'un changement de représentation, le problème est crucial car il ne s'agit pas de mesurer l'accélération ou le ralentissement de l'algorithme mais de constater le fait qu'il fonctionne correctement ou bien qu'au contraire il est inutilisable car il ne termine pas.

Simplification des clauses

Une dernière raison pour optimiser le test de θ-subsomption vient de la nécessité de disposer d'un algorithme de simplification des clauses efficace. Bien que la simplification d'une clause puisse être envisagée de bien des manières différentes[5], nous nous intéressons ici à un problème décidable dont l'algorithme de simplification associé est NP-complet.

Définition 3.2 *Relation d'équivalence fondée sur la θ-subsomption*
Deux clauses C et D sont dites θ-équivalentes (noté $C \sim_\theta D$) si $C \vdash_\theta D$ et $D \vdash_\theta C$.

[5]Par exemple, une clause est dite redondante lorsqu'elle est simplifiable étant donné un ensemble d'autres clauses (les connaissances du domaine). La redondance est cependant un problème indécidable dans le cas général [NC et al.97].

Définition 3.3 *Clause réduite*
Une clause C est dite réduite si il n'existe aucun sous-ensemble $D \subset C$ tel que $C \sim_\theta D$. Une clause réduite D telle que $C \sim_\theta D$ et $D \subseteq C$ est appelée réduction de C.

Exemple : La clause $C = p(x,y) \wedge p(y,x) \wedge p(y,y)$ peut être simplifiée en $D = p(y,y)$ qui est une réduction de C.

Une propriété remarquable des clauses réduites est leur unicité : si D_1 et D_2 sont deux clauses réduites d'une même clause C, alors D_1 et D_2 sont identiques à une substitution de renommage près. Comme le montrent les lignes suivantes, cette propriété est plus généralement valable pour toutes les clauses réduites θ-équivalentes.

Propriété 3.5 *Unicité des clauses réduites*
Soient C et D deux clauses réduites. Si $C \sim_\theta D$, alors C et D sont identiques à une substitution de renommage près.

En effet, si $C \sim_\theta D$, il existe des substitutions θ et σ telles que $C\theta \subseteq D$ et $D\sigma \subseteq C$. De plus, puisque C et D sont réduites, on a : $C\theta\sigma = C$ et $D\sigma\theta = D$: σ et θ sont donc des substitutions de renommage.

[Gottlob et al.93] proposent un algorithme effectuant au plus $|C|$ tests de θ-subsomption ainsi que des arguments concernant son optimalité. Cet algorithme prend en entrée une clause C et retourne sa réduction. Le pseudo-code est le suivant :

```
/* Algorithme FASTCONDENSE */
D ← C
E ← C
Tant que (E ≠ ∅)
     Choisir L ∈ E
     Si ∃L' ∈ D \ {L} : L ⊢θ L'
     Alors Si D \ {L} ⊢θ C
          Alors D ← D \ {L}
     E ← E \ {L}
→ D
```

L'intérêt en PLI d'une procédure efficace de simplification des clauses est double :

- En ce qui concerne l'utilisateur, un algorithme de simplification des clauses peut contribuer à une meilleure intelligibilité du résultat.
- En ce qui concerne le système d'apprentissage, l'algorithme de simplification permet de retirer de l'espace de recherche les doublons. D'autre part, le fait de travailler avec des formules de taille minimale peut réduire le coût de certains calculs.

3.2.2 Architecture générale

L'algorithme est organisé en trois parties. La première traite tout d'abord le cas trivial de l'appariement de deux atomes quelconques de D et E, ce qui permet d'une part de traiter immédiatement les cas de succès ou d'échec dégénérés et d'autre part de construire une représentation adaptée de l'espace de recherche. L'approche suivie consiste à travailler directement sur les substitutions partielles (entre littéraux), ce qui permet une reformulation non récursive du problème décrivant explicitement dans un tableau les $|E|.|D|$ contraintes associées. La composante complexe du problème (détermination de θ) ayant été isolée, la seconde partie de l'algorithme met à profit des propriétés relatives à l'indépendance, la combinaison ou l'incompatibilité des contraintes afin de réduire l'espace de recherche. La dernière partie de l'algorithme est dédiée à la recherche de la solution dans l'espace simplifié à l'aide d'une technique classique propageant les contraintes au cours de la construction de la solution.

3.2.3 Traitement de la partie fixe

Soit $|D|$ (resp. $|E|$) le nombre d'atomes de D (resp. E). L'ensemble $Vars(D) = \{X_1, ..., X_n\}$ des variables typées apparaissant dans la définition est connu. Il est également fait référence à un algorithme d'appariement de deux atomes $\delta \in D$ et $\varepsilon \in E$, noté $apparie\_atomes(\delta, \varepsilon)$, qui rend $vrai$ ou $faux$ ainsi que la substitution $\theta_{\delta,\varepsilon}$ nécessaire à la réalisation de l'appariement dans le cas où ce dernier est possible.

Exemple :

- Types numériques : $t_1^n,\ t_2^n$.
- Types symboliques : $t_1^s = \{a, b, c\},\ t_2^s = \{g, h, i\}$.
- Hiérarchie sur t_1^s : $a \rightarrow b$.
- Symboles de prédicat : $P(t_1^n, t_2^n, t_1^s, t_2^s),\ Q(t_1^n, t_2^n, t_1^s, t_2^s)$.
- Théorie : $P \rightarrow Q$.

On a : $apparie\_atomes(Q(1, [2, 4], \{b \vee c\}, X_2), P(*, 3, a, g)) = vrai \ \ et \ \ \theta = \{X_2/g\}$

Remarques :

- Le symbole $*$ s'apparie avec tout autre symbole du même type et ne pose donc aucune contrainte sur θ.
- On pourra noter par abus $\theta = \{X_2/g\} = \{X_1/*, X_2/g, \} = [*, g]$ afin de représenter les contraintes de manière plus concise.

Outre le traitement des cas triviaux (`succès final` et `échec final`), cette partie de l'algorithme génère les ensembles $Vals(X_i)$ des valeurs envisageables pour X_i ainsi que

$\mathcal{L}$, un tableau de listes de substitutions partielles qui permet de mémoriser pour chaque atome non fermé de D la liste des contraintes élémentaires associées.

/* Test de subsomption optimisé pour $\mathcal{SIAO}1$, partie 1/5 */

/* Traitement de la partie fixe. */

parcourir D ; pour chaque atome $\delta \in D$:

Si δ est fermé :

Si $\exists \varepsilon \in E$ / $apparie\_atomes(\delta, \varepsilon)$

Alors $\rightarrow$ sortir et passer à un autre atome δ

Sinon $\rightarrow$ échec final

Sinon : parcourir tous les $\varepsilon \in E$:

Si $apparie\_atomes(\delta, \varepsilon)$ Alors ajouter la substitution $\theta_{(\delta,\varepsilon)}$ à $\mathcal{L}(\delta)$ et mettre à jour $Vals(X_j)$ pour les variables X_j instanciées dans $\theta_{\delta,\varepsilon}$

Si $\nexists \varepsilon \in E$ / $apparie\_atomes(\delta, \varepsilon)$, Alors $\rightarrow$ échec final

Si ($\mathcal{L}$ a au plus une ligne) $\vee$ ($\forall X_i, |Vals(X_i)| = 1$) Alors $\rightarrow$ succès final

Exemple :

- Types numériques : $aucun$, Types symboliques : $t_1^s = \{a, b, c, d, e\}$.
- Symboles de prédicat : $objet(t_1^s)$, $lien(t_1^s, t_1^s)$
- Hiérarchie sur t_1^s : $aucune$, Théorie : $aucune$.
- D : $objet(X) \wedge objet(Y) \wedge objet(Z) \wedge lien(X, Y) \wedge lien(Y, Z) \wedge lien(Z, X)$.
- E : $objet(a) \wedge objet(b) \wedge objet(c) \wedge objet(d) \wedge objet(e) \wedge lien(a, b) \wedge lien(b, c) \wedge lien(c, d) \wedge lien(d, b)$.

On a :

$$\mathcal{L} = \begin{bmatrix} objet_1 & : & [a,*,*] & [b,*,*] & [c,*,*] & [d,*,*] & [e,*,*] \\ objet_2 & : & [*,a,*] & [*,b,*] & [*,c,*] & [*,d,*] & [*,e,*] \\ objet_3 & : & [*,*,a] & [*,*,b] & [*,*,c] & [*,*,d] & [*,*,e] \\ lien_1 & : & [a,b,*] & [b,c,*] & [c,d,*] & [d,b,*] & \\ lien_2 & : & [*,a,b] & [*,b,c] & [*,c,d] & [*,d,b] & \\ lien_3 & : & [b,*,a] & [c,*,b] & [d,*,c] & [b,*,d] & \end{bmatrix}$$

3.2.4 Simplification de la partie contrainte

Outre la résolution des succès et échecs triviaux, la première partie de l'algorithme a permis le passage d'une formulation classique du problème à une représentation adaptée (un tableau de contraintes sur les substitutions), puisque :

- On obtient une vue générale du problème rendant possible d'une part un travail global sur les contraintes en vue d'élaguer autant que possible l'espace de recherche et d'autre part un traitement optimisé du problème puisqu'il n'est plus défini de manière récursive.
- Tous les tests de résultat constant, c'est-à-dire liés à la composante syntaxique fixe du problème (recherche des atomes, tests de couverture des prédicats, tests sur les types de arguments, tests de couverture des arguments constants, ...) ne figurent plus dans l'énoncé : ils ont déjà été réalisés lors du traitement de la partie fixe du problème.

Le tableau obtenu $\mathcal{L}$ comporte un nombre n de lignes égal au nombre d'atomes δ_i non fermés contenus dans D. Chaque ligne i de $\mathcal{L}$ est constituée d'une ou plusieurs contraintes $\theta_{i,j}$ correspondant aux conditions nécessaires à l'appariement de δ_i avec un atome de E.

$$\mathcal{L} = \left[\begin{array}{cccccc} \delta_1 & : & \theta_{1,1} & \theta_{1,2} & ... & \theta_{1,p_1} \\ \delta_2 & : & \theta_{2,1} & \theta_{2,2} & ... & \theta_{2,p_2} \\ & & ... & & & \\ \delta_n & : & \theta_{n,1} & \theta_{n,2} & ... & \theta_{n,p_n} \end{array}\right]$$

FIG. 3.7: Représentation tabulaire du test de subsomption.

Comme on l'a vu dans le paragraphe précédant détaillant le fonctionnement de la méthode *apparie_atomes*, une fois l'ensemble $Vars(D)$ ordonné, toute contrainte d'unification $\theta_{i,j}$ peut être représentée par une liste de taille fixe égale à $|Vars(D)|$ dont chaque élément, noté $\theta_{i,j}(k)$, est soit un symbole de constante c, soit une étoile $*$. Soit $\mathcal{E}$ l'ensemble de toutes les contraintes θ ; il est possible de définir formellement les propriétés suivantes :

Définition 3.4 *Compatibilité*
On dit que deux contraintes θ_1 et θ_2 sont compatibles, *et on notera $\theta_1 \ -_c \ \theta_2$, lorsque : $\forall k \in [1..|Vars(D)|]$, soit $\theta_1(k) = *$ ou encore $\theta_2(k) = *$, soit $\theta_1(k) = \theta_2(k)$. Dans le cas contraire on dira que θ_1 et θ_2 ne sont pas compatibles, ce qui sera noté $\theta_1 \neg_c \theta_2$.*

Définition 3.5 *Généralité*
On dit que la contrainte θ_1 est plus générale ou équivalente *à la contrainte θ_2, et on notera $\theta_1 \ \geq_c \ \theta_2$ si et seulement si $\forall k \in [1..|Vars(D)|]$, soit $\theta_1(k) = *$, soit $\theta_1(k) = \theta_2(k)$.*

Propriété 3.6 $\geq_c$
La relation $\geq_c$ est une relation d'ordre partiel large bien fondée sur $\mathcal{E}$.

En effet d'une part $\geq_c$ est réflexive, antisymétrique et transitive, et d'autre part il ne peut exister de suite infinie strictement décroissante d'éléments de $\mathcal{E}$.

Propriété 3.7 $\sqcup$
Deux contraintes θ_1 et θ_2 ont toujours une borne supérieure θ notée $\theta_1 \sqcup \theta_2$ définie par :

- *si* $\theta_1(k) = \theta_2(k)$, *alors* $\theta(k) = \theta_1(k)$
- *sinon* $\theta(k) = *$.

Propriété 3.8 *Structure de* $\mathcal{E}$
L'espace $\mathcal{E}$ *des contraintes est un sup-demi treillis. Le plus grand élément de* $\mathcal{E}$ *est* $[*, *, ..., *]$.

$\mathcal{E}$ est un sup-demi treillis puisque toute paire d'éléments de $\mathcal{E}$ admet une borne supérieure. Par contre, $\mathcal{E}$ n'est pas un treillis car deux contraintes quelconques ne possèdent pas obligatoirement de borne inférieure.

Définition 3.6 *Combinaison*
On dit que la contrainte θ_3 *est la* combinaison *des deux contraintes* θ_1 *et* θ_2, *et on notera* $\theta_1 +_c \theta_2 = \theta_3$, *si et seulement si :*

- θ_1 *et* θ_2 *sont compatibles ;*
- $\forall k \in [1..|Vars(D)|]$, *si* $\theta_1(k) = *$ *alors* $\theta_3(k) = \theta_2(k)$, *sinon* $\theta_3(k) = \theta_1(k)$.

Exemples : Soient $\theta_1 = [a, b, *, *], \theta_2 = [a, *, c, *], \theta_3 = [a, b, c, *]$ *et* $\theta_4 = [d, *, *, *]$

On a (entre autres) : $\theta_1 -_c \theta_2$; $\theta_1 +_c \theta_2 = \theta_3$ et $\theta_1 \neg_c \theta_4$;

Dans ce contexte, la recherche d'une substitution θ telle que $D\theta \subseteq E$ revient à trouver un ensemble de contraintes compatibles $\mathcal{E} = \{\theta_{1,i_1}, \theta_{2,i_2}, ..., \theta_{n,i_n}\}$ puisque par construction la substitution θ résultant de leur combinaison permet l'appariement de tout atome δ_i de D avec un atome ε_j de E. Réciproquement, s'il n'existe aucun ensemble $\mathcal{E}$ de contraintes compatibles, alors il n'existe pas de substitution θ telle que $D\theta \subseteq E$.

$$\mathcal{L} = \begin{bmatrix} \delta_1 & : & \theta_{1,1} & ... & & \boxed{\theta_{1,i_1}} & & ... & \theta_{1,p_1} \\ \delta_2 & : & \theta_{2,1} & ... & \boxed{\theta_{2,i_2}} & & & ... & \theta_{2,p_2} \\ & & ... & & & & & & \\ \delta_n & : & \theta_{n,1} & ... & & & \boxed{\theta_{n,i_n}} & ... & \theta_{n,p_n} \end{bmatrix}$$

$$\mathcal{L} \equiv \underbrace{(\theta_{1,1} \vee \theta_{1,2} \vee ... \vee \theta_{1,p_1})}_{appariement\ de\ \delta_1} \wedge \underbrace{(\theta_{2,1} \vee \theta_{2,2} \vee ... \vee \theta_{2,p_2})}_{appariement\ de\ \delta_2} \wedge ... \wedge \underbrace{(\theta_{n,1} \vee \theta_{n,2} \vee ... \vee \theta_{n,p_n})}_{appariement\ de\ \delta_n}$$

Substitution solution : $\theta = \theta_{1,i_1} +_c \theta_{2,i_2} +_c ... +_c \theta_{n,i_n}$

FIG. 3.8: Résolution du test de subsomption.

La partie de l'algorithme d'appariement consacrée à la réduction de la taille de l'espace avant de procéder à la recherche proprement dite est basée sur le principe que tout élagage de coût raisonnable (donc au plus polynomial en temps et en mémoire) doit être effectué, puisque le gain apporté est exponentiel. Parmi toutes les méthodes de simplification, la seule pouvant conduire à une augmentation de taille mémoire (la combinaison des

contraintes) est contrôlée par l'utilisateur grâce à un paramètre fixé lors de l'initialisation de $\mathcal{SIAO}1$. Cette partie se déroule en quatre étapes :

1. Détection des composantes indépendantes du problème.
2. Suppression des contraintes redondantes ou globalement insatisfiables.
3. Combinaison contrôlée des contraintes et réorganisation de la table.

Détection des composantes indépendantes du problème

Une première technique de réduction de l'espace de recherche consiste à isoler d'éventuelles composantes indépendantes. La détection de telles composantes permet un gain considérable puisque n composantes indépendantes $\mathcal{C}_1$, ..., $\mathcal{C}_n$ comportant respectivement $|\mathcal{C}_1|$, ..., $|\mathcal{C}_n|$ choix possibles peuvent alors être résolues avec un coût dans le pire des cas de $\sum_{i=1}^{n} |\mathcal{C}_i|$ au lieu de $\prod_{i=1}^{n} |\mathcal{C}_i|$. Dans le cas présent, des sous-ensembles de lignes peuvent être résolus indépendamment si leurs contraintes portent sur des ensembles de variables disjoints. Comme l'illustre l'algorithme ci-dessous, de telles composantes peuvent être aisément obtenues grâce à l'utilisation de masques booléens mémorisant la présence ou l'absence de contraintes portant sur une variable dans une ligne donnée. On notera par la suite μ de tels masques (qui sont composés d'une liste de $|Vars(D)|$ booléens) et $\mu(k)$ le k^{ieme} élément de μ.

Définition 3.7 *Masque μ_i d'une ligne de contraintes δ_i*
On dit que μ_i est le masque associé à la ligne δ_i si et seulement si :
$\forall j \in [1..|Vars(D)|], \mu(j) = \begin{cases} 1 \ si \ \exists k \ / \ \theta_{i,k}(j) \neq * \ ; \\ 0 \ sinon. \end{cases}$

Définition 3.8 *Intersection et union de deux masques*
L'intersection (resp. union) de deux masques μ_1 et μ_2, notée $\mu_1 \cap \mu_2$ (resp. $\mu_1 \cup \mu_2$) est le masque μ / $\forall k, \mu(k) = \mu_1(k) \wedge \mu_2(k)$ (resp. $\mu_1(k) \vee \mu_2(k)$).

L'algorithme suivant détermine l'ensemble $\mathcal{C}$ des composantes indépendantes de $\mathcal{L}$.

```
/* Test de subsomption optimisé pour SIAO1, partie 2/5 */
/* Détection des composantes indépendantes du problème. */
C ← ∅
composante_courante ← ∅
couverts ← ∅
∀k ∈ [1..n], μ_k ← masque(δ_k)
Tant que ∃δ_i / δ_i ∉ couverts
    composante_courante ← composante_courante ∪ {δ_i}
    couverts ← couverts ∪ {δ_i}
    Tant que ∃μ_j / μ_j ∩ μ_i ≠ (0, ..., 0)
        μ_i ← μ_i ∪ μ_j
        composante_courante ← composante_courante ∪ {δ_j}
        couverts ← couverts ∪ {δ_j}
    C ← C ∪ {composante_courante}
    composante_courante ← ∅
Ordonner L en groupant ensemble les lignes appartenant à la même composante.
```

Exemple :

$$\text{Soit} : \mathcal{L} = \left[\begin{array}{lllll} \textit{Masque} & \textit{Contraintes} & & & \\ \hline \mu_1 = [1,0,0,0] & [a,*,*,*] & [b,*,*,*] & [e,*,*,*] & \\ \mu_2 = [0,0,0,1] & [*,*,*,a] & [*,*,*,d] & & \\ \mu_3 = [0,1,1,0] & [*,c,e,*] & [*,*,d,*] & [*,e,*,*] & [*,a,b,*] \\ \mu_4 = [1,0,1,0] & [a,*,d,*] & [e,*,e,*] & [b,*,b,*] & \end{array}\right.$$

Le déroulement de l'algorithme est le suivant :

Initialisation : $\mathcal{C} \leftarrow \{\emptyset\}$, *composante_courante* $\leftarrow \{\emptyset\}$, *couverts* $\leftarrow \{\emptyset\}$.

Étape 1 : *composante_courante* $= \{\mu_1\}$
L'itération permet d'ajouter à *composante_courante* μ_4 puis μ_3.
On obtient : $\mathcal{C} = \{\{\mu_1, \mu_4, \mu_3\}\}$.

Étape 2 : *composante_courante* $= \{\mu_2\}$
On ne peut plus rien ajouter à *composante_courante*.
On obtient : $\mathcal{C} = \{\{\mu_1, \mu_4, \mu_3\}, \{\mu_2\}\}$.

Le résultat final est :

$$\mathcal{L} = \left[\begin{array}{lllll} \textit{Masque} & \textit{Contraintes} & & & \\ \hline \mu_1 = [1,0,0,0] & [a,*,*,*] & [b,*,*,*] & [e,*,*,*] & \\ \mu_3 = [0,1,1,0] & [*,c,e,*] & [*,*,d,*] & [*,e,*,*] & [*,a,b,*] \\ \mu_4 = [1,0,1,0] & [a,*,d,*] & [e,*,e,*] & [b,*,b,*] & \\ \hline\hline \mu_2 = [0,0,0,1] & [*,*,*,a] & [*,*,*,d] & & \end{array}\right.$$

Remarque :

La double ligne dénote les composantes indépendantes et agit comme une coupure lors de la recherche : après avoir trouvé un ensemble de contraintes compatibles pour les trois premières lignes, la recherche peut se poursuivre en ignorant ces dernières. De manière générale, lors de la recherche, tout échec au niveau d'une coupure produira un échec général sans avoir à remettre en question les choix opérés avant celle-ci.

Suppression des contraintes redondantes ou globalement insatisfiables

Une fois les composantes indépendantes déterminées, les autres méthodes de réduction de l'espace de recherche ne travailleront plus que sur chaque composante. Ce paragraphe présente trois méthodes de pré-élagage permettant de réduire l'espace de recherche, que ce soit en largeur (diminution du facteur de branchement) ou en profondeur (suppression des lignes redondantes). Ces méthodes proviennent de la généralisation d'observations expérimentales concernant la structure de $\mathcal{L}$ qui peuvent être formulées plus intuitivement comme suit :

Redondances intra-ligne

Toute contrainte plus spécifique ou équivalente à une autre contrainte d'une même ligne peut être supprimée.

Exemple :

Soit $\mathcal{L} = \begin{bmatrix} \theta_{1,1}[a,*,*] & \theta_{1,2}[b,c,*] & \theta_{1,3}[a,c,*] & \theta_{1,4}[c,a,*] & \\ \theta_{2,1}[a,b,c] & \theta_{2,2}[b,c,a] & \theta_{2,3}[a,*,c] & \theta_{2,4}[c,a,b] & \theta_{2,5}[a,*,c] \end{bmatrix}$

Comme $\theta_{1,1}[a,*,*] \geq_c \theta_{1,3}[a,c,*]$, on peut supprimer $\theta_{1,3}$.

Comme $\theta_{2,3}[a,*,c] \geq_c \theta_{2,1}[a,b,c]$, on peut supprimer $\theta_{2,1}$.

Comme $\theta_{2,3}[a,*,c] \geq_c \theta_{2,5}[a,*,c]$, on peut supprimer $\theta_{2,5}[a,*,c]$.

Redondances inter-lignes

Si toutes les contraintes d'une ligne sont soit plus strictes ou équivalentes, soit incompatibles avec toutes les contraintes d'une autre ligne, alors on peut supprimer cette dernière.

Exemple :

Soit $\mathcal{L} = \begin{bmatrix} \theta_{1,1}[a,*,*] & \theta_{1,2}[b,*,*] & & \\ \theta_{2,1}[a,b,c] & \theta_{2,2}[b,c,a] & \theta_{2,3}[a,*,c] & \theta_{2,4}[b,a,*] \end{bmatrix}$

On peut supprimer la première ligne du tableau.

Cas particulier : Dans le cas où toutes les contraintes d'une ligne sont incompatibles avec toutes les contraintes d'une autre ligne, aucun appariement n'est possible et on a immédiatement un `échec final` .

Exemple :

Soit $\mathcal{L} = \begin{bmatrix} \theta_{1,1}[a,*,*] & \theta_{1,2}[b,a,*] & \theta_{1,3}[c,b,*] \\ \theta_{2,1}[b,c,a] & \theta_{2,2}[c,a,b] & \theta_{2,3}[b,c,c] \end{bmatrix}$

Il n'y a pas appariement.

Incompatibilités globales

S'il existe une contrainte $\theta_{i,j}$ incompatible avec l'ensemble des contraintes d'une ligne $i' \neq i$, alors on peut supprimer $\theta_{i,j}$.

Exemple :

Soit $\mathcal{L} = \begin{bmatrix} \theta_{1,1}[a,*,*] & \theta_{1,2}[b,a,*] & \theta_{1,3}[c,b,*] \\ \theta_{2,1}[b,c,a] & \theta_{2,2}[c,b,a] & \theta_{2,3}[a,b,c] \end{bmatrix}$

On peut supprimer $\theta_{1,2}$ et $\theta_{2,1}$.

L'algorithme qui découle de ces observations est immédiat :

/* Test de subsomption optimisé pour $\mathcal{SIAO}1$, partie 3/5 */

/* Supp. des contraintes redondantes ou globalement insatisfiables. */

$\forall l, i_1, i_2 \in [1..n] \times [1..p_l] \times [1..p_l]$ tels que $i_1 \neq i2$

Si $\theta_{l,i_1} \geq_c \theta_{l,i_2}$ Alors supprimer θ_{l,i_2}

$\forall l_1, l_2 \in [1..n] \times [1..n]$ tels que $l_1 \neq l_2$ et l_1, l_2 ne sont pas indépendantes

Si $\forall i_1, i_2 \in [1..p_{l_1}] \times [1..p_{l_2}]$, $\theta_{l_1,i_1} \neg_c \theta_{l_2,i_2}$ Alors $\rightarrow$ échec final

Si $\forall i_1, i_2 \in [1..p_{l_1}] \times [1..p_{l_2}]$, $(\theta_{l_1,i_1} \neg_c \theta_{l_2,i_2}) \vee (\theta_{l_1,i_1} \geq_c \theta_{l_2,i_2})$
Alors supprimer l_1

$fin\_point\_fixe \leftarrow faux$

Tant que $\neg fin\_point\_fixe$

$fin\_point\_fixe \leftarrow vrai$

$\forall l_1, i_1, l_2 \in [1..n] \times [1..p_{l_1}] \times [1..n]$ tels que $l_1 \neq l_2$ et l_1, l_2 ne sont pas indépendantes

Si $\forall i_2 \in [1..p_{l_2}], \theta_{l_1,i_1} \neg_c \theta_{l_2,i_2}$ Alors

supprimer θ_{l_1,i_1}

$fin\_point\_fixe \leftarrow faux$

Si $\exists l \in [1..n]$ / l ne contient plus aucune contrainte θ, Alors $\rightarrow$ échec final

Combinaison contrôlée des contraintes et réorganisation de la table

Après avoir découpé la table des contraintes suivant ses composantes indépendantes puis supprimé au sein de chaque composante les contraintes redondantes ou incompatibles avec une autre ligne, il est encore possible d'étudier les incompatibilités pour un nombre $n \geq 2$ de lignes. Soit par exemple la table 3.1.

$$\mathcal{L} = \left[\begin{array}{cccccc} \textit{Masque} & \textit{Contraintes} & & & & \\ \hline [1,1,0] & \theta_{1,1}[a,b,*] & \theta_{1,2}[b,c,*] & \theta_{1,3}[b,a,*] & \theta_{1,4}[c,d,*] & \boxed{\theta_{1,5}[d,d,*]} \\ [0,1,1] & \theta_{2,1}[*,b,d] & \theta_{2,2}[*,c,a] & \theta_{2,3}[*,a,c] & \boxed{\theta_{2,4}[*,d,b]} & \\ [1,0,1] & \theta_{3,1}[a,*,a] & \theta_{3,2}[b,*,d] & \theta_{3,3}[b,*,b] & \theta_{3,4}[c,*,c] & \boxed{\theta_{3,5}[d,*,b]} \end{array}\right]$$

TAB. 3.1: Contraintes incompatibles par combinaison.

On vérifie aisément que :

1. il n'existe aucune composante indépendante séparable ;
2. il n'existe aucune redondance intra-ligne ;
3. il n'existe aucune redondance inter-lignes ;
4. il n'existe aucune incompatibilité globale ;

ce qui implique que les méthodes de simplification évoquées jusqu'à présent seront inefficaces sur cette table, qui cependant est difficile puisque la seule substitution solution est $[d, d, b]$, obtenue suivant le chemin $\theta_{1,5} +_c \theta_{2,4} +_c \theta_{3,5}$. Comme suggéré précédemment, la difficulté provient du fait qu'une substitution $\theta_{i,j}$ peut être compatible avec n ensembles disjoints de substitutions sans pour autant être supprimée. Ainsi, dans l'exemple précédant, il existe de nombreux groupes de substitutions $\{\theta_{1,x}, \theta_{2,y}, \theta_{3,z}\}$ tels qu'une seule des substitutions est compatible avec les deux dernières (qui sont donc incompatibles entre elles). La notion d'incompatibilité globale est alors insuffisante, puisqu'il faut être capable pour détecter l'incompatibilité de considérer la compatibilité d'une substitution avec un ensemble de lignes du tableau. La figure 3.9 représente le problème relatif au tableau 3.1 d'une manière très proche de [Scheffer et al.96] :

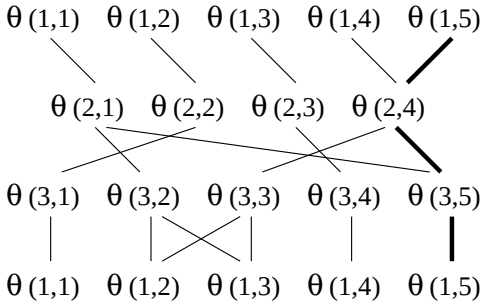


FIG. 3.9: Vue graphique du problème d'incompatibilité des contraintes par combinaison.

À chaque nœud du graphe est associée une substitution partielle correspondant à une contrainte du tableau. Afin de conserver l'organisation du tableau et d'améliorer la lisibilité, l'organisation du problème en lignes est reprise, et la première ligne est répétée en bas du graphe. Deux nœuds appartenant à deux lignes différentes sont reliés par un arc si les substitutions associées sont compatibles.

Sous cette forme, la recherche d'une substitution solution pour le problème de l'appariement consiste à trouver un chemin[6] entre un nœud quelconque de la première ligne et le même nœud $\theta_{1,x}$ répété sur la dernière ligne. L'unique solution, qui figure en gras sur le graphe, n'est trouvée (dans le cas d'une recherche classique) qu'après un nombre important d'essais inutiles.

Une solution pour réduire, voire supprimer la composante inutile du graphe consiste à combiner les lignes entre elles :

Définition 3.9 $\times_c$
On appelle composition de deux ensembles de contraintes $\mathcal{E}_1$ *et* $\mathcal{E}_2$ *l'ensemble* $\mathcal{E}_1 \times_c \mathcal{E}_2 = \bigcup_{\theta_1 \in \mathcal{E}_1, \theta_2 \in \mathcal{E}_2} \{\theta_1 +_c \theta_2\}$.

[6] Afin de dissiper toute méprise, il paraît nécessaire de préciser que le problème étant ici de taille trois, la recherche d'une clique de trois nœuds dans le graphe correspondant au tableau est équivalente à la recherche d'un chemin dans le graphe avec répétition de ligne. Bien entendu, dans le cas général ($n > 3$) la recherche d'une clique de taille n n'est pas équivalente à la recherche d'un chemin, et la simplification apportée par la représentation ci-dessus n'est plus exploitable.

Cependant, la combinaison de deux lignes peut produire dans le pire des cas un accroissement exponentiel de la taille de la table des contraintes, et ce même lorsque les lignes regroupées ne sont pas indépendantes :

Propriété 3.9 $|\mathcal{E}_1 \times_c \mathcal{E}_2|$ *(cas général).*
On a toujours : $0 \leq |\mathcal{E}_1 \times_c \mathcal{E}_2| \leq |\mathcal{E}_1|.|\mathcal{E}_2|$.

Propriété 3.10 $|\mathcal{E}_1 \times_c \mathcal{E}_2|$ *(après élagage).*
Lorsque $\mathcal{E}_1$ *et* $\mathcal{E}_2$ *ne sont pas indépendants et ne comportent aucune contrainte globalement incompatible, on a :* $Max(|\mathcal{E}_1|, |\mathcal{E}_2|) \leq |\mathcal{E}_1 \times_c \mathcal{E}_2| < |\mathcal{E}_1|.|\mathcal{E}_2|$.

Définition 3.10 *Taux de compression*
On appelle taux de compression le rapport : $Tc(\mathcal{E}_1, \mathcal{E}_2) = 1 - \frac{|\mathcal{E}_1 \times_c \mathcal{E}_2|}{|\mathcal{E}_1|.|\mathcal{E}_2|}$.

$$\mathcal{L} = \begin{bmatrix} & (\mathbf{1}) & \\ \theta_{(k,1)} & \ldots & \theta_{(k,p_k)} \\ \theta_{(k+1,1)} & \ldots & \theta_{(k+1,p_{k+1})} \\ & (\mathbf{2}) & \end{bmatrix} \xrightarrow{combinaison} \mathcal{L} = \begin{bmatrix} & (\mathbf{1}) & \\ \theta_{(k',1)} & \ldots & \theta_{(k',|\mathcal{E}_k \times_c \mathcal{E}_{k+1}|)} \\ & (\mathbf{2}) & \end{bmatrix}$$

FIG. 3.10: Gain apporté par la combinaison de lignes.

Comme le montre la figure 3.10, le gain produit par la combinaison des deux lignes k et $k+1$ en une ligne k' lors de l'exploration de l'espace peut être mesuré en nombre de tests de compatibilité évités lors de la prise en compte des contraintes des deux lignes (soit $|\mathcal{E}_1|.|\mathcal{E}_2| - |\mathcal{E}_1 \times_c \mathcal{E}_2|$) ou encore comme un ratio définissant un taux de compression Tc (cf. définition 3.10). Un premier élagage est obtenu lors de l'exploration de la partie supérieure du tableau. En effet le coût de la prise en compte des lignes k et $k+1$ peut être réduit à chaque retour sur échec dans la partie **(1)** – et il y en a dans le pire des cas un nombre exponentiel par rapport au nombre de lignes de **(1)** – d'un facteur Tc. La réduction se comporte alors comme une mémoire évitant de reproduire lors de chaque prise en compte de ces lignes les retour-arrières sur échec entre les lignes k et $k+1$. De plus, l'élagage cumulé de plusieurs (k) combinaisons de lignes produit également une décroissance exponentielle de la taille de l'espace de recherche, ce dernier étant globalement réduit d'un facteur $\prod_{i=1}^{i=k} Tc_i$ ($= Tc^k$ dans le cas d'un gain Tc_i constant).

En conséquence, l'algorithme contrôlera la combinaison des lignes en fonction de la taille des lignes résultant des différents regroupements possibles. En effet, non seulement l'accroissement exponentiel de la taille de la table rend la combinaison systématique de lignes impossible, mais cette dernière n'est pas souhaitable puisque son intérêt du point de vue de la recherche est proportionnel au taux de compression apporté. Comme le décrit le pseudo-code ci-dessous, les lignes sont regroupées suivant une stratégie de type meilleur

d'abord , et ce tant que la taille de la ligne résultant du regroupement envisagé ne dépasse pas une borne fixée et qu'il existe des lignes non indépendantes optimisables.

Pour finir, le tableau peut être réorganisé d'une part en forçant la prise en compte des contraintes inévitables dès le début de la recherche (ce qui réduit d'autant l'espace à explorer), et d'autre part en appliquant une heuristique de type les composantes indépendantes les moins complexes d'abord , comme décrit dans la seconde partie du pseudo-code.

```
/* Test de subsomption optimisé pour SIAO1, partie 4/5 */
/* Combinaison contrôlée des contraintes. */
∀C_i (composante indépendante) :
    fin_point_fixe ← faux
    Tant que ¬fin_point_fixe
        trouver (l_i, l_j) ∈ C_i × C_i tels que
          (l_i ≠ l_j)  ∧  (|l_i ×_c l_j|  minimum)
        Si (|l_i ×_c l_j| ≤ Max)  ∨  (|l_i ×_c l_j| ≤ l_i)  ∨  (|l_i ×_c l_j| ≤ l_j)
            Alors combiner les lignes l_i et l_j
            Sinon fin_point_fixe ← vrai
/* Réordonnancement de l'espace de recherche. */
placer les composantes indépendantes par ordre de |C_i| croissant
déplacer les lignes ne comportant qu'une unique contrainte en tête
du tableau
```

3.2.5 Parcours de l'espace de recherche

Le pseudo-code ci-dessous décrit la recherche proprement dite d'une substitution solution par le parcours du tableau $\mathcal{L}$ de n listes de contraintes qui a été simplifié. La dérécursivation se fait au moyen de deux tableaux qui contiennent les informations correspondant aux paramètres gérés par la pile dans le cas de l'algorithme récursif. Soit i la ligne de travail courante, `Indices` permet de mémoriser les numéros des contraintes (compatibles) constituant le chemin en cours d'exploration, alors que `Contraintes` correspond à l'accumulation des contraintes portant sur la solution en cours de construction. La notation `V++`, empruntée au langage C, est utilisée afin de gagner en concision et en compréhensibilité par rapport à l'écriture `V` $\leftarrow$ `V+1`.

Le mécanisme de propagation des contraintes (de type *forward checking*) est décrit dans la fonction `Propage` et met en œuvre deux tableaux supplémentaires. Le premier, `Propa`, est de même dimension que $\mathcal{L}$ et associe à chaque contrainte une valeur dans l'ensemble $\{compatible, 1, 2, ..., n-1\}$ suivant que la contrainte associée est compatible avec la solution potentielle propagée ou non (dans ce dernier cas, le numéro de la première ligne en conflit est également mémorisé). Le second tableau, `Compatibles`, contient pour chaque

ligne de $\mathcal{L}$ le nombre de contraintes dont le statut est *compatible* avec le choix en cours de propagation. Dès que ce nombre devient nul pour une ligne, il est nécessaire de revenir sur le dernier choix propagé.

Finalement, l'heuristique classique les choix les plus contraints d'abord est implantée en choisissant, après chaque propagation, la ligne restante de la composante indépendante courante dont la valeur de `Compatibles` est minimale. Son fonctionnement est décrit dans la procédure `Réordonne_`$\mathcal{L}$.

```
/* Test de subsomption optimisé pour SIAO1, partie 5/5 */

Créer le tableau Contraintes[n]  ←  [[* * ... *], ..., [* * ... *]]
Créer le tableau Indices[n + 1]  ←  [1, 1, ..., 1]
Créer le tableau Propa[n][p_i]  ←  [[compatible, ...], ...]
Créer le tableau Compatibles[n]  ←  [p_1, ..., p_n]
i ← 1
Tant que (i ≤ n)  ∧  (Indices[1] ≤ p_1)
     Indices[i] ← Prochain_indice_compatible[i]
     Si (i = 1)
          Alors Contraintes[i] ← L[1][Indices[i]]
          Sinon Contraintes[i] ← Contraintes[i − 1] +_c L[i][Indices[i]]
     Si Propage(Contraintes[i], i)
          Alors
              i++
              Indices[i] ← 1
          Sinon Retour_arrière(i)
Si (i = n + 1)   Alors → succès final   Sinon → échec final

/* Propagation de la contrainte C de la ligne l_1 */

Fonction Propage(C, l_1)
Soit l_2 l'indice maximum des lignes de même composante indépendante que l_1
Si (l_1 = l_2) Alors → succès
Sinon, ∀l ∈ [l_1 + 1, l_2] :
     ∀ contrainte c ∈ L[l] telle que Propa[l][indice(c)] = compatible
     Si C ¬_c c Alors
          Propa[l][indice(c)] ← l_1
          Compatibles[l]--
          Si (Compatibles[l] = 0) Alors → échec
→ succès
```

/* Retour-arrière sur la contrainte en cours de la $i^{ème}$ ligne */

Procédure Retour_arrière(i)

i--

Tant que $(i > 0) \wedge (Prochain\_indice\_compatible[i] > p_i)$

i--

Remplacer dans Propa toutes les valeurs supérieures ou égales à i par la valeur *compatible* et répercuter ces modifications sur le tableau Compatibles

/* Heuristique les choix les plus contraints d'abord */

Procédure Réordonne_$\mathcal{L}$(l_1)

Permuter dans les tableaux Contraintes, Indices, Propa et Compatibles la ligne $l_1 + 1$ avec la ligne de même composante indépendante que l_1, d'indice supérieur à l_1, pour laquelle Compatibles$[l_1]$ est minimum (si elle existe)

3.2.6 Propriétés de l'algorithme de subsomption

Analyse de la complexité de l'algorithme

La complexité de la relation de subsomption proposée ici étant de même grandeur asymptotique que la complexité du test de θ-subsomption (voir section 3.1.3), il n'est pas étonnant d'obtenir une fonction exponentielle de la taille des entrées puisque le test de θ-subsomption est un problème NP-difficile. Deux points méritent cependant d'être notés :

1. La complexité de la partie de l'algorithme consacrée au pré-traitement (construction du tableau $\mathcal{L}$) est une fonction polynomiale de $|E|$ et de $|D|$. Du point de vue des techniques de propagation de contraintes [Tsang93], cette partie correspond à une vérification de la cohérence par nœud puis par arc associée à la détection des composantes indépendantes. Le mécanisme de simplification des composantes fortement contraintes correspond à une procédure de cohérence par chemin qui regroupe certains nœuds choisis du graphe et dont la profondeur est adaptée à la complexité de chacune des composantes considérées.

2. Par contre, la complexité de la recherche de la substitution solution dans le tableau $\mathcal{L}$ n'est pas une fonction polynomiale. En supposant que $\mathcal{L}$ comporte d lignes ($1 \leq d \leq |D|$) d'au plus e contraintes ($1 \leq e \leq |E|$), la complexité de la recherche peut être bornée par $e.d.e^d$ tests de compatibilité (puisque chacun des e^d tests de compatibilité est complété par une propagation de type *lookahead* (il s'agit d'un *forward checking*) qui nécessite dans le pire des cas $e.d$ tests de compatibilité).

Comme beaucoup de problèmes de satisfaction de contraintes difficiles, il est malaisé de justifier *a priori* une méthode de réduction ou une méthode de recherche : ces dernières sont toutes adaptées à différents types de problèmes. Il n'est pas rare en effet de constater expérimentalement que l'élagage de meilleure qualité obtenu à l'aide d'une méthode de réduction donnée ne compense pas en pratique le surcoût de calcul provoqué. Ce phénomène est illustré par la figure 3.11 de la section 3.2.8 dans le cas de l'élagage basé sur le contexte de graphe utilisé dans le test de subsomption de T. Scheffer. Cette même section montre (empiriquement) que le test proposé est adapté aux problèmes qui ont motivé son développement.

Adaptation à la PLI sous θ-subsomption

Le cadre de la programmation logique comporte deux particularités non partagées avec $\mathcal{SIAO}1$.

D'une part, contrairement à $\mathcal{SIAO}1$, l'existence possible de variables dans les têtes des clauses modifie l'énoncé du problème qui devient :

> *Soient deux clauses de Horn* $D : D_{tete} \leftarrow D_{corps}$ *et* $E : E_{tete} \leftarrow E_{corps}$. *Déterminer s'il existe une substitution* θ *vérifiant :*
>
> **(1)** $D_{tete}\theta = E_{tete}$,
>
> **(2)** $D_{corps}\theta \subseteq E_{corps}$.

Il est cependant possible de se ramener au problème de $\mathcal{SIAO}1$ de la manière suivante :

- Dans un premier temps, chercher une substitution θ_1 telle que $D_{tete}\theta_1 = E_{tete}$. Puisque D_{tete} et E_{tete} sont deux atomes, θ_1 est unique (si elle existe) et la trouver est un problème élémentaire.
- Si θ_1 n'existe pas, D ne peut subsumer E puisque la contrainte **(1)** n'est plus respectée.
- Dans le cas contraire, il reste à résoudre **(2)** en propageant θ_1, ce qui revient à chercher θ_2 telle que $D_{corps}\theta_1\theta_2 \subseteq E_{corps}$. En posant $D'_{corps} = D_{corps}\theta_1$ on retrouve alors un cas particulier du problème résolu dans ce chapitre puisqu'il s'agit de chercher θ_2 telle que $D'_{corps}\theta_2 \subseteq E_{corps}$ dans le cas d'un langage ne comportant ni arguments étendus (disjonctions ou intervalles), ni hiérarchies, ni symboles `?` et `*`.

D'autre part, l'impossibilité de traiter les symboles de fonction (ce qui est également une caractéristique de la plupart des systèmes de PLI) pose à première vue également problème. Cependant, cette difficulté peut être aisément contournée en procédant par aplatissement des formules comportant de tels symboles [Rouveirol94].

3.2.7 Autres approches

Parmi les nombreux travaux qui portent sur les solutions pratiques à apporter au problème de la complexité du test de θ-subsomption, il est possible de distinguer deux grandes approches.

Modifier le cadre du problème pour rendre la complexité de $\vdash_\theta$ polynomiale

Cette première approche est réalisée soit en limitant les ressources allouées au test de subsomption (ce qui conduit par exemple M. Sebag et C. Rouveirol à définir la notion d'appariement stochastique [Sebag et al.97a]), soit en contraignant la forme des clauses (par exemple [Kietz et al.94] limitent la taille k des composantes existentielles non déterminées des clauses, appelées k-locales : la complexité du test de subsomption n'est plus alors en $|E|^{|D|}$ mais en $|E|^k$).

Optimiser les algorithmes implantant le test $\vdash_\theta$

Les travaux faisant référence dans ce domaine [Scheffer et al.96] proposent une méthode de résolution du test en deux temps. La première étape consiste à élaguer l'espace de recherche au moyen du contexte de graphe, implanté par l'auteur pour une taille $k = 2$. La seconde étape reformule le problème de la subsomption comme un problème de recherche d'une clique dans un graphe dont les nœuds sont les substitutions candidates. Les arêtes représentent un lien de compatibilité entre deux substitutions candidates.

Il convient également de mentionner les travaux de [Kim et al.92] qui proposent une représentation tabulaire similaire à celle qui a été présentée ici. Cependant, si un codage des contraintes à l'aide de chaînes de bits permet des tests de compatibilité très efficaces, les seuls mécanismes de réduction de l'espace de recherche mis en œuvre correspondent à la détection des incompatibilités globales et des composantes indépendantes de taille $k = 1$. Aucun des autres mécanismes de réduction (composantes indépendantes de taille $k > 1$, combinaison des lignes, redondances inter et intra-lignes, etc.) n'est proposé. L'exploration de l'espace de recherche est réalisée par une méthode exhaustive puisque toutes les contraintes d'une solution potentielle sont combinées avant de tester la validité de cette solution. Aucun mécanisme de propagation de contraintes n'est utilisé.

3.2.8 Évaluation expérimentale

Le test de subsomption (noté $\vdash \mathcal{SIAO}1$) a été implanté en C++. Du fait d'une conception orientée-objet proposant une méthode d'ajout incrémental des contraintes dans $\mathcal{L}$, il a bénéficié d'une amélioration (mineure) par rapport à la version décrite dans ce chapitre : il est en effet possible de procéder à certains élagages relatifs au pré-traitement au fur et à mesure de l'arrivée des contraintes.

La validation expérimentale de $\vdash \mathcal{SIAO}1$ est tout simplement réalisée en constatant que $\mathcal{SIAO}1$ est capable de travailler dans les espaces relationnels complexes qui ont motivé

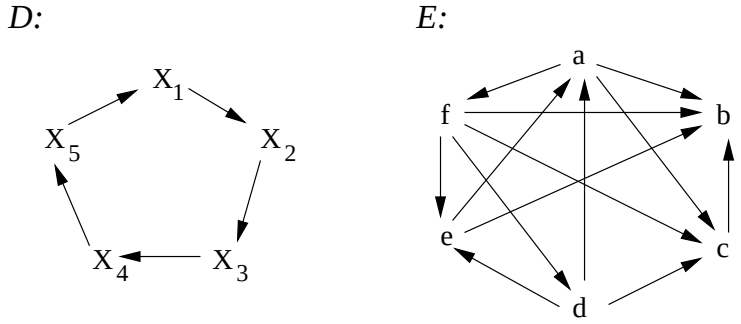
son élaboration (comme par exemple le problème *Mutagenesis* décrit en annexe B section B.4), alors qu'auparavant le même algorithme ne terminait pas sur ces mêmes problèmes (en un temps raisonnable et avec les machines actuelles).

Afin de quantifier plus précisément le gain apporté, trois tests ont été réalisés :

- Le premier compare sur un problème très simple d'appariement de graphes le comportement du test de subsomption canonique (décrit section 3.2.1) et celui du test de subsomption optimisé.
- Le second propose une comparaison avec l'algorithme de T. Scheffer [Scheffer et al.96] (décrit section 3.2.7) sur un problème difficile construit à l'aide des données de *Mutagenesis*.
- Le troisième a été réalisé afin d'apporter des informations plus précises quand aux causes des différences de comportement constatées sur le problème précédent.

Comparaison avec le test canonique

Afin de montrer l'apport du test proposé par rapport à la version canonique classique, un problème jouet de couverture de graphes a été construit :



Les graphes sont représentés au moyen des littéraux $objet()$ et $lien(,)$. Par exemple le graphe D s'écrit :

$$concept \leftarrow objet(X_1) \wedge ... \wedge objet(X_5) \wedge lien(X_1, X_2) \wedge ... \wedge lien(X_5, X_1).$$

Comme le montre le tableau ci-dessous, il est possible d'accélérer le test de subsomption de plus de trois ordres de grandeur sur cet exemple, qui est un problème extrêmement simple (les graphes ne comportent respectivement que 5 et 6 sommets).

$T_{\vdash_{\mathcal{SIAO}_1}}$ (s)	$T_{\vdash_{canonique}}$ (s)	Accélération
$5,025.10^{-4}$	$1,287$	2 561

Un tel décalage peut être expliqué en constatant que dans le cas du test optimisé, la réduction (de complexité polynomiale) de la table $\mathcal{L}$ est suffisante pour connaître le résultat du test (à savoir que D ne couvre pas E), alors que dans le cas du test canonique, pas moins de 248 245 tests de compatibilité sont nécessaires pour arriver au même résultat. De plus, on remarquera également que si le nombre de tests de compatibilité est peu important par rapport au nombre de combinaisons possibles entre littéraux ($|E|^{|D|} = 19^{10} \simeq 6,13.10^{12}$), il est supérieur au nombre de valuations possibles des variables ($6^5 = 7\ 776$).

Comparaison avec le test proposé par T. Scheffer

Une seconde étape à la validation du test de subsomption de $\mathcal{SIAO}1$ consiste à comparer ce dernier aux meilleurs tests de θ-subsomption. Cependant, il est important de souligner que le test de subsomption sur $\mathcal{L}_h$ est différent d'un test de θ-subsomption classique puisque si tout problème de θ-subsomption se ramène à un problème de subsomption sur $\mathcal{L}_h$ (comme l'a montré la section 3.2.6), la réciproque est fausse. Considérons par exemple le problème de subsomption suivant :

$$P(X,X) \stackrel{?}{\vdash} P(a,*)$$

La relation de subsomption est vérifiée car le symbole $*$ n'introduit aucune contrainte, conformément à la sémantique établie (la substitution solution est alors $\theta = \{X/a\}$). Cependant un test de θ-subsomption classique ne mettant en œuvre aucun mécanisme de traitement spécial de $*$ sera confronté à l'impossibilité de trouver une substitution solution puisque d'une part dans le contexte de la PLI le problème de l'appariement se réduit à la recherche d'une substitution, et que d'autre part une substitution est une fonction (donc $\theta = \{X/a, X/*\}$ n'est pas une substitution).

Une autre indication quand à la modification de la nature du problème induite par le symbole $*$ est donnée par l'étude des pré-traitements fondés sur les signatures. En effet, une méthode très efficace d'élagage consiste à associer à chaque constante ou variable (resp. littéral) une représentation, appelée signature, caractérisant l'environnement structurel de l'objet considéré dans la formule. Soit par exemple :

$$F : P(X,Y) \wedge P(Y,Z) \wedge P(Z,Z)$$

L'environnement structurel de la variable X est caractérisé par le fait que cette dernière ne peut apparaître qu'en première position dans le prédicat P, ce qui sera noté par $X : P1$. De manière analogue, les signatures des autres variables sont $Y : P1\ P2$ et $Z : P2\ P12$. De telles contraintes structurelles fortes permettent un élagage très efficace de l'espace de recherche puisqu'il est possible d'éliminer des solutions candidates toutes celles dont les éléments appariés ne partagent pas la même signature. Cependant, la sémantique du symbole $*$ rend impossible l'utilisation d'un tel mécanisme puisque l'équivalence des

signatures de deux symboles n'est plus une condition nécessaire pour que leur appariement puisse générer une solution au problème de la subsomption.

Tobias Scheffer a eu l'extrême gentillesse de nous fournir le code source (écrit en langage C) du test de subsomption proposé dans [Scheffer et al.96], qui fait référence dans le domaine. Le protocole d'expérimentation mis en œuvre pour comparer les deux algorithmes est le suivant :

- Des données réelles sont utilisées. Elles proviennent du problème *Mutagenesis*.
- Il est possible de paramétrer la difficulté du test en faisant varier un paramètre n qui correspond au nombre de littéraux de l'hypothèse.
- Pour une valeur de n donnée, chaque test est exécuté sur les mêmes instances D et E, où E est la molécule la plus complexe (en nombre de littéraux) de la base *Mutagenesis* (on a $|E| = 84$), et D est une hypothèse obtenue par extraction de n littéraux différents de E (déterminés aléatoirement) dont les arguments relationnels sont variabilisés.

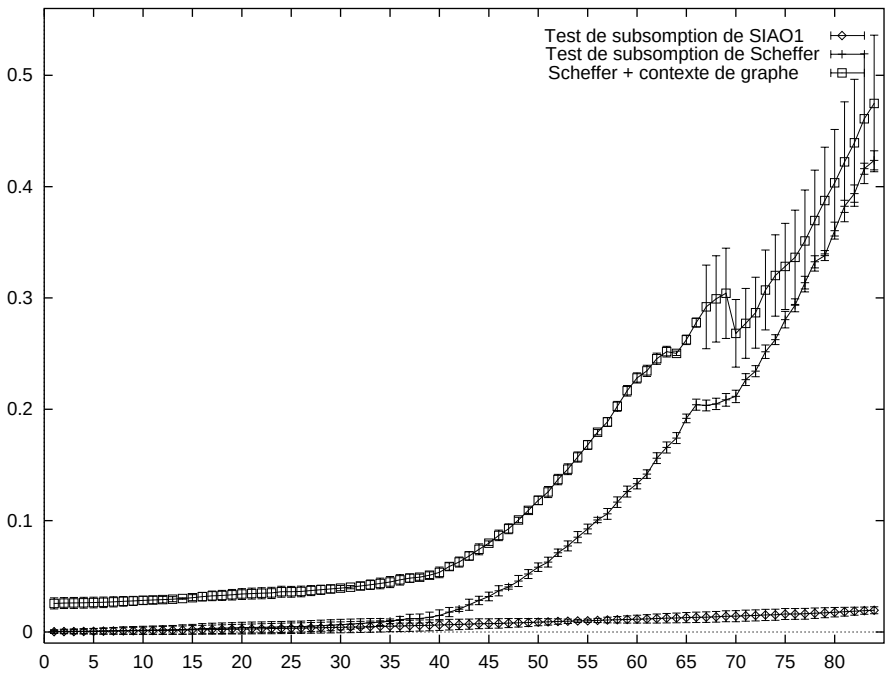


FIG. 3.11: Performance des tests de subsomption sur le problème *Mutagenesis*.

Les figures 3.11 et 3.12 ont été obtenues en générant plusieurs milliers d'hypothèses pour chaque valeur de n. Les temps moyens (en secondes) ainsi que les écarts-types ont été calculés. Il est possible d'observer sur ce problème :

- que la croissance du coût du test de subsomption proposé par Scheffer est bien plus importante que ce que l'on observe dans le cas de $\mathcal{SIAO}1$;

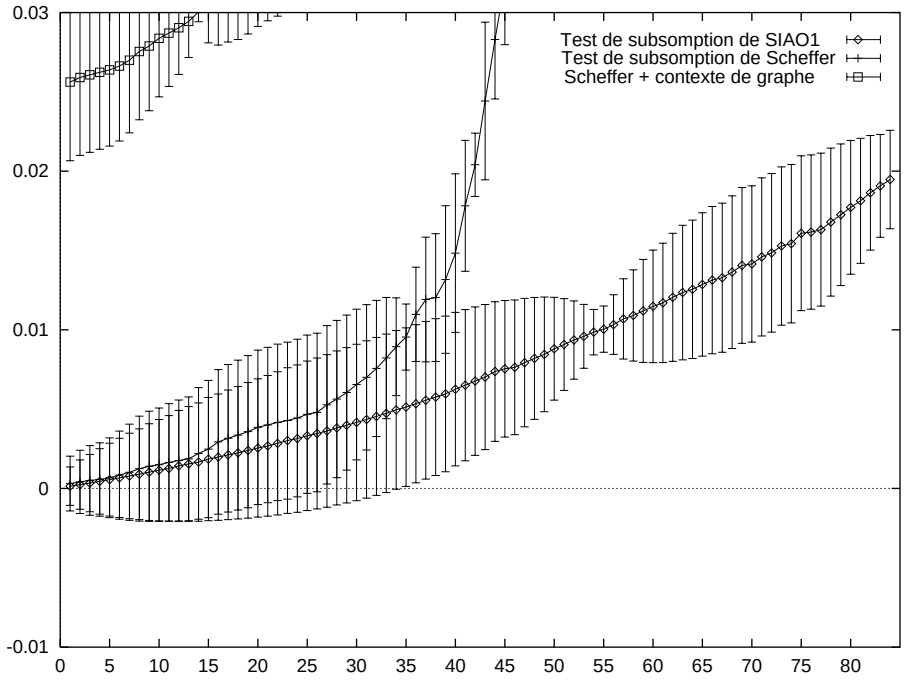


FIG. 3.12: Performance des tests de subsomption sur le problème *Mutagenesis* (détail).

– que l'utilisation du contexte de graphe avec $k = 2$ pour élaguer l'espace de recherche dégrade les performances du test de subsomption proposé par Scheffer.

L'inutilité du contexte de graphe peut s'expliquer aisément par le fait que lorsque $k = 2$, il n'est possible de considérer au mieux que des structures de type $atome(X, ...) \wedge lien(X, Y)$, qui sont bien trop simples pour permettre un élagage structurel efficace. Par contre, il est plus difficile de comprendre pourquoi le test fondé sur la recherche d'une clique est moins efficace que le test fondé sur un algorithme de propagation de contraintes. Il est néanmoins possible de remarquer que, dans le cas d'une hypothèse D (resp. une instance E) de $|D|$ (resp. $|E|$) littéraux dont l'arité des prédicats est bornée par k, la taille des structures manipulées par les deux algorithmes est différente. En effet, dans le cas du test implanté pour $\mathcal{SIAO}1$, la taille de la table des contraintes est bornée par $k.|E|.|D|$, alors que la taille du graphe utilisé par le test proposé par Scheffer[7] est proportionnelle à $(|E|.|D|)^2$.

Coût en espace des algorithmes

Un dernier test a été réalisé afin de mesurer les tailles mémoires respectives des deux tests de subsomption. Le problème choisi est une reformulation en logique relationnelle du problème des n reines car ce dernier, considéré comme difficile, ne peut être décomposé. Le tableau ci-après détaille le temps T (mesuré en secondes) nécessaire à la résolution du problème ainsi que la taille mémoire M (mesurée en kilo-octets). Alors que la taille maximale allouée reste très modeste dans le cas du test de subsomption optimisé proposé

[7] En effet, ce graphe représente la compatibilité des substitutions engendrées par les couples de littéraux (d, e) avec $d \in D$ et $e \in E$.

n	$\|D\|$	$\|E\|$	$\vdash \mathcal{SIAO}1$			$\vdash$ Scheffer		
			$E(T)$	$\sigma(T)$	$E(M)$	$E(T)$	$\sigma(T)$	$E(M)$
4	28	58	0,0310	0,00685	1 840	0,2523	0,00679	2 984
5	45	97	0,0808	0,00611	1 952	2,1536	0,05068	11 496
6	66	146	1,1921	1,04033	1 960	20,5678	1,05028	49 792
7	91	205	0,5421	0,07577	2 048	80,4209	3,05927	180 100
8	120	274	70,9252	78.7985	2 160	ND	ND	ND

TAB. 3.2: Comportement des tests $\vdash \mathcal{SIAO}1$ et $\vdash$ Scheffer sur le problème des n reines.

dans ce chapitre, cette dernière se trouve être un paramètre limitant dans le cas du test proposé par T. Scheffer, puisqu'il est nécessaire de disposer de plus de 180 000 Ko de mémoire pour résoudre le problème des 7 reines (les machines disponibles ne disposaient pas de suffisamment de mémoire pour permettre à ce test de résoudre le problème des 8 reines).

3.3 Bilan

Ce chapitre a été tout d'abord l'occasion de détailler le langage de représentation de $\mathcal{SIAO}1$ et de le situer par rapport aux travaux de PLI dans le contexte de l'apprentissage à partir d'interprétations. De par sa construction $\mathcal{L}_h$ combine :

- Les caractéristiques des langages relationnels qui permettent de décrire les relations liant différents objets au moyen de variables et de prédicats.
- Certaines caractéristiques des langages attribut-valeur, qu'il s'agisse de symboles spécifiques (`?`, `*`), ou d'éléments syntaxiques adaptés aux opérateurs de généralisation usuels pour ces langages (disjonctions, intervalles, etc.).
- Une théorie du domaine sous la forme de hiérarchies définies sur les constantes et les prédicats qui permet, par rapport à des mécanismes plus généraux tels que la saturation, de décrire succintement les observations.

$\mathcal{SIAO}1$ peut ainsi aborder à la fois les problèmes classiques d'ordre 0 (qui comprennent souvent de nombreuses valeurs numériques et manquantes) et les problèmes relationnels (dont l'aspect symbolique est prépondérant).

La relation de subsomption a ensuite été définie : tout comme dans le cas de la θ-subsomption, la complexité du test $D \vdash E$ est intrinsèquement exponentielle en fonction de la taille des entrées (sauf si P = NP). Un algorithme de subsomption efficace, basé sur des techniques de réduction de problèmes et de propagation de contraintes a alors été présenté. Son évaluation expérimentale a montré d'une part sa faisabilité dans le cas de données fortement relationnelles, et d'autre part son intérêt par rapport aux complexités en temps et en espace des approches existantes en PLI, bien que le test de subsomption

étudié soit *a priori* plus difficile que le test de θ-subsomption.

Les fondations de $\mathcal{SIAO}1$ ayant été exposées et validées, le chapitre suivant est consacré à l'algorithme de recherche du système.

Chapitre 4

$\mathcal{SIAO}1$

Ce chapitre décrit le système $\mathcal{SIAO}1$ qui apprend, à partir d'exemples, des définitions exprimées dans le langage relationnel $\mathcal{L}_h$. La première section, qui s'appuie sur les résultats mentionnés dans le chapitre 3, présente l'architecture générale du système ainsi que la structure des chromosomes codant les formules. Les différentes composantes de la fonction de qualité sont ensuite explicitées, ce qui permet d'analyser le biais d'évaluation de l'algorithme et de montrer son adéquation à la stratégie de recherche. La section 4.3 est consacrée aux opérateurs d'évolution (mutation et croisement), et souligne les différences importantes qui, combinées à la stratégie de recherche adoptée, distinguent nettement le système proposé des autres approches évolutionnaires. Les deux sections suivantes sont consacrées d'une part au post-traitement et à la classification de nouvelles instances à l'aide des règles obtenues, et d'autre part aux connaissances du domaine d'ordre symbolique ou numérique pouvant être prises en compte par le système afin de fournir des définitions plus précises et plus intelligibles. Finalement, une dernière partie présente les résultats obtenus sur différentes bases d'apprentissage de référence.

4.1 Architecture générale

L'apprentissage de formules relationnelles est un problème de recherche dans un espace dont la taille est gigantesque et dont la structure, bien qu'ordonnée, comporte de nombreux extremums locaux. De plus, ce problème est multi-critères (comme le montre la section 4.2, de nombreux traits, parfois antagonistes, doivent être optimisés), et les données sont souvent bruitées. Les algorithmes d'évolution sont donc de bons candidats pour ce type de problème. La stratégie de recherche de $\mathcal{SIAO}1$ est fondée sur deux principes :

1. L'utilisation d'une méthode stochastique, dont la pertinence est justifiée par la taille, la structure et les propriétés de l'espace de recherche.

2. L'exploitation de diverses informations relatives au problème, qu'il s'agisse des exemples d'apprentissage ou de la relation d'ordre $\vdash_2$ décrite au chapitre 3.

Le paragraphe suivant présente la méthode de couverture simultanée des exemples qui est utilisée ; cette dernière est une évolution de la méthode de couverture itérative pro-

posée dans le cadre des algorithmes d'évolution par G. Venturini pour le système SIA [Venturini94b].

4.1.1 Méthode de recherche

$\mathcal{SIAO}1$ apprend un nombre variable (borné par l'utilisateur) de règles décrivant un ou plusieurs concepts de la manière suivante :

Recherche unidirectionnelle et ascendante

Soit Ex un exemple de l'un des concepts-cible cl, déterminé aléatoirement suivant une probabilité de sélection proportionnelle au poids des instances. Le noyau $N(Ex)$ est construit en remplaçant dans la formule Ex toutes les occurrences du symbole `?` par le symbole `*`[1]. $N(Ex)$ est appelé noyau générateur de la population $P(Ex)$ puisque toutes les formules de $P(Ex)$ seront plus générales que $N(Ex)$ (au sens de la relation d'ordre $\vdash_2$). En pratique, la propriété de couverture du noyau est garantie par le fait que les opérateurs d'évolution sont ascendants : l'espace de recherche est un treillis ordonné par $\vdash_2$ dont le plus petit élément est $N(Ex)$ et le plus grand élément est la clause $cl \leftarrow .$ dont le corps est vide. L'évolution de $P(Ex)$ est décrite par l'algorithme ci-dessous :

`/* Algorithme d'évolution d'une population` $P(Ex)$ `, partie 1/1 */`

1. $P(Ex) \leftarrow \{N(Ex)\}$; $p_m \leftarrow 1/2$
2. Répéter
   (a) Générer un individu r' à partir d'un individu r sélectionné proportionnellement à sa qualité :
      i. par mutation avec la probabilité p_m : $r \rightarrow r'$;
      ii. par croisement avec la probabilité $p_c = 1 - p_m$: $r, r_2 \rightarrow r'$. Le second parent r_2 est soit un autre individu de $P(Ex)$, soit une instance de la base d'apprentissage de même classe que Ex.
   (b) Insérer r' dans $P(Ex)$ si $r' \notin P(Ex)$:
      - si la taille maximum de la population (notée P_{max}) n'est pas atteinte, alors ajouter r' à $P(Ex)$;
      - si la taille maximum de la population est atteinte et si la qualité de r' est supérieure à la qualité du pire individu de $P(Ex)$, alors remplacer cet individu par r'.
   (c) Mettre à jour p_m.
3. Jusqu'à ce que le critère d'arrêt soit vérifié ou qu'un message de réinitialisation soit reçu.

[1] Il n'est en effet pas concevable qu'une description générale de concept puisse faire explicitement référence à une valeur définie mais inconnue relative à une instance bien particulière.

Le schéma d'évolution d'une population de $\mathcal{SIAO}1$ est caractérisé par les propriétés suivantes, dont les deux dernières proviennent du système SIA :

- *Une direction de recherche ascendante.* Cette dernière est garantie par les opérateurs d'évolution construits pour fournir une hypothyèse plus générale au sens de $\vdash_2$ que la règle `r` à laquelle ils s'appliquent, et permet une recherche efficace puisqu'exempte de retours-arrière et de cycles.

- *Une sélection incrémentale et élitiste.* Conformément au principe de recherche, le schéma de sélection choisi permet une exploration stable et progressive du treillis puisque, contrairement aux schémas générationnels, la sélection incrémentale et élitiste assure la conservation des meilleurs individus. D'autre part, le test d'insertion maintient la diversité de la population qui correspond à une liste d'individus distincts triés en fonction de leur qualité. La valeur par défaut de la taille maximum d'une population est de $P_{max} = 50$ individus.

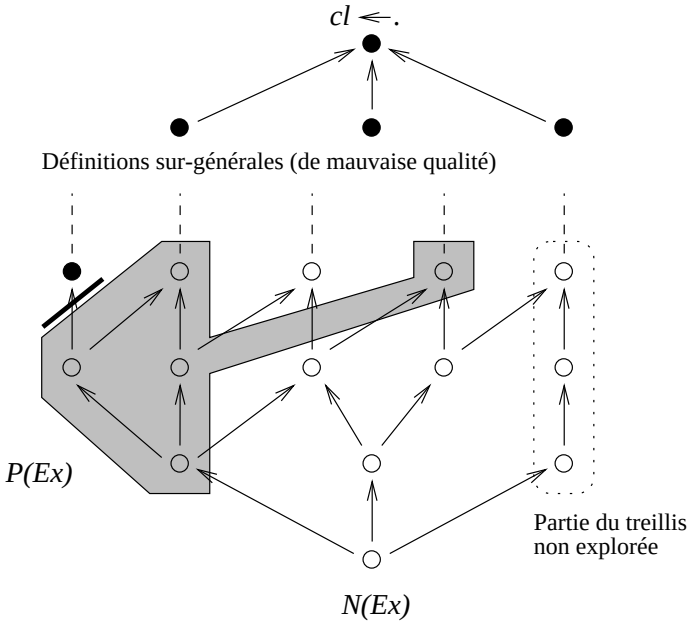


FIG. 4.1: Évolution de $P(Ex)$ dans le treillis défini par $N(Ex)$ et $\vdash_2$.

La figure 4.1 illustre le processus de recherche : les éléments de $P(Ex)$ sont des nœuds distincts du treillis générés parmi les successeurs[2] d'un élément de $P(Ex)$. Les nœuds sur-généraux (qui figurent en noir sur le schéma) ne peuvent être insérés dans la population car leur qualité (fondée principalement sur leur cohérence), est alors très faible. Le processus d'évolution se termine lorsque $P(Ex)$ se situe près de la frontière séparant les hypothèses maximalement correctes des hypothèses sur-générales : il n'est plus possible de progresser dans le treillis et aucun retour en arrière ne peut être effectué puisque les opérateurs sont ascendants. La section 4.2, qui détaille le calcul de la qualité des individus,

[2]Il peut s'agir d'un successeur immédiat ou d'un $n^{ième}$ successeur.

montre comment il est possible de définir une fonction fondée avant tout sur la cohérence, favorisant les hypothèses générales, et tolérante au bruit. La stochasticité du processus combinée à la taille du treillis ne permet pas d'assurer une exploration exhaustive qui serait de toute manière bien trop coûteuse.

– *L'auto-adaptation des taux de mutation et de croisement.* Chaque population adapte dynamiquement le taux de mutation local $p_m \in [0,1;0,9]$ suivant la loi de relaxation :

$$p_m(g+1) = (1-\alpha).p_m(g) + \alpha.res$$

avec :

- $\alpha = \frac{1}{100}$;
- $res = \begin{cases} - & 0,9 \textit{ si la dernière opération est une mutation (resp. un croisement), et l'individu généré a pu (resp. n'a pas pu) être inséré dans la population} ; \\ - & 0,1 \textit{ sinon.} \end{cases}$

Les mécanismes d'auto-adaptation constituent un champ de recherche très actif dans le domaine des algorithmes évolutionnaires [Sebag et al.97b, Ravise et al.96, Spears91, Davis89]. En ce qui concerne $\mathcal{SIAO}1$, les bénéfices sont nombreux puisque :

- l'utilisateur n'est pas sollicité pour ajuster les paramètres ;
- ces derniers s'adaptent au type de problème rencontré ;
- de plus, ils s'adaptent également pendant l'évolution.

– *Une diversité maximale des individus de la population.* La multiplication d'individus identiques au sein de la population est justifiée dans le contexte de l'algorithme génétique canonique par l'argument du bandit manchot à k bras (mentionné chapitre 2 section 2.1.2), qui démontre la validité de cette stratégie lorsque l'objectif de l'algorithme est d'allouer ses ressources (les individus) de manière à maximiser le cumul des gains (retournés par la fonction de qualité). Cependant, le contexte de $\mathcal{SIAO}1$ est sensiblement différent puisque l'objectif n'est pas de maximiser une somme, mais de trouver en un temps raisonnable une valeur aussi proche du maximum que possible. Le maintien de la diversité des individus au sein de la population, hérité du système SIA, est donc justifié puisqu'il permet un meilleur échantillonnage de l'espace de recherche et diminue d'autant le risque de convergence prématurée.

– *L'assurance de la terminaison de l'algorithme.* Ce résultat, qui provient du système SIA, est partagé par tous les systèmes d'apprentissage évolutionnaires. Cependant, alors qu'il est habituellement garanti par la spécification d'une borne au nombre maximum de générations, il dépend dans le cas de $\mathcal{SIAO}1$ (comme dans le cas de SIA) du critère d'arrêt :

Si aucun individu n'a été inséré depuis g_{max} générations,
alors l'évolution de la population est terminée.

Le paramètre g_{max} permet de régler l'intensité de la recherche, dont la valeur absolue (en nombre de générations) est adaptée à l'environnement du noyau.

4.1.2 Développement simultané de plusieurs noyaux

Hormis le problème de la recherche d'une hypothèse couvrant un ensemble d'exemples, une autre source de complexité pour la tâche d'apprentissage provient des concepts disjonctifs. Ces derniers ne peuvent être caractérisés que par un ensemble d'hypothèses : le système d'apprentissage doit alors s'efforcer de minimiser leur nombre sous les contraintes de cohérence et de complétude étendues à la totalité du modèle. Les solutions apportées à ce problème peuvent être groupées en deux grandes catégories :

- Les stratégies locales, qui apprennent les définitions l'une après l'autre. La méthodologie de l'étoile, détaillée chapitre 1 section 1.3.4, est un exemple de stratégie appartenant à cette catégorie.

- Les stratégies globales, qui isolent simultanément les sous-concepts. Le principe diviser pour régner (chapitre 1 section 1.3.5), l'algorithme des Vraizamis [Torre99] ou, dans le domaine des algorithmes d'évolution, la sélection au suffrage universel (chapitre 2 section 2.2.4) sont autant d'exemples de méthodes globales.

Dans le cas de $\mathcal{SIAO}1$, la stratégie employée consiste à développer simultanément plusieurs noyaux. Un nombre maximum de définitions N_{cl} est fourni par l'utilisateur pour chacun des concepts-cible (par défaut ce nombre est fixé à 5 pour toutes les classes). Des noyaux tous différents de classe appropriée sont alors déterminés afin d'initialiser chaque population. Celles-ci évoluent indépendamment de manière synchrone conformément à l'algorithme donné section 4.1.1. À intervalles réguliers (par défaut tous les $N_{cycle} = 100$ générations), un processus d'élagage est déclenché parmi les populations n'ayant pas encore convergé. Ce dernier consiste à tester le recouvrement du noyau N_i par la définition de meilleure qualité de toutes les populations $H_{max}(N_j), j \neq i$ concluant sur la même classe. Alors, si la qualité de $H_{max}(N_j)$ est supérieure à la qualité de $H_{max}(N_i)$, la $i^{ème}$ population est abandonnée et ré-initialisée. Le nouveau noyau est choisi par tirage aléatoire pondéré par les poids des instances parmi les exemples de classe cl non couverts par les définitions ayant convergé.
La figure 4.2 illustre le fonctionnement du processus d'élagage dynamique (les deux noyaux représentés concluent sur la même classe) :

- Dans le premier cas, le noyau N_1 sera très probablement abandonné car d'une part il est couvert par la meilleure hypothèse issue du noyau N_2, et d'autre part la couverture de $H_{max}(N_2)$ étant supérieure à celle de $H_{max}(N_1)$, la qualité de $H_{max}(N_2)$ est très probablement supérieure à celle de $H_{max}(N_1)$. Par contre, le noyau N_2 ne peut être abandonné car $H_{max}(N_1)$ ne couvre pas N_2

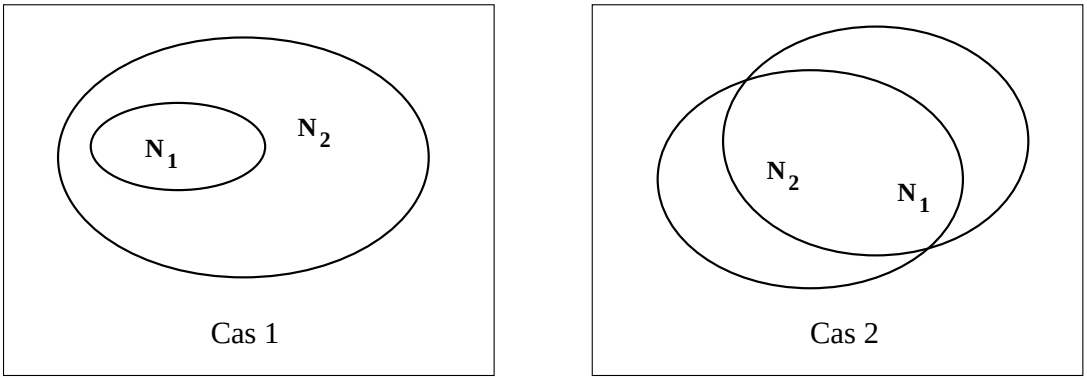


FIG. 4.2: Processus d'élagage au cours de l'évolution.

– Dans le second cas, il est certain que l'un des deux noyaux sera abandonné puisque chacun est couvert par la meilleure hypothèse issue de l'autre. Les couvertures de $H_{max}(N_1)$ et $H_{max}(N_1)$ étant voisines, l'élagage sera principalement fonction de la cohérence des meilleures hypothèses de chaque noyau.

4.1.3 Représentation des formules

Une des particularités des approches évolutionnaires appliquées à l'apprentissage relationnel est le caractère diamétralement opposé des deux méthodes de représentation mises en œuvre.

La première méthode, employée par les systèmes fondés sur les algorithmes génétiques [Angelano et al.97, TN et al.00], consiste à représenter les formules relationnelles sous la forme de chaînes binaires de taille fixe : il est donc nécessaire de fournir un modèle de la structure relationnelle de la solution. En pratique, le modèle peut :

– Être fourni par l'utilisateur. Toutefois, bien qu'il soit commode de reporter sur ce dernier un choix particulièrement difficile, cette méthode fait l'hypothèse que l'utilisateur puisse fournir une forme approchée de la solution. Ce dernier se trouve alors face à un dilemme : soit il choisit un modèle très expressif au risque d'un surcoût de calcul et d'une dégradation possible de la qualité de la solution dûe à la taille de l'espace de recherche, soit il spécifie (lorsque cela est possible) une forme moins expressive et plus précise, au risque de manquer la forme optimale en cas d'erreur.

– Provenir de la structure d'un exemple. Cette alternative, qui a le double avantage d'être simple et de ne pas solliciter l'utilisateur, a néanmoins l'inconvénient de négliger de nombreuses formes potentiellement utiles. En effet, comme le montre sur un exemple la figure 4.13 de la section 4.3.2, la plus petite forme couvrant deux exemples peut n'être incluse dans aucune des formes de ces exemples.

– Être construit à partir de la base d'apprentissage. Cependant, en examinant la combinatoire d'une telle méthode, il semble difficile d'automatiser le processus de con-

struction sans contrainte d'aucune sorte. En effet, un modèle généré à partir de n exemples de taille k peut atteindre une taille de k^n, ce qui a par exemple motivé la mise en place de biais de représentation spécifiques dans le cas du système de PLI GOLEM (voir chapitre 1, section 1.5.2).

L'utilisation d'un modèle est donc une méthode qui peut être critiquée. En effet, comme le montrent les trois approches détaillées ci-dessus, la structure de la solution est, dans le cas général, une composante du problème de l'apprentissage relationnel qu'il est particulièrement difficile de résoudre séparément. De plus, les chaînes binaires de taille fixe ont l'inconvénient d'être très peu adaptées aux problèmes relationnels mettant en jeu des valeurs numériques.

La seconde méthode provient des algorithmes évolutionnaires fondés sur la programmation génétique [Wong et al.95]. Ces derniers fournissent une alternative d'autant plus intéressante que les structures arborescentes suppriment la nécessité de spécifier un modèle. En outre, cette approche a été validée à la fois sur des problèmes-jouets relationnels tels que les arches de Winston [Winston75], ainsi que sur des problèmes récursifs comme la définition de la fonction factorielle. Cependant, un certain nombre d'inconvénients doivent être considérés avant d'envisager l'application de telles techniques à des problèmes réels :

- le coût de calcul est important, et ce même lorsque le problème traité est de taille modeste et non bruité ;
- la technique d'exploration semble peu adaptée aux spécificités des problèmes de PLI[3]. Par exemple, le croisement est un opérateur hautement disruptif pour les chaînes relationnelles.

L'approche suivie par $\mathcal{SIAO}1$, qui peut être qualifiée de médiane par rapport aux deux autres méthodes, a été définie en répondant aux questions suivantes :

1. Quelle est la plus petite unité d'information au sein d'une formule dont la modification pourrait avoir un sens du point de vue d'un algorithme de recherche de PLI travaillant à l'aide d'opérateurs de haut niveau ?
2. Quels sont les blocs d'information dont la combinaison constitue une solution ? Quelle est la structure la plus simple permettant la représentation d'une solution à l'aide de ces blocs ?

Du point de vue de la conception d'un algorithme évolutionnaire, la première question isole le concept de gène (unité sur laquelle opère la mutation), et la deuxième précise la structure du chromosome ainsi que l'opérateur de croisement. Au vu du langage $\mathcal{L}_h$, les réponses apportées sont les suivantes :

[3]Ce sentiment est renforcé par le fait que la définition du concept d'arche est trouvée à la seconde génération, et la définition de la factorielle à la quinzième (le nombre maximum de générations étant fixé à 20). En outre, comme la taille des populations est fixée à 1000 individus, il est difficile de déduire de ces résultats l'adéquation des opérateurs à un processus fondé sur l'évolution.

1. Un gène est une unité syntaxique de base, c'est-à-dire soit un symbole de prédicat, soit un argument. Les opérations envisagées pour la mutation sont en effet des règles de généralisation fondées soit sur la synatxe, comme $Atome(X, c) \xrightarrow{\mu} Atome(X, \{c \vee h\})$, soit sur la théorie du domaine, comme $Voiture(X) \xrightarrow{\mu} Véhicule(X)$.

2. Un bloc de construction est un atome. En effet, définir un point de coupure entre deux gènes quelconques d'un chromosome peut produire un résultat non conforme aux signatures des prédicats. De même que pour le système GABIL (voir chapitre 2, section 2.2.2), il est nécessaire de déterminer des points de coupure sémantiquement compatibles . Dans le cas de $\mathcal{L}_h$, une solution consisterait à choisir ces derniers avant un gène représentant un symbole de prédicat.

Ainsi :

– Un gène est une unité syntaxique possédant une forme, un type et un nombre variable de valeurs. Les différentes formes, définies d'après $\mathcal{L}_h$, sont énumérées dans le tableau ci-dessous[4] :

Forme	Type	Valeurs
Symbole de prédicat	—	1
Symbole de variable	Num / Symb	1
Symbole de constante	Num / Symb	1
Disjonction	Num / Symb	> 1
Intervalle numérique	Num	2
Variable étiquetée	Num / Symb	2

TAB. 4.1: Les différentes formes d'un gène.

– Un chromosome est une liste (de taille variable) de gènes respectant les contraintes relatives aux arités, aux types et aux modes définis sur $\mathcal{L}_h$. La figure 4.3 détaille la structure du chromosome représentant la formule $Atome(X, \{c \vee h\})$.

Chromosome	Forme	Type		Valeurs	
gène n° 1	*Prédicat*	—	→	*Atome*	
gène n° 2	*Variable*	`Id_a`	→	*X*	
gène n° 3	*Disjonction*	`Élément`	→	*c*	*h*

FIG. 4.3: Structure d'un chromosome.

[4]La dernière forme n'est présente qu'au cours de calculs intermédiaires relatifs au développement du moindre généralisé.

4.2 Calcul de la qualité

La fonction de qualité a pour objectif d'évaluer les individus générés lors de la recherche. Cette fonction doit non seulement prendre en compte de multiples critères d'appréciation parfois antagonistes, mais elle doit également être adaptée au type de la recherche afin de guider l'algorithme en favorisant les individus susceptibles de conduire à la solution. Dans le cas de $\mathcal{SIAO}1$, cette fonction est constituée de deux composantes.

La première composante – prépondérante – est uniquement fondée sur la couverture de la règle à évaluer. Comme indiqué au chapitre 1, figure 1.5 et table 1.6, la couverture peut être décrite à l'aide des effectifs $n_{\mathcal{C}\wedge\mathcal{H}}$, $n_{\bar{\mathcal{C}}\wedge\mathcal{H}}$, $n_{\mathcal{C}\wedge\bar{\mathcal{H}}}$ et $n_{\bar{\mathcal{C}}\wedge\bar{\mathcal{H}}}$ du tableau de contingence.

La seconde composante reflète un biais fondé sur la syntaxe de l'individu. Le but poursuivi est de permettre, à critère de couverture égal, l'insertion dans la population des descriptions fournissant le meilleur compromis entre généralité et concision.

Les paragraphes suivants sont consacrés à la description et à l'analyse de ces deux composantes, ainsi qu'à quelques considérations concernant la complexité de la fonction de qualité ainsi que l'optimisation de son calcul.

Une dernière composante, correspondant à un terme correctif permettant la prise en compte des préférences de l'utilisateur a été proposée dès [Augier et al.95]. L'intérêt en est double : d'une part, le fait de permettre à l'utilisateur de spécifier des biais spécifiques à un problème donné, et donc de guider la recherche peut présenter un intérêt pratique dans le cadre de certaines applications très particulières pour lesquelles il dispose de connaissances qu'il souhaite exploiter ; d'autre part ce terme correctif peut également permettre à l'utilisateur de spécifier ses propres critères d'intelligibilité. Cependant, cette composante ne présente pas d'intérêt dans un contexte expérimental classique, surtout lorsqu'il s'agit d'évaluer le comportement de l'algorithme dans un but de comparaison et sur des problèmes dont les caractéristiques sont parfois connues. Par conséquent, elle n'a pas été utilisée lors des expérimentations.

4.2.1 L'estimateur de Laplace

Comme l'ont montré les chapitres 1 et 2, de très nombreux critères ont été proposés pour évaluer la couverture d'une règle sur les exemples. Ce paragraphe détaille l'estimateur de Laplace, qui fournit une approximation plus fiable d'une fréquence que le résultat obtenu par simple comptage, et est appliqué dans le cadre de $\mathcal{SIAO}1$ à l'estimation de la cohérence de la règle à évaluer. L'utilisation de cet estimateur repose sur des arguments théoriques aussi bien que pratiques. Du point de vue théorique, il est possible de justifier son emploi de par sa construction [Jaynes96, Kodratoff98].

Justification théorique de l'estimateur de Laplace

Soit f_r la fréquence réelle et f_e son estimation, l'objectif étant de minimiser l'erreur vue comme le carré de l'écart entre ces deux valeurs, soit $(f_r - f_e)^2$. Ne connaissant pas f_r, il est néanmoins possible de reformuler le problème en considérant la densité de probabilité, notée $P(f_r)$ et définie sur $f_r \in [0,1]$. Le problème de minimisation de l'erreur revient donc à trouver f_e minimisant l'erreur moyenne donnée par l'expression :

$$Err(f_e) = E((f_r - f_e)^2) = \int_0^1 (f_r - f_e)^2 . P(f_r)\ df_r \quad \textbf{(Lapl 1)}$$

Cette fonction de f_e étant continue et dérivable sur $\Re \supset [0,1]$, le f_e minimum ne peut être trouvé qu'aux bornes (ce qui est impossible puisque $lim_{f_e \to +\infty} Err(f_e) = lim_{f_e \to -\infty} Err(f_e) = +\infty$) ou au(x) point(s) vérifiant $\frac{d(Err(f_e))}{df_e} = 0$. Or,

$$\begin{aligned} Err(f_e) &= \int_0^1 (f_r^2 - 2.f_r.f_e + f_e^2).P(f_r)\ df_r \\ &= \int_0^1 f_r^2.P(f_r)\ df_r - 2.f_e.\int_0^1 f_r.P(f_r)\ df_r + f_e^2.\underbrace{\int_0^1 P(f_r)\ df_r}_{=1} \end{aligned}$$

D'où :

$$\begin{aligned} \frac{d(Err(f_e))}{df_e} &= \underbrace{\frac{d}{df_e}\left[\int_0^1 f_r^2.P(f_r)\ df_r\right]}_{=0} + \frac{d}{df_e}\left[-2.f_e.\int_0^1 f_r.P(f_r)\ df_r\right] + \frac{d}{df_e}\left[f_e^2\right] \\ &= -2.\int_0^1 f_r.P(f_r)\ df_r + 2.f_e \end{aligned}$$

Ainsi, résoudre $\frac{d(Err(f_e))}{df_e} = 0$ revient à chercher f_e telle que :

$$f_e = \int_0^1 f_r.P(f_r)\ df_r \quad \textbf{(Lapl 2)}$$

La densité de probabilité $P(f_r)$ est donnée dans le cas d'un problème de classification binaire par la formule suivante, attribuée à Bayes et mentionnée dans [Munteanu96] :

$$p(f) = \frac{(p+n+1)!}{p!n!}.f^p.(1-f)^n \quad (\mathcal{D})$$

p et n sont les effectifs des exemples positifs et négatifs. De plus, la formule précédente décrivant une distribution de probabilité, on a :

$$\int_0^1 \frac{(p+n+1)!}{p!n!}.f^p.(1-f)^n.df = 1;\ \Rightarrow\ \int_0^1 f^p.(1-f)^n.df = \frac{p!n!}{(p+n+1)!}$$

En posant $p = k+1$, on obtient :

$$\int_0^1 f^{k+1}.(1-f)^n.df = \frac{(k+1)!n!}{(k+n+2)!} \quad \textbf{(Lapl 3)}$$

Il est donc possible de résoudre l'équation **(Lapl 2)** :

$$f_e = \int_0^1 f_r.P(f_r)\,df_r = \int_0^1 f_r.\left[\frac{(p+n+1)!}{p!n!}.f_r^p.(1-f_r)^n\right]\,df_r = \frac{(p+n+1)!}{p!n!}\int_0^1 f_r^{p+1}.(1-f_r)^n\,df_r$$

En utilisant **(Lapl 3)**, on obtient :

$$f_e = \frac{(p+n+1)!}{p!n!}.\frac{(p+1)!n!}{(p+n+2)!} = \frac{p+1}{p+n+2}$$

Ce résultat peut être généralisé pour un nombre quelconque k de classes :

$$f_e = Estimateur\ de\ Laplace\ (p,n,k) = \frac{p+1}{p+n+k}$$

Intérêt pratique de l'estimateur de Laplace

Un autre résultat remarquable peut être obtenu à partir de la densité de probabilité $(\mathcal{D})$. Si, au lieu de minimiser l'erreur quadratique de l'estimation de la fréquence **(Lapl 1)**, on se contente de choisir la valeur réalisant le maximum de cette densité de probabilité, on obtient le risque empirique f_{max}[5] :

$$\frac{d}{df}\left[p(f)\right] = 0 \Rightarrow \frac{d}{df}\left[\frac{(p+n+1)!}{p!n!}.f^p.(1-f)^n\right] = 0 \Rightarrow \underbrace{\frac{(p+n+1)!}{p!n!}}_{\neq 0}.\frac{d}{df}\left[f^p.(1-f)^n\right] = 0$$

$$\Rightarrow p.f^{p-1}.(1-f)^n - f^p.n.(1-f)^{n-1} = 0 \Rightarrow p.(1-f) = n.f \Rightarrow p - f.p = n.f \Rightarrow p = f.(n+p)$$

Finalement, l'évaluation du risque empirique donne : $f_{max} = \frac{p}{n+p} = cohérence(n,p)$.

Ce résultat montre que la définition classique de la cohérence (telle qu'elle est observée sur un ensemble d'apprentissage) n'est qu'une première approximation de la fréquence recherchée, et que l'estimation de cette fréquence basée sur un critère de minimisation d'erreur met en jeu, contrairement à la première approximation, le nombre k de classes.

En pratique, l'utilisation de l'estimateur de Laplace pour l'évaluation des règles de $\mathcal{SIAO}1$ s'est révélée tout à fait positive, et ce pour deux raisons :

Prise en compte de la représentativité statistique

Un premier avantage de l'estimateur de Laplace est que contrairement à la cohérence, il est sensible aux effectifs (donc à la couverture) de la règle considérée. Soient par exemple deux règles R_1 et R_2, couvrant respectivement $(p_1 = 2, n_1 = 1)$ et $(p_2 = 20, n_2 = 10)$ exemples positifs et négatifs. La cohérence ne permet pas de les distinguer, puisque $f_{max}(R_1) = f_{max}(R_2) = \frac{2}{3}$, alors que l'estimateur de Laplace fournira un meilleur score à la règle dont la couverture est plus importante : $f_e(R_1) = \frac{3}{5} = 0,6$, et $f_e(R_2) = \frac{21}{32} = 0,65$ (voir figure 4.4).

En ce qui concerne $\mathcal{SIAO}1$, cette propriété fait de l'estimateur de Laplace un excellent compromis. En effet :

- D'une part, la cohérence observée f_{max} de la règle est une caractéristique critique, puisque l'algorithme d'apprentissage procédant par généralisations successives, f_{max} ne peut que décroître, et il faut donc immédiatement pénaliser fortement toute règle peu cohérente. Cette première contrainte est bien sûr prise en compte puisque l'estimateur de Laplace estime une fréquence qui correspond à la cohérence. Ce dernier est d'ailleurs très proche du point de vue de son expression comme de son comportement de f_{max}.

[5] $(\mathcal{D})$ est bien sûr continue et dérivable sur $[0,1]$; l'étude de la dérivée (dont le calcul est donné ci-après) montre que $\mathcal{D}$ est croissante de 0 jusqu'à un point f_{max} puis décroissante de f_{max} à 1.

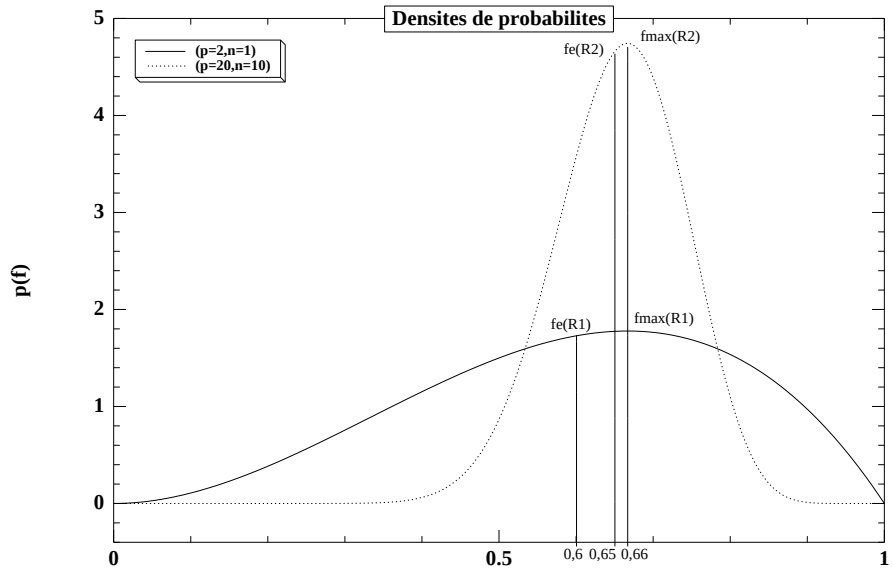


FIG. 4.4: Comparaison entre f_e et f_{max} pour R_1 et R_2.

– D'autre part, il faut également être capable de prendre en compte la couverture de la règle afin de favoriser le développement du noyau. Comme l'a montré l'exemple précédent, seul f_e prend en compte cet aspect, alors que f_{max} l'ignore.

Robustesse par rapport au bruit

Le second avantage de l'estimateur de Laplace est qu'en corollaire à sa prise en compte de la représentativité statistique, il est moins sensible au bruit que la cohérence : soient de nouveau R_1 et R_2, couvrant cette fois respectivement $(p_1 = 10, n_1 = 0)$ et $(p_2 = 99, n_2 = 1)$ exemples positifs et négatifs. Si la cohérence donne une préférence à la première règle ($f_{max}(R_1) = 1$, $f_{max}(R_2) = 0,99$), ce sera l'inverse dans le cas de l'estimateur de Laplace ($f_e(R_1) = 0,92$, $f_e(R_2) = 0,98$).

Conclusion sur l'estimateur de Laplace

L'estimateur de Laplace, qui est utilisé ici afin de mesurer la cohérence des règles apprises, s'est révélé parfaitement adapté de par sa simplicité de calcul (aucun paramètre ne doit être ajusté par l'utilisateur), sa capacité à résister au bruit et sa prise en compte de la couverture des règles par le biais de sa sensibilité aux effectifs mesurés. De plus, de par sa construction, il fournit une combinaison élégante et mathématiquement justifiée des notions de cohérence et de complétude.

4.2.2 Le biais syntaxique

Il peut arriver que la couverture mesurée sur la base d'apprentissage ne soit pas suffisante pour guider le processus d'induction. En effet, le nombre d'exemples étant le plus souvent très petit au regard de la taille de l'espace de recherche, un grand nombre d'hypothèses auront des couvertures identiques. En particulier, il n'est pas rare que de nombreux descendants d'un individu (c'est-à-dire un grand nombre de ses généralisations) aient la même qualité. Or, dans le cas où l'algorithme de recherche est confronté à un plateau, c'est-à-dire à une région de l'espace comportant de nombreuses hypothèses de couvertures équivalentes, il est possible que la population ne contienne plus que des individus ayant tous la même qualité. Si de plus aucune des hypothèses au voisinage de ces individus ne permet une amélioration de la qualité, la recherche est bloquée puisqu'aucune insertion d'individu dans la population n'est possible. Ce phénomène a motivé la prise en compte par la fonction de qualité d'un autre type d'information basé sur l'aspect syntaxique de la formule codant l'hypothèse.

La généralité syntaxique

Une solution au problème exposé ci-dessus a été apportée sous la forme d'une fonction à valeur dans $]0, 1]$ quantifiant la généralité syntaxique d'une définition [Augier et al.95]. Cette fonction était par construction croissante pour toute modification syntaxique conduisant à une généralisation (abandon de prédicat, création ou élargissement d'une disjonction, etc.), et son intégration se fit de manière à garantir qu'elle ne puisse supplanter le critère de couverture en la pondérant par un coefficient suffisamment petit. Intuitivement, l'ajout de cette fonction peut se concevoir comme une modification de la topographie du paysage de la fonction de qualité substituant aux plateaux parfaitement aplatis des zones légèrement orientées de telle sorte que l'algorithme soit guidé vers les bords des plateaux correspondant aux formules les plus générales.

Le principe de concision

Cependant, s'il semble évident de chercher à favoriser dans le cadre de $\mathcal{SIAO}1$ les définitions les plus générales, il est également important de prendre en compte, dans le biais syntaxique d'évaluation des règles, d'autres préférences partiellement contradictoires avec le principe de généralité énoncé ci-dessus. Ainsi l'argument du rasoir d'Occam (expliqué chapitre 1 section 1.4.3), qui conduit à préférer – toutes choses étant égales par ailleurs – les explications les plus courtes est apparu très pertinent dès les expériences préliminaires puisque l'utilisation de disjonctions internes dans le langage des prédicats annotés favorise l'apprentissage par cœur dans les bases dont le nombre de symboles est important par rapport au nombre d'exemples. Il paraît donc nécessaire de favoriser les définitions concises, et en particulier de pénaliser l'ajout d'éléments dans les disjonctions lorsque ces derniers n'apportent aucun gain de couverture.

Définition

La composante correspondant au biais syntaxique est prise en compte au moyen de la fonction BS définie ci-dessous. Cette fonction d'une hypothèse H et à valeur dans $[0,1]$ réalise un compromis entre les critères syntaxiques de généralité et de concision :

$$BS(H) = 1 \; - \; \frac{1}{L_{max}} \sum_{i=1}^{\delta} \sum_{j=1}^{|D_i|} P(D_i(j)) \quad \textbf{(BS 1)}$$

Les notations sont les suivantes :

- H est l'hypothèse considérée, composée de δ atomes : $H = D_1 \wedge ... \wedge D_\delta$.
- L_{max} correspond à la longueur maximale des chromosomes de la base d'apprentissage (il s'agit donc également du nombre maximum de gènes de toute hypothèse).
- Chaque atome D_i de H est d'arité $|D_i|$; $D_i(k)$, avec $1 \leq k \leq |D_i|$, dénote alors le $k^{ième}$ argument de l'atome D_i.
- $D_i(0)$ fait référence au symbole de prédicat de l'atome.

Type du gène		Valeur de P
Symbole de prédicat	p	$\frac{prof_*(p)+1}{\|\|p\|\|}$
Étoile numérique ou symbolique	$*$	0
Constante symbolique	c	$\frac{prof_*(c)+1}{\|\|c\|\|.disj_{max}}$
Constante numérique	n	$\frac{1}{disj_{max}}$
Disjonction symbolique	$\{c_1 \vee ... \vee c_k\}$	$\frac{1}{\|\|c_1\|\|.disj_{max}} . \sum_{i=1}^{k}(prof_*(c_i)+1)$
Disjonction numérique	$\{n_1 \vee ... \vee n_l\}$	$\frac{l}{disj_{max}}$
Intervalle numérique	$[n_1...n_2]$	$\frac{2}{disj_{max}.lg(n_1,n_2)}$
Variable symbolique	X	$\frac{1}{\|\|X\|\|.disj_{max}}$ ou 0 [a]
Variable numérique	Y	$\frac{1}{\|\|Y\|\|.disj_{max}}$ ou 0 [a]

[a] La pénalité de complexité n'est appliquée qu'à chaque *nouveau* symbole de variable rencontré.

TAB. 4.2: Fonction de pénalité définie sur les gènes.

- La fonction P attribue une pénalité dans [0..1] à tout gène, comme décrit dans le tableau 4.2. $prof_*(x)$ dénote le nombre de symboles strictement supérieurs à x par la hiérarchie définie sur son type ; $||x||$ le nombre total de symboles de même type que x ; $disj_{max}$ correspond au nombre maximum d'éléments que peut comporter une disjonction et $lg(n_1, n_2)$ au nombre de constantes numériques comprises dans l'intervalle $[n_1, n_2]$.

Propriétés

De par sa définition d'une part, et la pondération de la fonction de pénalité P d'autre part, la formule (**BS 1**) favorisera par ordre de préférence les traits suivants :

Au niveau des prédicats

1. L'absence de symbole de prédicat.

2. La présence d'un symbole de prédicat aussi général que possible d'après la hiérarchie définie sur les prédicats qui provient de la théorie du domaine.

Au niveau des types numériques

1. La présence d'une étoile $*$, ce symbole ayant la propriété de s'apparier avec tout symbole du même type sans propager quelque contrainte que ce soit.

2. La présence d'un symbole de variable, qui contrairement à l'étoile peut correspondre à l'ajout implicite d'une contrainte d'égalité dans le cas où au moins un autre symbole identique figure dans l'hypothèse. Afin de favoriser l'intelligibilité de la formule (au détriment de sa généralité), la pénalité appliquée avantage les hypothèses minimisant le nombre de symboles de variable différents.

3. La présence d'un intervalle numérique aussi large que possible. Comme la notion de largeur d'un intervalle n'a pas de sens dans l'absolu, elle est mesurée ici par rapport aux données comme le nombre de constantes numériques du type considéré couvertes par cet intervalle.

4. La présence d'une constante.

5. La présence d'une disjonction contenant aussi peu d'éléments que possible. Conformément au critère du rasoir d'Occam, la concision des hypothèses est ici préférée à leur généralité puisque les disjonctions sont les structures les plus pénalisées.

Au niveau des types symboliques

1. La présence d'une étoile $*$.

2. La présence d'un symbole de variable.

3. La présence d'une constante aussi générale que possible d'après la hiérarchie définie sur le type considéré qui provient de la théorie du domaine.

4. La présence d'une disjonction comportant aussi peu d'éléments que possible, ces derniers étant aussi généraux que possible.

Bien évidemment, l'additivité des pénalités provenant de (**BS 1**) implique que l'ordre des préférences n'est pas absolu. Par exemple, il est possible qu'une hypothèse combinant trois traits moyennement pénalisés obtienne un meilleur score qu'une hypothèse combinant un trait privilégié et deux traits pénalisés. De même, il est possible qu'une disjonction contenant un petit nombre de symboles très généraux (au sens de la hiérarchie) soit préférée à une constante très spécifique (au sens de cette même hiérarchie).

Intégration du biais syntaxique dans la fonction de qualité

Le biais syntaxique étant destiné à permettre une comparaison plus fine des hypothèses sans pour autant remettre en question le critère de couverture (qui est primordial), la fonction de qualité est définie par :

$$Q(H) = \underbrace{f_e(H)}_{\substack{estimateur \\ de\ Laplace}} + \alpha \ . \ \underbrace{BS(H)}_{\substack{biais \\ syntaxique}}$$

en choisissant un coefficient α suffisamment petit pour que les variations de $f_e(H)$ ne puissent être couvertes par les variations de $BS(H)$.

Conclusion sur le biais syntaxique

La plupart des algorithmes d'apprentissage supervisé fournissant un modèle compréhensible par l'utilisateur intègrent, en sus des critères classiques de couverture, des biais liés à la concision ou à l'intelligibilité des règles. Ainsi, dans le cadre de l'apprentissage attribut-valeur basé sur les arbres de décision, l'algorithme C4.5 (qui fait référence dans ce domaine) intègre une mesure permettant la mise en œuvre du principe de description de longueur minimum (MDL, cf. [Quinlan93], chapitre 5, section 2, pages 51 et suivantes). Dans le cadre de l'apprentissage relationnel (et plus précisément de la programmation logique inductive), les systèmes FOIL et PROGOL (qui font référence dans ce domaine), utilisent respectivement les critères de description de longueur minimum (MDL, cf. [Quinlan et al.93]) et de compression maximale de l'information [Muggleton95]. Enfin, en ce qui concerne l'apprentissage relationnel basé sur les algorithmes génétiques, le système REGAL et son successeur G-Net prennent en compte la simplicité des règles apprises (cf. [Neri97], chapitre 11, section 1, pages 107-108).

En ce qui concerne $\mathcal{SIAO}1$, il a été possible d'observer expérimentalement que l'ajout du biais syntaxique a eu des répercussions sur l'intelligibilité des règles induites, cette dernière ayant parfois pu être notablement améliorée (en particulier en ce qui concerne les bases comportant peu d'exemples et fortement relationnelles comme *Mutagenesis*).

4.2.3 Réduction du coût de calcul

La plus grande partie du temps de calcul des algorithmes de type générer et tester est consacrée à l'évaluation des hypothèses construites, le coût d'application de l'opérateur de raffinement étant dans la plupart des cas négligeable. Le tableau 5.2 (cf. chapitre 5) montre qu'en pratique, dans le cas de $\mathcal{SIAO}1$, le temps consacré aux tests de subsomption dépasse 90% de la durée totale du fonctionnement de l'algorithme dès que la taille de la base d'apprentissage avoisine un millier d'exemples, et ce même sur les problèmes les moins complexes (qui ne sont pas relationnels). Dans le cas de problèmes fortement relationnels, il va de soi que cette proportion augmente plus rapidement.

Afin de réduire autant que possible cette durée, qui correspond au facteur limitant – du point de vue du temps d'exécution – de l'algorithme, plusieurs approches agissant à différents niveaux de l'architecture séquentielle peuvent être envisagées[6]. Ces approches sont énumérées ci-dessous en allant du niveau le plus interne au niveau le plus externe :

1. **Diminuer la durée de chaque test de subsomption**
   La procédure d'appariement entre les hypothèses générées et les exemples d'apprentissage pouvant être extrêmement complexe en logique relationnelle, il est nécessaire de pouvoir à la fois résoudre très rapidement les cas simples sans surcharge dûe à un pré-traitement trop lourd tout en réduisant autant que possible la combinatoire des instances plus difficiles. Le chapitre 3 détaille un algorithme performant permettant la résolution de ce problème.

2. **Diminuer la durée de chaque évaluation**
   En se basant sur la définition de la fonction de qualité, le coût de calcul de $Q(H)$ dépend des coûts des fonctions $f_e(H)$ et $BS(H)$. Cependant le coût de $BS(H)$ est tout-à-fait négligeable, alors que celui de $f_e(H)$, qui correspond à l'évaluation de la couverture de H sur les exemples d'apprentissage est (exprimé en nombre de tests de subsomption) égal au nombre d'exemples de la base d'apprentissage. Puisque le simple coût de ces tests représente habituellement plus de 90% de la durée totale du fonctionnement de l'algorithme, toute diminution de la durée des évaluations se basant sur une réduction du nombre des tests de subsomption nécessaires aura des répercussions notables. La méthode d'élagage proposée ci-après s'appuie sur ce principe.

3. **Diminuer le nombre d'évaluations**
   Une dernière possibilité consiste à réduire le nombre total d'évaluations effectuées par l'algorithme. Plutôt que de modifier la stratégie de recherche, ce qui risquerait de provoquer une convergence prématurée diminuant la qualité de la solution fournie, un système de mémorisation (partielle) des évaluations est proposé et analysé.

[6] Le chapitre 5 sera consacré aux approches moins immédiates consistant à modifier les données fournies à l'algorithme (techniques d'échantillonnage) ou à répartir l'implantation de ce dernier sur plusieurs machines.

Réduction de la durée d'une évaluation

Puisque la quasi-totalité du temps de calcul de $\mathcal{SIAO}1$ est consacrée aux tests de subsomption, une stratégie de réduction du nombre de tests a été implantée. Celle-ci met à profit la contrainte posée par le schéma de sélection élitiste spécifiant qu'un individu ne peut être inséré dans une population que si sa qualité est supérieure à la qualité minimale observée dans cette population. Plus généralement, elle peut être utilisée dans le cadre de tout algorithme de type « générer et tester » utilisant un seuil basé sur la couverture défini par une fonction $f(p, n)$ croissante sur p et décroissante sur n.

```
/* Test de couverture f_e(H) optimisé pour SIAO1 , partie 1/1 */
max_pos_couv ← |{ex ∈ E/ex est de même classe que H}|
min_neg_couv ← 0
i ← 1
Faire
    Si apparie(H, E_i)
    Alors :
        Si ex n'est pas de la classe de H Alors
            min_neg_couv++
            Si (f_e(max_pos_couv, min_neg_couv) < f_e min) Alors
                échanger E_i et Ei ÷ 2
                → 0
    Sinon :
        Si ex est de la classe de H Alors
            max_pos_couv--
            Si (f_e(max_pos_couv, min_neg_couv) < f_e min) Alors
                échanger E_i et E_{i÷2}
                → 0
    i++
Tant que (i ≤ |B|)
→ f_e(max_pos_couv, min_neg_couv)
```

Le test de couverture optimisé repose sur la constatation qu'il est possible, connaissant la valeur $f_{e\ min}$ en dessous de laquelle l'hypothèse évaluée ne sera pas insérée, de raffiner au fur et à mesure que les appariements sont effectués une estimation de la borne supérieure $f_{e\ max}(H)$ de l'évaluation de la couverture de H. Dès que $f_{e\ max}(H) < f_{e\ min}$, l'évaluation de la couverture est court-circuitée et l'individu est rejetté. Si la complexité en nombre de

tests de subsomption effectués dans le pire des cas de cette technique reste inchangée par rapport au test de couverture naïf, il est possible de noter une nette variation en moyenne. Il est également possible de constater que cette variation devient d'autant plus forte que l'algorithme, au cours de son déroulement, travaille sur des sous-populations de qualité croissante dont le seuil $f_{e\ min}$ augmente.

De plus, les performances du filtrage ont été améliorées en déplaçant les exemples négatifs couverts par H ainsi que les exemples positifs non couverts par H de telle manière qu'ils soient testés en premier : selon l'application, les premiers exemples de la base d'apprentissage correspondront donc aux soit aux exemples les plus discriminants (appelés nuances critiques ou *near-misses*), soit aux exemples bruités. Un compromis validé expérimentalement a été trouvé afin de minimiser le coût de la réorganisation de B, à la fois en nombre de déplacements ainsi que du point de vue de la complexité de la méthode utilisée. Ce dernier consiste à ne procéder qu'à un déplacement au maximum par évaluation de couverture en échangeant les exemples d'indices i et $i \div 2$ lorsque le $i^{ème}$ exemple permet l'application immédiate d'un court-circuit. En effet, d'une part cet exemple n'est autre que l'exemple discriminant d'indice maximal dont on peut avoir connaissance (compte-tenu de la stratégie adoptée), et d'autre part l'échange des exemples i et $i \div 2$ correspond à un déplacement vers le début de la liste des exemples qui est proportionnel à l'éloignement et de coût très faible (contrairement, par exemple, à un décalage de plusieurs indices dans une liste).

Validation expérimentale

Afin de valider globalement l'optimisation proposée, des données relatives au développement *d'un* noyau choisi aléatoirement ont été récoltées sur différents problèmes d'apprentissage (cf. tableau 4.3). Il découle de ces résultats que le gain observé en pratique est important, qu'il s'agisse de la réduction du nombre d'appariements ou du temps d'exécution. D'autre part, le fait que les gains en nombre d'appariements et en temps soient voisins démontre à la fois la prépondérance du coût relatif aux tests de couverture ainsi que le faible surcoût de calcul de $f_{e\ min}$ et du raffinement progressif de l'estimation de f_e.

Base	**Gain** (temps)[a]	**Gain** (appariements)	**Appariements**	
			Méthode normale	Méthode optimisée
Iris	$51,56\%$	$60,02\%$	$8,8641 \cdot 10^6$	$3,5437 \cdot 10^6$
1000 Trains	$55,37\%$	$57,96\%$	$9,6230 \cdot 10^6$	$4,0451 \cdot 10^6$
Mushrooms	$73,78\%$	$75,14\%$	$2,0360 \cdot 10^7$	$5,0616 \cdot 10^6$
Census-Income	ND	$25,83\%$	$2,1369 \cdot 10^9$	$1,5848 \cdot 10^9$

TAB. 4.3: Réduction observée du nombre d'appariements sur différentes bases.

[a] Cette mesure comprend le chargement des données, le développement du noyau et le post-traitement.

Les graphes 4.5 – 4.8 détaillent la variation du nombre moyen d'appariements nécessaires à l'évaluation d'un individu au cours du développement *d'un* noyau. Chaque évolution a été découpée en 50 parties égales ; pour chacune de ces parties le ratio moyen du nombre d'appariements $\frac{méthode\ optimisée}{méthode\ normale}$ et son écart-type ont été calculés (les courbes correspondent donc à des données normalisées). Le phénomène de réduction progressive du nombre moyen d'appariements nécessaires à l'évaluation des individus est très nettement perceptible sur les bases d'apprentissage peu bruitées. Ce phénomène combine d'une part les effets du déplacement des exemples les plus discriminants et d'autre part l'augmentation du seuil $f_{e\ min}$ au cours de l'évolution. Par contre, dans le cas des bases comportant un taux de bruit important (c'est-à-dire lorsqu'il est nécessaire de relaxer la cohérence des règles afin d'obtenir une bonne couverture), la réduction du nombre d'appariements est moindre et non décroissante dans la durée.

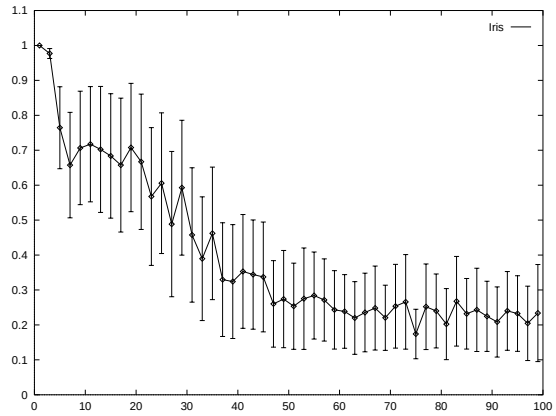


FIG. 4.5: *Iris*

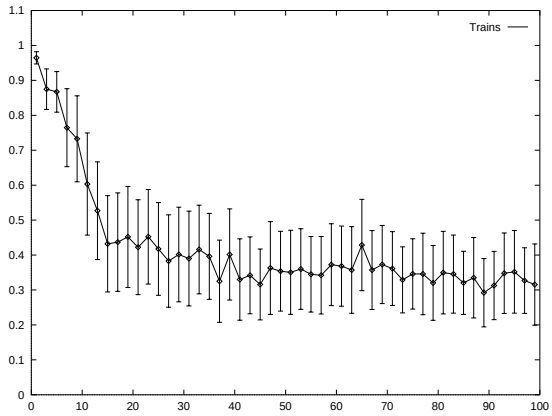


FIG. 4.6: *1000 Trains*

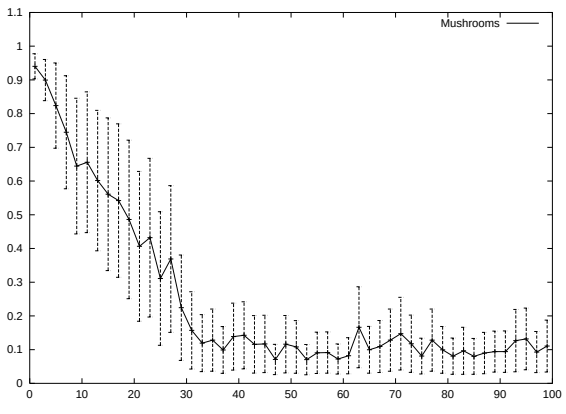


FIG. 4.7: *Mushrooms*

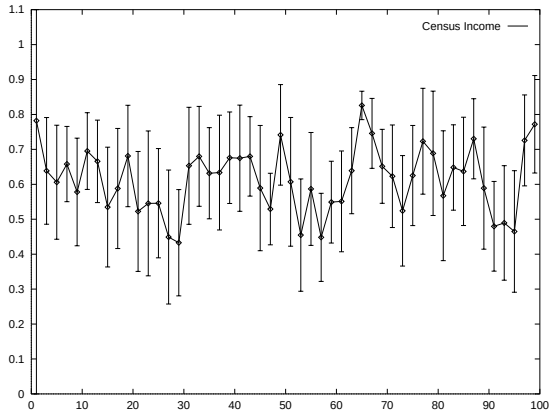


FIG. 4.8: *Census-Income*

Réduction du nombre d'évaluations

Un technique utilisée par bien des algorithmes de recherche travaillant dans des domaines où le coût d'évaluation d'un individu est prohibitif consiste à s'appuyer sur une mémoire conservant les résultats des évaluations. Dans le cadre des algorithmes d'évolution, cette idée vient d'autant plus facilement que la population tient déjà ce rôle pour l'algorithme. De plus, comme $\mathcal{SIAO}1$ est doté d'un schéma de sélection élitiste non redondant, l'idée de l'ajout d'un filtre écartant les individus similaires en amont du test d'inclusion – c'est-à-dire avant évaluation – est immédiate.

Il est cependant possible de compléter ce filtre puisque ce dernier ne s'appuie que sur une mémoire des individus performants déjà présents. Une autre mémoire (de taille identique à la population), structurée sous forme de file et fonctionnant comme un ensemble tabou [Glover et al.93], a donc été ajoutée afin de prendre en compte les individus peu performants ne faisant pas partie de la population.

Validation expérimentale

Les résultats ci-dessous montrent que si le premier filtre classique ne diminue pas notablement le nombre d'évaluations, le second, basé sur la mémorisation des individus tabous, se révèle plus efficace.

Base	**Gain total** (évaluations)	**Évaluations**		
		Sans filtres	Filtre simple	Filtre tabou
Iris	72,31%	63 764	7 256 (11,38%)	38 855 (60,93%)
1000 Trains	65,78%	74 341	5 357 (7,21%)	43 544 (58,57%)
Mushrooms	70,52%	75 474	8 584 (11,37%)	44 648 (59,15%)
Census-Income	38,43%	2 211 304	400 620 (18,11%)	449 247 (20,32%)

TAB. 4.4: Réduction observée du nombre d'évaluations sur différentes bases.

4.3 Les opérateurs d'évolution

Les opérateurs de mutation et de croisement de $\mathcal{SIAO}1$, qui se distinguent très clairement des variantes évolutionnaires habituelles, sont décrits en détail dans cette section. Les principales différences, qui sont motivées par les principes fondant la stratégie de recherche de $\mathcal{SIAO}1$, sont rappelées et résumées à la section 4.3.3, consacrée à l'analyse du biais de recherche.

4.3.1 La mutation

Tout comme la mutation classique utilisée dans l'AGC, il s'agit d'un opérateur unaire stochastique dont la fonction est de permettre l'exploration du voisinage immédiat d'un individu. Intuitivement, l'application de la mutation peut donc se concevoir du point de vue de l'exploration de l'espace de recherche comme une marche aléatoire ayant pour origine l'individu et procédant par petits pas.

Son fonctionnement est le suivant :

1. Déterminer l'ensemble $\mathcal{G}$ des gènes généralisables (il s'agit des gènes ne valant pas $*$ et n'appartenant pas à un atome abandonné). Soit par exemple le chromosome :

Chromosome	Forme	Type		Valeurs
gène n° 1	*Prédicat*	—	→	*Père*
gène n° 2	*Variable*	`Personne`	→	X
gène n° 3	*Constante*	`Personne`	→	*jean*
gène n° 4	*Prédicat*	—	→	$\times$
gène n° 5	*Variable*	`Personne`	→	Y
gène n° 6	*Constante*	`Personne`	→	*christophe*
gène n° 7	*Prédicat*	—	→	*Mère*
gène n° 8	*Variable*	`Personne`	→	X
gène n° 9	*Constante*	`Personne`	→	`*`

   $\mathcal{G} = \{1, 2, 3, 7, 8\}$ puisque le neuvième gène (qui vaut $*$) ne peut être généralisé et que l'atome relatif aux gènes 4 à 6 est abandonné ($\times$ dénote un symbole de prédicat abandonné).

2. Sélectionner aléatoirement (équiprobabilité) un gène-cible $g \in \mathcal{G}$.

3. Sélectionner en fonction du type de g et des biais sur ce type l'une des règles $\mathcal{R}_1 ... \mathcal{R}_6$ de généralisation et l'appliquer.

Les règles de généralisation, qui sont décrites ci-dessous, sont pour la plupart adaptées de celles proposées par Michalski dans le cadre du calcul des prédicats annotés (APC, [Michalski84]). A la description de la règle ont été ajoutés entre crochets les noms des règles figurant dans [Michalski84] qui s'en approchent le plus. L'élément souligné dans les exemples fournis correspond au gène g sur lequel porte la mutation.

$\mathcal{R}_1$ Généralisation d'un symbole de prédicat

[Constructive generalization rule, Dropping condition rule]

Le symbole de prédicat sélectionné est remplacé par son père dans la hiérarchie. S'il n'a pas de père, l'atome est abandonné.

Exemples :

$\underline{\text{Voiture}}(X) \xrightarrow{\mu} \text{Véhicule}(X)$

$\underline{\text{Véhicule}}(X) \xrightarrow{\mu} \text{Vrai}$

$\mathcal{R}_2$ Généralisation fonctionnelle d'une constante symbolique

[Climbing generalization rule, Extending reference rule]

Cette règle applique aléatoirement l'une des deux transformations suivantes : soit le symbole de constante g sélectionné est remplacé par son père dans la hiérarchie, soit il donne naissance à une disjonction interne le contenant et à laquelle on ajoute un autre symbole de constante *choisi*. Si g n'a pas de père, ou ne peut être complété par un autre symbole de constante (l'ensemble de *choix* est vide), alors il est remplacé par $*$.

Exemples :

$\text{Élément}(X,\underline{\text{rectangle}},\text{rouge}) \xrightarrow{\mu} \text{Élément}(X,\text{polygone},\text{rouge})$

$\text{Élément}(X,\underline{\text{polygone}},\text{rouge}) \xrightarrow{\mu} \text{Élément}(X,*,\text{rouge})$

$\text{Élément}(X,\underline{\text{rectangle}},\text{rouge}) \xrightarrow{\mu} \text{Élément}(X,\{\text{rectangle} \vee \text{cercle}\},\text{rouge})$

$\mathcal{R}_3$ Généralisation relationnelle d'une constante symbolique

[Turning constraints into variable rule]

Les occurrences du symbole de constante sélectionné sont remplacées par un nouveau symbole de variable[7] avec une probabilité donnée.

Exemples :

$\text{Père}(\text{pierre},\underline{\text{jean}}) \wedge \text{Père}(\text{jean},\text{christophe}) \wedge \text{Mère}(\text{jean},\text{elisabeth})$
$\xrightarrow{\mu} \text{Père}(\text{pierre},X) \wedge \text{Père}(X,\text{christophe}) \wedge \text{Mère}(X,\text{elisabeth})$

$\text{Père}(\text{pierre},\underline{\text{jean}}) \wedge \text{Père}(\text{jean},\text{christophe}) \wedge \text{Mère}(\text{jean},\text{elisabeth})$
$\xrightarrow{\mu} \text{Père}(\text{pierre},X) \wedge \text{Père}(X,\text{christophe}) \wedge \text{Mère}(\text{jean},\text{elisabeth})$

$\mathcal{R}_4$ Généralisation fonctionnelle d'une constante numérique

[Extending reference rule, Closing interval rule]

La constante numérique g sélectionnée donne naissance soit à une disjonction interne la contenant et à laquelle on ajoute une autre constante numérique *choisie*, soit à un intervalle numérique généré en *choisissant* une constante supérieure ou inférieure.

[7] En effet, la variabilisation à l'aide d'un symbole de variable déjà présent ne produit pas une formule strictement plus générale comme le montre la transformation suivante : $P(a,b,c) \xrightarrow{\theta:\{X/a\}} P(X,b,c) \xrightarrow{\theta:\{X/c\}} P(X,b,X)$.

Exemples :

Température(X,97,3) $\xrightarrow{\mu}$ Température(X,[97,3 .. 98,5])

Température(X,97,3) $\xrightarrow{\mu}$ Température(X,{86,2 $\vee$ 97,3})

$\mathcal{R}_5$ Généralisation relationnelle d'une constante numérique

[Turning constraints into variable rule]

Les occurrences du symbole de constante sélectionné sont remplacées par un nouveau symbole de variable[6] avec une probabilité donnée.

Exemple :

Plus(3,0,3) $\xrightarrow{\mu}$ Plus(X,0,X)

$\mathcal{R}_6$ Extension des domaines

[Extending reference rule]

Il est possible de poursuivre la relaxation des contraintes en continuant de généraliser des domaines g lorsque ceux-ci sont sélectionnés.

Exemples :

Élément(X,{ rectangle $\vee$ cercle},rouge)
$\xrightarrow{\mu}$ Élément(X,{ rectangle $\vee$ cercle $\vee$ triangle},rouge)

Père(pierre,X) $\wedge$ Père(X,christophe) $\wedge$ Mère(X,elisabeth)
$\xrightarrow{\mu}$ Père(pierre,Y) $\wedge$ Père(Y,christophe) $\wedge$ Mère(X,elisabeth)

Température(X,[97,3 .. 98,5]) $\xrightarrow{\mu}$ Température(X,[97,3 .. 102,4])

Température(X,{97,3 $\vee$ 86,2}) $\xrightarrow{\mu}$ Température(X,{86,2 $\vee$ 97,3 $\vee$ 98,5})

Bien que la mutation de $\mathcal{SIAO}1$ partage quelques points communs (mentionnés auparavant) avec l'opérateur de mutation de l'AGC, son adaptation à la stratégie de recherche ainsi qu'à la représentation utilisées lui confère des propriétés qui l'en différencient nettement :

Orientation ascendante

Afin de se conformer au type de recherche envisagé (la croissance de noyaux par généralisations successives), la mutation est ici un opérateur unidirectionnel pour la relation d'ordre $\vdash_2$. Contrairement à la mutation de l'AGC, il n'existe donc aucune possibilité de retour en arrière , puisqu'une telle opération correspondrait à une spécialisation de la formule.

Fréquence élevée

Dans le cadre de l'AGC (expliqué par [Goldberg89]), le rôle de la mutation se réduit à perturber avec une faible probabilité les individus de la population afin de s'assurer

que tout point de l'espace de recherche peut être atteint. En ce qui concerne $\mathcal{SIAO}1$, le rôle de la mutation est aussi important que celui du croisement puisqu'il s'agit de permettre une ascension du treillis de généralisation dans le voisinage de l'individu considéré. L'importance de cet opérateur peut être mesurée par le fait que son taux d'utilisation, déterminé par la stratégie d'allocation des ressources aux opérateurs, peut être voisin de 50% sur certains problèmes.

Comportement non local

Bien qu'elle s'applique à partir d'un unique gène sélectionné et qu'elle ait pour but d'explorer le voisinage immédiat de l'individu, la mutation de $\mathcal{SIAO}1$ peut être amenée à modifier ou avoir des répercussions directes sur plus d'un gène dans le chromosome (par exemple lors d'une variabilisation). Ce phénomène est dû au fait que les espaces génotypiques et phénotypiques ne sont pas structurés de la même manière, puisque certaines modifications de faible distance phénotypique ne peuvent être obtenues que par des modifications plus importantes du point de vue génotypique. Les figures 4.9 et 4.10 montrent les distances entre différentes généralisations d'une formule du point de vue du génotype (il s'agit du nombre de gènes modifiés) et du point de vue du phénotype (il s'agit de la distance dans le treillis des généralisés qui est structuré par la relation de subsomption).

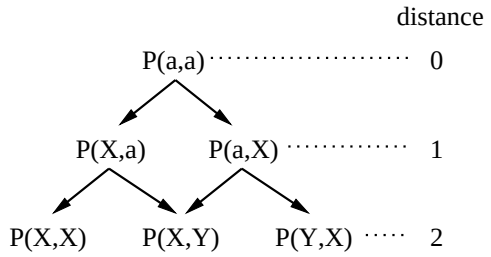


FIG. 4.9: Structure de l'espace (génotype)

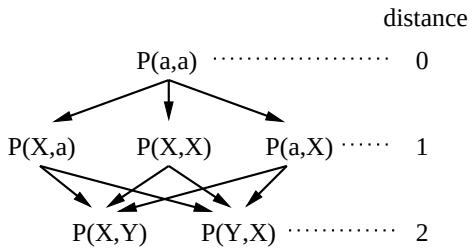


FIG. 4.10: Structure de l'espace (phénotype)[1]

[1] La relation $\vdash_2$ étant transitive, les liens redondants du treillis n'ont pas été représentés.

Fonctionnement (faiblement) dirigé par les données

Une dernière caractéristique de la mutation proposée, qui la différencie de l'opérateur classique, provient des critères mis en jeu lors des *choix* nécessaires à l'application des règles. En effet, malgré l'aspect stochastique et peu informé de l'opérateur, il est possible de prendre en compte certaines informations simples collectées lors du pré-traitement de la base d'apprentissage, comme par exemple la distribution des constantes symboliques ou numériques au sein des exemples d'apprentissage en fonction de leur classe. Ne faisant aucune hypothèse sur les relations possibles au sein des données, les critères de choix spécifiés ci-dessous peuvent tout de même guider dans une certaine mesure la recherche d'une solution et éviter certaines surgénéralisations de disjonctions internes.

- Le *choix* du symbole de constante numérique (resp. symbolique) à ajouter pour créer ou étendre une disjonction numérique (resp. symbolique) interne se fait à

équiprobabilité parmi toutes les constantes non déjà présentes dans la disjonction et dont la fréquence d'apparition, parmi les exemples de la classe cible, est non nulle.

– Le *choix* de la constante numérique étendant la borne supérieure (resp. inférieure) d'un intervalle numérique se porte sur la première valeur de coupure supérieure (resp. inférieure) à la borne considérée. Une valeur numérique est dite de coupure si les exemples au sein desquels elle apparaît ne sont pas d'une même classe ou si les exemples au sein desquels apparaissent la valeur immédiatement précédente ou suivante peuvent appartenir à une classe différente.

Exemples :

Constantes Symboliques	Distribution[a]	
	Classe A	Classe B
a *	1	15
b	0	23
c *	10	0
d *	14	7

Constantes Numériques	Distribution[a]	
	Classe A	Classe B
1,0	0	4
1,5 #	0	6
1,8 # *	2	5
2,0 # *	7	0

[a]sur la base d'apprentissage.

En considérant une règle concluant sur la classe A, seules les constantes suivies d'une étoile dans les tableaux peuvent être intégrées dans une disjonction symbolique (resp. numérique). Dans le cas de la création ou de l'extension des intervalles numériques, chaque point de coupure entre valeurs est symbolisé par un trait vertical. Une valeur de coupure (dénotée par #) est située de part ou d'autre d'un point de coupure.

4.3.2 Le croisement

Les opérateurs de croisement classiques, que l'on retrouve dans les algorithmes d'évolution d'inspiration génétique tels que les algorithmes génétiques et la programmation génétique, sont caractérisés par le fait qu'ils s'appliquent sur deux individus de la population et procèdent à un échange de parties (déterminées aléatoirement) des chromosomes correspondants. L'analyse de ces algorithmes dans le cadre de leur application à l'optimisation de fonctions (chapitre 2, section 2.1) montre que le croisement semble particulièrement adapté aux représentations dont les briques de construction sont petites, non redondantes, et indépendantes (d'après l'hypothèse des *building blocks* qui découle du théorème des schémas).

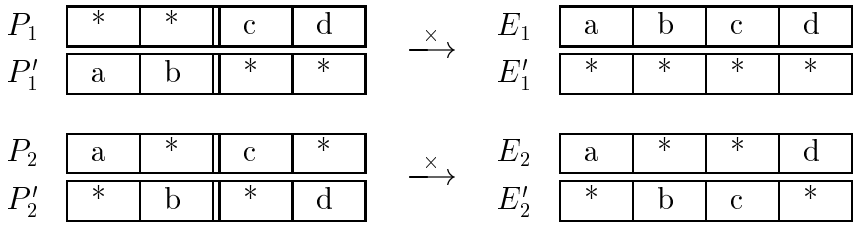
Inconvénients des croisements usuels

Bien que les opérateurs de croisement classiques (croisements à 1 ou n points, croisement uniforme, ...) soient incorporés dans la plupart des systèmes d'apprentissage fondés sur les algorithmes évolutionnaires, ils présentent de nombreux inconvénients du point de vue

de l'AS. Les paragraphes suivants détaillent ces défauts et motivent la conception d'un opérateur différent pour $\mathcal{SIAO}1$.

Les croisements classiques sont des opérateurs aveugles pour l'AS

Contrairement au cadre de l'optimisation qui caractérise le contexte d'application des algorithmes génétiques, le domaine de l'apprentissage supervisé permet, rien que de par sa spécification, de disposer de connaissances (dérivées, par exemple, de la relation de généralité) qui pourraient être exploitées par les opérateurs de recherche. La figure 4.11 illustre le comportement multidirectionnel du croisement du point de vue de la relation de généralité[8] : étant donnés les parents (P_1, P_1') et (P_2, P_2'), il est possible de générer par échange de gènes des enfants (E_1, E_1') et (E_2, E_2') plus généraux, plus spécifiques ou encore incomparables par rapport à leurs parents ; et ce même lorsque le langage est d'ordre 0 et que tous les individus proviennent d'un noyau commun (dans cet exemple : $\boxed{a \mid b \mid c \mid d}$). D'autre part, comme il a



E_1 est plus général que P_1 et P_1', alors que E_1' est plus spécifique que P_1 et P_1'. E_2 et E_2' sont incomparables avec P_2 et P_2'. Les points de coupure sont représentés par $||$.

FIG. 4.11: Exemples de croisements par échange de gènes.

été montré dans la section précédente, des opérateurs unaires tels que la mutation peuvent être faiblement dirigés par les données à l'aide de connaissances concernant les distributions des symboles par rapport aux classes. Il devrait donc être possible de faire bénéficier des opérateurs binaires tels que le croisement de connaissances encore plus spécifiques telles que les exemples de la base d'apprentissage.

Les opérateurs de croisement classiques sont donc aveugles sur les problèmes d'apprentissage puisqu'ils ne tirent pas parti d'informations spécifiques à ce contexte telles que :

- la relation de généralité qui structure l'espace de recherche ;
- la base d'apprentissage qui fonde en grande partie le critère d'évaluation des hypothèses.

[8]Sous l'hypothèse, toujours vérifiée en pratique, qu'un gène représente un élément syntaxique pouvant être généralisé.

Cas des concepts fortement disjonctifs ou non convexes

L'une des premières intuitions[9] concernant le fonctionnement du croisement est que ce dernier est d'autant plus satisfaisant que la combinaison des points forts des parents produit des enfants performants. Cependant cette propriété n'est pas vérifiée dans le domaine de l'apprentissage supervisé de concepts disjonctifs, puisque les briques de construction correspondent, pour la plupart des représentations choisies, aux domaines des descripteurs. Comme le montre la figure 4.12 dans le cas d'un

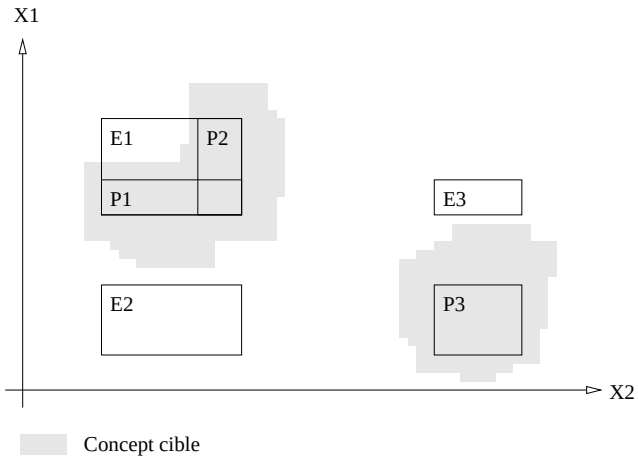


FIG. 4.12: Croisements dans un espace à deux descripteurs.

domaine à deux descripteurs, où chaque règle est de la forme "$X_1 \in [a_1, b_1] \wedge X_2 \in [a_2, b_2]$", il est très peu probable que le croisement de deux individus appartenant à des sous-concepts distincts crée des individus viables : les règles P1 et P3 ne peuvent que produire E2 ou E3 dont la capacité prédictive du concept cible est nulle. De plus, dès que les règles ont une couverture importante, et à moins que le concept cible soit convexe, le croisement d'individus d'un même sous-concept risque d'être inefficace : la combinaison de P1 et P2 produit soit un individu moins général que ses parents, soit l'individu E1 de faible cohérence. Cette figure illustre également le comportement erratique des croisements classiques, autant du point de vue de la structure de l'espace de recherche que du point de vue des connaissances relatives au problème. En effet, l'exploitation judicieuse de la relation de généralité conduit typiquement les opérateurs à dilater ou réduire les rectangles, alors que la prise en compte des exemples d'apprentissage consiste par exemple à ajuster un rectangle de telle manière qu'il couvre (resp. ne couvre pas) un exemple (resp. contre-exemple).

Les briques de construction ne sont pas indépendantes

Le problème de la pertinence des croisements évolutionnaires classiques est encore plus aigu dans le cadre de l'apprentissage supervisé relationnel parce que les briques de construction sont très fortement interdépendantes. Les liens créés par les symboles

[9]Il s'agit ici d'une reformulation intuitive des résultats qui découlent du théorème des schémas permettant de caractériser les fonctions AG-difficiles.

de variable sont en effet autant de contraintes non locales ayant une très grande influence sur la qualité de la règle codée.

Une multitude de travaux en PLI soulignent cet aspect du domaine, d'autant plus que les nombreux systèmes basés sur des approches descendantes (par spécialisation) sont particulièrement sensibles à ce phénomène. Ainsi, le fait que FOIL ([Quinlan et al.93, Quinlan et al.95], voir également chapitre 1 section 1.5.2) ne puisse induire certains concepts à cause de l'heuristique locale du gain d'information qui est basée sur le calcul de l'avantage à ajouter un atome est un inconvénient reconnu. Une multitude de stratégies, telles que la recherche de chemins relationnels [Richards et al.92] à l'aide du graphe des liens entre constantes, ou encore la spécification de clichés relationnels [Silverstein et al.91] (motifs génériques associant plusieurs prédicats ainsi que des liens spécifiques) ont été développées afin d'échapper à la myopie des approches locales : elle permettent de simuler partiellement un mécanisme d'anticipation (*look-ahead*), tout en maintenant l'explosion combinatoire dans des bornes raisonnables.

D'autres systèmes nécessitent une intervention de l'utilisateur afin que ce dernier spécifie des biais qui conduiront à la découverte du chemin relationnel : PROGOL ([Muggleton95], voir également chapitre 1 section 1.5.2) se réfère par exemple à un ensemble M de modes qui doit lui être fourni. Ces modes, qui définissent l'espace de recherche contraint $\mathcal{L}(M)$ décrivent entre autres :

- les prédicats pouvant figurer dans la tête ou le corps des clauses générées (déclarations `modeh` et `modeb`) ;
- le type des arguments des prédicats (`+type` et `-type` font référence à un symbole de variable typé et orienté, `#type` demande la présence d'un terme clos typé) ;
- la structure des liens reliant les prédicats (un symbole de variable étiqueté `+` dans le corps de la clause doit obligatoirement avoir au moins une occurrence déjà présente dans le corps de la clause et étiquetée `-`, ou dans la tête de la clause et étiquetée `+`) ;
- une borne (éventuellement infinie) au nombre maximum d'occurrences de chaque symbole de prédicat.

Les biais de recherche et de langage évoqués ci-dessus sont autant d'indices soulignant d'une part l'interdépendance souvent très forte entre les différentes composantes de formules solution aux problèmes de PLI, et d'autre part la complexité prohibitive de la recherche de ces liens lorsqu'elle n'est pas contrainte.

Le croisement n'est pas adapté à $\mathcal{SIAO}1$

La stratégie sous-tendant l'algorithme de $\mathcal{SIAO}1$ étant basée sur le développement de noyaux à l'aide d'une recherche unidirectionnelle ascendante, cette dernière conduit à préférer des opérateurs aussi monotones ascendants et minimaux que possible. Comme l'ont montré les remarques précédentes, les croisements évolutionnaires

classiques ne sont ni monotones ascendants, ni minimaux puisqu'ils génèrent des individus plus généraux, spécifiques ou incomparables à leurs parents et que l'échange de sous-chaînes brise les liens relationnels.

Les premières versions de $\mathcal{SIAO}1$ [Augier et al.95] mettaient en œuvre deux types de croisements procédant par échange de gènes. Puisque toutes les formules d'une population proviennent d'un même noyau et que l'opérateur de mutation ne modifie pas la structure des formules, ces opérateurs généraient des chromosomes sémantiquement corrects. Cependant, leur intérêt était limité puisqu'ils se comportaient au mieux comme des opérateurs d'hypermutation.

Le paragraphe qui suit est consacré à l'opérateur de croisement actuel qui procure, par rapport à la version précédente, plusieurs avantages majeurs :

- Il peut être dirigé par les données, et génère des individus dont la structure relationnelle est adaptée au problème et non obligatoirement incluse dans la structure du noyau.
- Il est ascendant (par rapport à $\vdash_2$), rapidement calculable et non disruptif.

L'opérateur de croisement de $\mathcal{SIAO}1$

L'ensemble des remarques portant sur l'adéquation des croisements évolutionnaires classiques aux problèmes d'apprentissage relationnel et à la stratégie de recherche de $\mathcal{SIAO}1$ ont motivé la création d'un opérateur s'éloignant sensiblement des canons du domaine.

Le croisement proposé est fondé sur une approximation du moindre généralisé relationnel (*lgg : least general generalization, [Plotkin70])* noté *mg* en français :

Définition 4.1 *Moindre généralisé de deux atomes*
Soient $A_1 = P_1(t_{(1,1)}, ..., t_{(1,k)})$ *et* $A_2 = P_2(t_{(2,1)}, ..., t_{(2,l)})$ *deux atomes. Alors, si les symboles de prédicat sont égaux (et donc de même arité),*

$$mg(A_1, A_2) = P_1(mg(t_{(1,1)}, t_{(2,1)}), ..., mg(t_{(1,k)}, t_{(2,k)}))$$

avec :

- $mg(t_{(1,i)}, t_{(2,i)}) = t_{(1,i)}$ *si* $t_{(1,i)} = t_{(2,i)}$
- $mg(t_{(1,i)}, t_{(2,i)}) = X_{(t_{(1,i)}, t_{(2,i)})}$ *si* $t_{(1,i)} \neq t_{(2,i)}$.
  $X_{(t_{(1,i)}, t_{(2,i)})}$ *est un symbole de variable étiqueté.*

Définition 4.2 *Moindre généralisé de deux formules*
Soient $F_1 = A_{(1,1)} \wedge ... \wedge A_{(1,n)}$ *et* $F_2 = A_{(2,1)} \wedge ... \wedge A_{(2,m)}$ *deux formules. Alors,*

$$mg(F_1, F_2) = \bigwedge_{i \in [1,n], j \in [1,m]} mg(A_{(1,i)}, A_{(2,j)})$$

Ces deux définitions sont immédiatement applicables dans le cadre du langage $\mathcal{L}_h$ par extension des définitions de $mg(A_1, A_2)$:

- Lorsque deux prédicats P_1 et P_2 ont un plus petit ancêtre commun dans la hiérarchie représentant la théorie du domaine, ce dernier est par définition $mg(P_1, P_2)$.
- Par contre, le moindre généralisé de deux constantes symboliques $t_{(1,i)}$ et $t_{(2,j)}$ est toujours la disjonction symbolique $\{t_{(1,i)} \vee t_{(2,j)}\}$ dans le cas d'une modalité fonctionnelle, ou un symbole de variable étiqueté dans le cas d'une modalité relationnelle. L'exclusion mutuelle des modalités relationnelles et fonctionnelles assure l'unicité du moindre généralisé de deux termes.
- Dans le cas des constantes numériques, il a été décidé pour des raisons d'intelligibilité de générer, plutôt qu'une disjonction, un intervalle. Cette règle ne produit pas le moindre généralisé des deux termes mais un presque moindre généralisé .

Les figures 4.13 et 4.14 représentent graphiquement le calcul des moindres généralisés :

- $mg(\ P(A,B) \wedge P(B,C) \wedge P(C,D) \wedge P(D,A)\ ,\ P(X,Y) \wedge P(Y,Z) \wedge P(Z,X)\ )$;
- $mg(\ P(A,B) \wedge P(B,C) \wedge P(C,A)\ ,\ P(X,Y) \wedge P(Y,Z)\ )$.

Les chiffres correspondent aux symboles de variable du résultat. Ces derniers sont issus du regroupement d'une variable de chaque formule fournie en argument. Par exemple, dans le cas de la figure 4.13, le chiffre 1 correspond à la variable étiquetée V_{AX} issue des variables A et X.

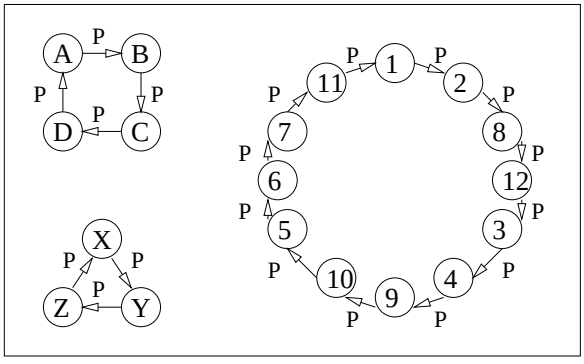


FIG. 4.13: Exemple 1

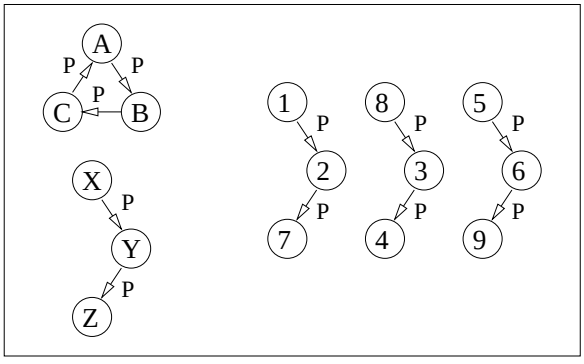


FIG. 4.14: Exemple 2

Il est possible d'observer :

- que la taille maximale du moindre généralisé de deux formules F_1 et F_2 est $|F_1|.|F_2|$ (voir figure 4.13) ;
- que la formule résultant du calcul n'est pas toujours sous forme simplifiée (voir figure 4.14).

Afin d'atténuer, comme dans le cas de GOLEM, l'explosion combinatoire des tailles des formules générées, une stratégie a été proposée dans le but d'extraire rapidement du presque moindre généralisé une composante de taille N aussi spécifique que possible[10]. En notant Res le résultat construit par ajout successif d'atomes du presque moindre généralisé Pmg, le pseudo-code de l'algorithme d'extraction est le suivant :

- Générer la matrice de connexion entre les littéraux du presque moindre généralisé
- Tant que $(N > |Res|) \wedge (Pmg \neq \emptyset)$
  - Placer dans Res un atome de Pmg dont le taux de connection avec les éléments de Res est maximum.

Les avantages de la solution proposée sont nombreux :

Contrairement aux croisements classiques, l'opérateur proposé peut être dirigé par les données (lorsqu'il s'applique à un exemple de la base d'apprentissage de classe adéquate) : il est donc capable de construire une formule relationnelle dont la structure est adaptée au problème. De plus, le croisement de $\mathcal{SIAO}1$ n'est pas disruptif puisque les blocs de construction considérés ne sont pas relatifs aux éléments syntaxiques des chromosomes, mais aux positions des formules dans le treillis $\vdash_2$.

Contrairement à [Sebag98], qui propose dans le cadre de l'identité d'objet un opérateur stochastique dont l'avantage est d'être extrêmement rapide, la forme générée par le croisement peut être différente des formes des exemples de la base d'apprentissage.

Du point de vue de $\mathcal{SIAO}1$, cet opérateur est monotone ascendant pour la relation $\vdash_2$. L'heuristique utilisée permet de gérer le compromis entre la taille des hypothèses générées et leur proximité (dans le treillis) du moindre généralisé des formules sur lesquelles il s'applique : la recherche étant ascendante, il est en effet préférable d'éviter autant que possible les surgénéralisations.

4.3.3 Récapitulatif : le biais de recherche de $\mathcal{SIAO}1$

Le biais de recherche de $\mathcal{SIAO}1$ est caractérisé au niveau des opérateurs d'évolution par deux grands principes qui le distinguent clairement des autres approches évolutionnaires.

Les opérateurs sont ascendants et procèdent par petits pas

Une première caractéristique de $\mathcal{SIAO}1$ est l'exploration dirigée du treillis défini par la relation de généralité $\vdash_2$. Cette prise en compte de la structure de l'espace de recherche, associée au schéma de sélection, assure une exploration efficace puisque les cycles et les redondances sont rendus impossibles. Par contre, cette même propriété suppose que les

[10] Le paramètre N est un biais de langage pouvant être ajusté par l'utilisateur.

opérateurs puissent parfois générer des individus suffisamment proches de leurs parents afin de fournir une solution de bonne qualité : puisqu'aucun retour en arrière n'est possible, l'algorithme doit être capable de se rapprocher de la solution par petits pas ascendants dans le treillis.

Les opérateurs sont dirigés par les données

Une seconde caractéristique de $\mathcal{SIAO}1$ vient de l'utilisation, par les opérateurs, de données provenant de la base d'apprentissage, qu'il s'agisse de la structure relationnelle des exemples ou de la distribution des symboles en fonction des classes des instances dans lesquelles ils apparaissent. Deux points de vue peuvent alors être portés sur cette démarche.

Une première analyse, pragmatique, permet de constater que l'introduction d'opérateurs dirigés par les données peut sensiblement accélérer la recherche et améliorer les résultats. En effet, en supposant que les instances d'apprentissage soient représentatives des concepts-cible, il semble légitime que les sauts inductifs fondés sur les caractéristiques de ces instances occasionnent de bonnes performances en généralisation.

Une seconde analyse, plus dogmatique, critique cette approche du point de vue des algorithmes évolutionnaires car elle s'apparente à un schéma Lamarckien plutôt que Darwiniste. Ce jugement doit néanmoins être tempéré en observant que si les autres approches, qui suivent le modèle générer et tester , ont effectivement un aspect Darwiniste bien plus prononcé, ces dernières mettent également en place – de manière bien plus ténue – de tels mécanismes. Par exemple REGAL et G-NET utilisent un opérateur (le *seeding*) dirigé par les données puisqu'il génère un nouvel individu à l'aide d'un exemple du concept-cible.

4.4 Post-traitement et Classification

Cette section concerne le regroupement des règles de classification fournies par l'algorithme d'évolution afin d'obtenir l'ensemble de règles final décrivant les données, ainsi que l'exploitation de cet ensemble pour la classification de nouvelles instances.

Contrairement à SIA, la définition d'une distance entre formules est beaucoup plus difficile à obtenir dans le cas de la logique relationnelle que dans le cadre attribut-valeur. Cette différence a motivé l'élaboration de nouvelles stratégies pour élaguer l'ensemble des règles obtenues et pour utiliser cet ensemble afin de classer de nouvelles instances. A une stratégie de décision basée sur la distance des instances aux règles et dans le cas d'un conflit s'appuyant sur la qualité normalisée des règles en désaccord, on a préféré un système de vote basé sur la cohérence observée des règles ainsi que l'adjonction d'une classe par défaut. Par un effet rétroactif, la prise en compte de la cohérence des règles lors du vote a nécessité le développement d'un algorithme d'élagage basé prioritairement sur la cohérence, et ne prenant en compte que dans un second temps l'aspect lié à la couverture de la règle considérée.

4.4.1 Classification des nouvelles instances

De très nombreux travaux en apprentissage automatique traitent des méthodes de combinaison de prédicteurs afin d'obtenir un nouveau prédicteur plus performant. Les deux méthodes les plus connues sont le *bagging (bootstrap aggregating)* et le *boosting* [Breimann96, Freund et al.96, Bauer et al.99]. Bien que d'une part ces méthodes aient pour but de combiner différents résultats obtenus en utilisant plusieurs fois un même algorithme d'apprentissage (chaque prédicteur correspond ainsi à un résultat final concernant toutes les classes cibles sur l'ensemble de l'espace de recherche), et que d'autre part la variation des conditions initiales nécessite la possibilité d'extraire différentes bases d'apprentissage (*bagging*) ou encore d'être capable de modifier les poids des instances (*boosting*), il est possible de se placer dans le contexte des règles apprises par $\mathcal{SIAO}1$. En effet, les conditions issues du cadre d'où proviennent ces travaux sont les suivantes :

1. Les prédicteurs obtenus doivent pouvoir être en désaccord ;
2. Les erreurs des prédicteurs doivent être peu ou pas corrélées ;
3. L'algorithme d'apprentissage doit s'adapter facilement à toute variation des données fournies en entrée.

Il est aisé de constater que les trois propriétés ci-dessus sont vérifiées par $\mathcal{SIAO}1$ puisqu'il est possible de considérer chaque développement d'un noyau donné comme un algorithme d'apprentissage indépendant. Le fait que l'estimateur de Laplace utilisé pour évaluer les règles générées tolère un certain niveau de bruit implique qu'il peut exister des portions de l'espace où différents prédicteurs entreront en conflit. D'autre part, chaque prédicteur est particulièrement sensible aux contraintes relatives à la portion de l'espace d'où provient le noyau à partir duquel il a été généré. Ce phénomène de localité rend donc les prédicteurs très peu corrélés (bien qu'il soit impossible de supprimer toute corrélation sur les erreurs dues au bruit). Enfin, le phénomène de localité implique également qu'un noyau développé par $\mathcal{SIAO}1$ est bien plus sensible aux variations dans son environnement qu'une méthode très stable comme k les plus proches voisins (lorsque k devient suffisamment important).

En conséquence, il semble tout à fait opportun dans le cadre de $\mathcal{SIAO}1$ de s'appuyer sur les travaux concernant la combinaison de prédicteurs. Ces travaux ayant démontré l'intérêt du vote pour la gestion des conflits entre prédicteurs peu corrélés, c'est cette méthode qui a été implantée. De plus, d'autres travaux (par exemple [Clark et al.91]) ont démontré un gain de performance lors de l'utilisation d'une procédure de vote pondéré par rapport à la méthode classique d'ordonnancement des règles en liste de décision. Au vu des résultats obtenus, il semble en effet que cette méthode, qui consiste à classer une instance par la classe de la première règle la couvrant, soit à la fois moins performante (car la combinaison des règles est bien trop simpliste) et moins intelligible (car une liste de décision contient de nombreuses négations implicites).

En ce qui concerne la méthode de vote, le choix s'est porté sur une pondération des règles

proportionnelle à l'estimation de leur cohérence pour diverses raisons : d'une part, la grande majorité des travaux du domaine ont adopté soit cette méthode, soit le vote non pondéré, et d'autre part, cette méthode est intuitive dans le sens où il semble raisonnable d'affecter un degré de croyance inversement proportionnel au taux d'erreur des règles. De plus, bien que certains travaux proposent de conserver la totalité des règles apprises et d'ajuster leur poids de manière à minimiser le taux d'erreur sur la base d'apprentissage [Merz99], de nombreuses raisons jouent en faveur de la combinaison d'un algorithme de vote et d'un algorithme d'élagage :

- la procédure d'élagage simplifie la base de règles, ce qui améliore son intelligibilité ;
- les poids des règles ont une signification pour l'utilisateur ;
- l'ajustement des poids combiné au phénomène de localité augmente les risques de surapprentissage lorsque la base est bruitée.

$\mathcal{SIAO}1$ classe donc une nouvelle instance en faisant voter l'ensemble des règles la couvrant proportionnellement à l'estimateur de Laplace de leur cohérence. Comme dans tout vote, les contributions des règles concluant sur une même classe sont ajoutées. Dans le cas où plusieurs classes obtiennent la contribution maximum, la décision résulte d'un tirage aléatoire parmi celles-ci. Lorsqu'aucune règle ne couvre l'instance à classer, une classe par défaut est choisie.

Le paragraphe suivant détaille l'obtention de l'ensemble de règles final ainsi que le choix de la classe par défaut.

4.4.2 Élagage des règles

Description de l'algorithme d'élagage

L'objectif de cette procédure de post-traitement est de réduire autant que possible le nombre de règles de décision tout en garantissant de bonnes performances par le système de vote (le choix porte donc sur les règles candidates ainsi que sur la règle par défaut qu'il est possible d'ajuster en fonction du sous-ensemble considéré). En partant du constat que tout algorithme cherchant l'ensemble de règles optimal (du point de vue de la précision) sera dans le pire des cas de complexité exponentielle en fonction du nombre de règles (puisqu'à partir de n règles il est possible de générer 2^n ensembles de règles), la recherche d'un algorithme d'élagage s'oriente presque obligatoirement vers une solution approchée. L'algorithme proposé, qui ne teste dans le pire des cas que $\frac{n.(n+1)}{2}$ ensembles de règles, est le dual de celui proposé par [Quinlan87]. En effet, alors que dans le cas de l'apprentissage (par spécialisation) d'arbres de décision, l'ensemble de règles $\mathcal{R}$ final est crée en retirant à chaque étape la règle r dont l'avantage[11] est le plus important ; dans le cas de l'appren-

[11] Il s'agit du nombre d'instances pour lesquelles la classe correcte est fournie par $\mathcal{R} \setminus r$ et non par $\mathcal{R}$, soustrait au nombre d'instances pour lesquelles le contraire se produit.

tissage par généralisation de $\mathcal{SIAO}1$ l'approche est inverse : $\mathcal{R}$ est construit à partir de zéro en ajoutant à chaque étape la règle fournissant le meilleur gain en précision sur la base d'apprentissage :

```
/* Algorithme d'élagage final pour SIAO1, partie 1/1 */
Soit Reste = {règles ayant convergé} et R = ∅
Faire
    Soit r ∈ Reste / précision(r ∪ R) soit maximum
    Soit GainP = précision (r ∪ R) − précision (R)      (*)
    Si (GainP > 0) Alors
        R ← R ∪ r
        Reste ← Reste \ r
Tant que ((GainP > 0) ∧ (Reste ≠ ∅))
```

Afin de maximiser la précision, la classe par défaut – qui est réévaluée et prise en compte à chaque calcul de *précision* – correspond à la classe la plus probable parmi les instances non couvertes par $\mathcal{R}$.

L'algorithme d'élagage proposé a pour principaux avantages d'être simple, rapide, et de fournir en pratique des solutions de bonne qualité. Par contre, il a l'inconvénient d'être inefficace lorsque toutes les hypothèses ont une couverture peu importante et caractérisent la classe majoritaire. Dans ce dernier cas, l'algorithme s'arrête immédiatement sans considérer les ensembles constitués de plusieurs hypothèses qui pourraient améliorer la qualité de la solution. En pratique, il est possible d'éviter cette configuration défavorable en ajoutant parmi les classes-cible les classes non majoritaires.

Complexité de l'algorithme d'élagage

Soit r le nombre total de règles et n le nombre d'exemples de la base d'apprentissage. En supposant que l'implantation de l'algorithme mémorise *précision*$(\mathcal{R})$ (afin d'éviter son recalcul lors de chaque évaluation de la ligne `(*)`), la complexité dans le pire des cas de l'algorithme d'élagage, notée C_1, est déterminée par le fait qu'à chaque étape $1 \leq k \leq r$ de l'algorithme, $\mathcal{R}$ contiendra $k-1$ règles et *Reste* en contiendra $r-k+1$. Il faudra donc mesurer la précision de $r-k+1$ ensembles contenant chacun k règles, pour un coût de $n.k.(n-k+1)$ tests de couverture. De plus, le calcul unique de *précision*$(\mathcal{R})$ nécessitera $n.(k-1)$ tests de couverture. Ainsi :

$$C_1(n,k) = n.\sum_{k=1}^{r} k.(r-k+1) + (k-1)$$

$$\Rightarrow C_1(n,k) = n.\sum_{k=1}^{r} k.(r+2) - k^2 - 1 = n.\left[\frac{r.(r+1)}{2}.(r+2) - \frac{r.(r+1).(2.r+1)}{6} - r\right]$$

$$\Rightarrow C_1(n,k) = n.r.\left[\frac{1}{6}.(r+1).(r+5) - 1\right] = o(n.r^3)$$

La complexité dans le pire des cas de l'algorithme d'élagage est donc, comme dans le cas de C4.5, une fonction cubique du nombre total de règles.

Réduction de la complexité de l'algorithme d'élagage

Il est possible de diminuer la complexité de l'algorithme d'élagage pour un coût en espace de $n.c$ nombres réels. Ce coût correspond à la quantité de mémoire nécessaire au stockage des sommes des votes des règles de $\mathcal{R}$ pour chaque classe et pour chaque exemple. Ce résultat étant mémorisé, l'évaluation, à chaque étape, des $r-k+1$ ensembles de k règles afin d'obtenir les valeurs de $\textit{précision}(r \cup \mathcal{R})$ ne nécessitera plus que la prise en compte de la règle r ajoutée. La complexité dans le pire des cas de l'algorithme n'est plus alors que quadratique en le nombre total de règles :

$$C_2(n,k) = n.\sum_{k=1}^{r}(r-k+1) + (k-1) = n.r^2 = o(n.r^2)$$

La figure 4.15 illustre les complexités des deux versions du post traitement $C1 = \frac{C_1(n,k)}{n} = \frac{1}{6}.r.(r+1).(r+5)$ et $C2 = \frac{C_2(n,k)}{n} = r^2$, ainsi que le gain réalisé en optant pour l'implantation de la version optimisée $C2$.

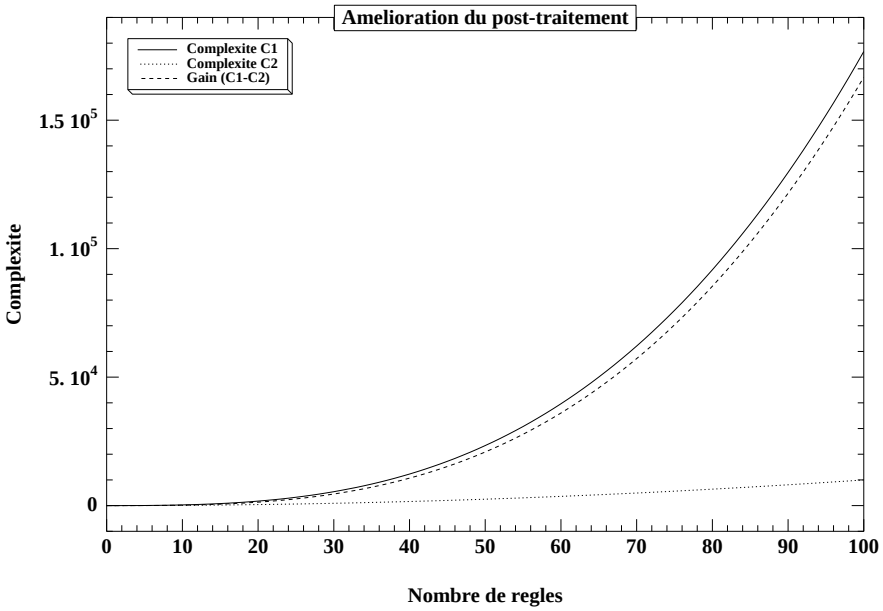


FIG. 4.15: Complexités comparées des implantations du post-traitement.

Remarque :

Poursuivant toujours dans cette direction, il semble possible d'obtenir une solution encore plus performante en mémorisant la couverture de chaque exemple par chaque règle. Le coût en espace est alors de $r.n.c$ nombres réels et le coût en temps de $r.n$ appariements. Cette solution n'apporte cependant pas les gains attendus car si le nombre d'appariements diminue effectivement, la complexité reste globalement inchangée puisqu'aux appariements évités se sont substituées des accès à la mémoire et des additions.

4.5 Connaissances du domaine

Deux mécanismes de pré-traitement ont été implantés dans $\mathcal{SIAO}1$ afin de permettre à l'utilisateur d'enrichir les données fournies au système à l'aide de connaissances propres au domaine.

Complétion symbolique relationnelle

Une première technique, appelée saturation et décrite plus formellement dans [Rouveirol94], permet d'ajouter à chaque exemple les connaissances du domaine le concernant.

Par exemple, dans le cas du problème *Mutagenesis* dont une représentation simplifiée était donnée en figure 3.1 du chapitre 3, la saturation permet de compléter chaque description de molécule qui ne comporte initialement que la classe (cancérigène ou non) et l'identifiant de la molécule par l'ensemble des atomes et des liens qui lui sont liés.

Cette technique est complémentaire des connaissances du domaine exprimées sous forme de hiérarchies sur les constantes ou les prédicats. En effet, il est préférable de définir la relation $Voiture(X) \rightarrow Véhicule(X)$ plutôt que d'ajouter dans chaque formule contenant une occurrence du prédicat $Voiture$ une occurrence du prédicat $Véhicule$ appliqué aux mêmes arguments puisque d'une part le gain en concision est un gain en espace mémoire et en intelligibilité, et que d'autre part une telle connaissance pourra être exploitée par les opérateurs de $\mathcal{SIAO}1$.

Complétion numérique

Dans le but d'améliorer le traitement des bases numériques, un système de complétion numérique a également été implanté. Ce dernier permet la dérivation de nouvelles constantes à l'aide d'une technique d'analyse de données appelée analyse discriminante (AD). Pour chaque nouvelle valeur numérique devant être créée l'utilisateur fournit au système :

- la classe-cible cl à discriminer ;
- des règles d'extraction de v valeurs numériques ;

L'AD fournit alors la combinaison linéaire des v valeurs permettant de discriminer au mieux les instances de classe cl des instances de classe $\overline{cl}$.

Soit n le nombre d'instances de la base d'apprentissage dont toutes les valeurs à extraire existent et sont définies. La matrice obtenue est alors de taille $n \times v$, et la matrice d'inertie (symétrique) est de taille $v \times v$. La complexité de l'AD est donc peu importante puisque la seule opération réellement coûteuse (une inversion de matrice) s'applique sur une matrice de taille $v \times v$, et ne dépend donc que du nombre de valeurs extraites et non du nombre d'instances de la base d'apprentissage.

La figure 4.16 illustre le comportement de $\mathcal{SIAO}1$ lors d'une pré-étude portant sur l'estimation du risque cardio-vasculaire réalisée en collaboration avec une partie de l'équipe du service d'informatique médicale de l'hôpital Broussais.

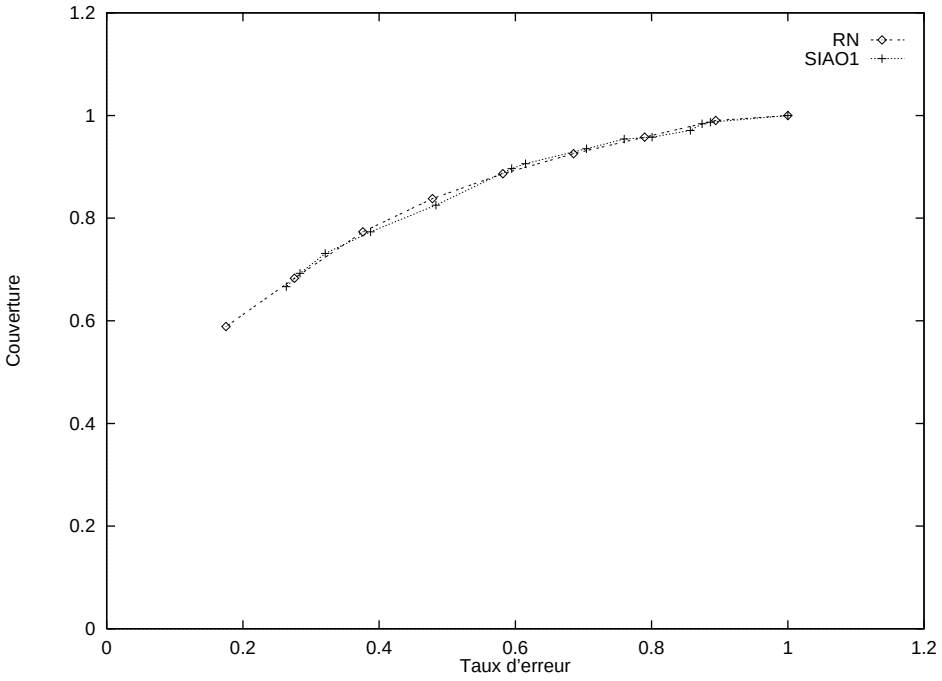


FIG. 4.16: Prédiction du risque cardio-vasculaire.

Les courbes ROC [Provost et al.97] représentées permettent de comparer le comportement d'un réseau de neurones avec celui de $\mathcal{SIAO}1$ sur la base de données *Indana* qui est bruitée et dont les 11 attributs sont numériques. Chaque point de la courbe correspond à un modèle obtenu par apprentissage, avec porté en abscisse le taux d'erreur du modèle et en ordonnée sa couverture. Il est possible de constater que les modèles fournis par $\mathcal{SIAO}1$, qui ont l'avantage d'être intelligibles, sont de qualité équivalente à ceux obtenus à l'aide de réseaux de neurones (méthodes très performantes dans le domaine numérique). La qualité des modèles fournis par $\mathcal{SIAO}1$ est dûe en grande partie à l'AD portant sur les 11 attributs initiaux qui a généré un nouvel attribut très discriminant.

4.6 Évaluation

Cette section présente les résultats obtenus par $\mathcal{SIAO}1$ sur des bases variées faisant référence et décrites en annexe B. Les résultats sont comparés, lorsque cela est possible, à ceux obtenus par différentes autres méthodes :

- C4.5, ID3 et CART sont trois systèmes d'apprentissage fondés sur les arbres de décision qui font référence pour ces techniques. Ils sont présentés au chapitre 1.
- G-Net et REGAL ont été décrits au chapitre 2. Les résultats mentionnés proviennent respectivement des thèses [Lo bello98] et [Neri97].
- PPV (1) et PPV (3) sont deux méthodes basées sur les plus proches voisins et mentionnées dans les travaux de R. Kohavi.
- RBN correspond au réseau bayésien naïf qui est une méthode extrêmement simple et rapide, fournissant dans de nombreux cas un classeur précis [Zheng et al.00].
- FOIL, PROGOL et TILDE sont des systèmes de PLI qui ont été présentés au chapitre 1.

4.6.1 Résultats sur *Iris* et *Mushrooms*

Iris et *Mushrooms* (cf. annexe B sections B.2 et B.3) sont deux bases très utilisées en apprentissage automatique : la première est de nature numérique (mais les 3 classes ne sont pas linéairement séparables), et la seconde est de nature symbolique (et fait référence du point de vue de l'apprentissage de concepts disjonctifs).

Système	Taux d'erreur (%)
C4.5	5,7
$\mathcal{SIAO}1$	6,0
RBN	6,3
CART	7,0

TAB. 4.5: Évaluation sur la base *Iris*

Dans les deux cas $\mathcal{SIAO}1$ se montre aussi performant que les méthodes de référence. En ce qui concerne la base *Mushrooms*, un taux d'erreur de 0% n'a rien d'exceptionnel pour les méthodes actuelles car il est très facile de caractériser le concept de champignon comestible. Par contre, $\mathcal{SIAO}1$ est capable de caractériser le concept de champignon empoisonné avec uniquement 3 règles et une précision de 100%, ce qui correspond au meilleur résultat obtenu par les méthodes spécialisées dans l'apprentissage de concepts disjonctifs.

Système	Taux d'erreur (%)
$\mathcal{SIAO}1$	0,0
C4.5	0,0
G-NET	0,0
REGAL	0,0
RBN	0,3
STAGGER	5,0
HILLARY	5,0

TAB. 4.6: Évaluation sur la base *Mushrooms*

4.6.2 Résultats sur 1000-*trains* et *Mutagenesis*

Les bases 1000-*trains* et *Mutagenesis* permettent d'évaluer le comportement de $\mathcal{SIAO}1$ sur des bases relationnelles.

Bien que $\mathcal{SIAO}1$ induise des concepts totalement corrects et complets sur la base d'apprentissage des 1000-*trains*, les définitions obtenues ne couvrent pas 20 des 1000 instances de test.

Système	Taux d'erreur (%)
G-Net	0,0 %
$\mathcal{SIAO}1$	2,0 %
REGAL	2,6 %

TAB. 4.7: Évaluation sur la base 1000-*trains*

La base *Mutagenesis* a été choisie afin de montrer que $\mathcal{SIAO}1$ est capable de travailler sur des problèmes fortement relationnels :

Système	Connaissances du domaine	Taux d'erreur (%)
PROGOL	B_2	19%
TILDE	B_2	21%
$\mathcal{SIAO}1$	B_2	22%
FOIL	B_2	39%

TAB. 4.8: Évaluation sur la base *Mutagenesis* (B_2)

Remarque : Pour ces deux bases, le nombre de noyaux développés a été porté à 15.

4.6.3 Résultats sur *Census-Income* et *Adult*

Census-Income (décrit annexe B section B.1) est la plus grande base publique à notre disposition. Elle permet de démontrer la capacité de l'architecture parallèle de $\mathcal{SIAO}1$ à

gérer un grand nombre d'exemples (environ 200 000 en apprentissage et 100 000 en test), dans le cas d'une base réelle extrêmement bruitée. Les seules évaluations disponibles, fournies par R. Kohavi, concernent les méthodes fondées sur les arbres de décision telles que C4.5 et la méthode du réseau bayésien naïf, qui sont toutes deux connues pour leur rapidité.

Système	Taux d'erreur (%)
C4.5[a]	4,8
$\mathcal{SIAO}1$	4,9
C4.5[b]	6,2
RBN	23,2

TAB. 4.9: Évaluation sur la base *Census-Income*

[a]Associé à une autre méthode (sans doute le boosting).
[b]Paramétrage par défaut.

Il est possible de remarquer que $\mathcal{SIAO}1$ obtient des résultats comparables à ceux de C4.5, et ce uniquement grâce à l'utilisation de la fonction de qualité fondée sur l'estimateur de laplace[12].

Système	Taux d'erreur (%)
NBTree	14,10
C4.5	15,54
CN2	16,00
RBN	16,12
$\mathcal{SIAO}1$	16,38
ID3 (vote)	16,47
T2	16,84
1R	19,54
PPV (3)	20,35
PPV (1)	21,42
Pebls	Crashed

TAB. 4.10: Évaluation sur la base *Adult*

Adult est une base générée par R. Kohavi à partir de *Census-income* par extraction d'environ 10% de la base et suppression de certains attributs. Son principal interêt est de permettre l'évaluation de nombreux autres systèmes ne pouvant traiter le volume de données de *Census-Income*. Dans ce cas encore, $\mathcal{SIAO}1$ se classe correctement sans nécessiter aucun biais d'évaluation particulier adapté aux bases bruitées ou de grande taille.

[12]Conformément aux conditions d'expérimentation préconisées par R. Kohavi, les couvertures des hypothèses sont évaluées sur les occurrences des instances et aucun mécanisme de pondération n'est mis en place.

Remarque :

Les bases *Adult* et *Census-income* étant très fortement bruitées, le nombre de noyaux développés pour caractériser le concept `Sup_50K` a été fixé à 100, et les implantations parallèles détaillées au chapitre suivant ont été utilisées. Dans le cas de *Census-income*, la durée d'un apprentissage est d'environ 5 jours sur un ensemble de 6 machines dont la fréquence des processeurs varie de 500 à 733 MHz. La durée d'apprentissage pour la base *Adult* est d'environ 8 heures et demie dans les mêmes conditions. Dans les deux cas, ces machines étaient partagées avec d'autres utilisateurs : les durées, fournies à titre indicatif, sont donc sur-estimées.

4.6.4 Analyse des résultats expérimentaux

Les résultats ont montré que $\mathcal{SIAO}1$ peut fournir des règles de qualité satisfaisante sur une très grande variété de problèmes, qu'ils soient de nature symbolique ou numérique, fortement relationnels ou propositionnels, y compris lorsque les volumes de données sont importants. De plus, le seul paramètre ayant été modifié en fonction des applications rencontrées est le nombre de noyaux. Ce dernier n'a pas une importance critique puisque la complexité de $\mathcal{SIAO}1$ augmente dans le pire des cas linéairement en fonction de ce paramètre.

4.7 Bilan

Nous avons décrit dans ce chapitre le système $\mathcal{SIAO}1$, qui propose une solution originale au problème de l'apprentissage supervisé relationnel, fondée sur une recherche évolutionnaire, ascendante et dirigée par les données dans l'espace des formules du langage $\mathcal{L}_h$. Il a pour principales caractéristiques :

- De pouvoir être appliqué à la fois dans le contexte de l'apprentissage relationnel, qui est caractérisé par la prépondérance de l'aspect symbolique, et dans le contexte attribut-valeur où les données numériques jouent un rôle plus important. La variété des problèmes traités montre une bonne adaptation du système à des contextes très variés.

- Une évolution contrôlée par les données de la structure des hypothèses qui n'est pas limitée à la forme des exemples ou à un modèle fourni par l'utilisateur. En particulier, les résultats expérimentaux ont été obtenus sans que l'utilisateur ait a spécifier les biais d'évaluation (la fonction de qualité est toujours la même) ou de recherche (il n'est pas nécessaire de fournir la forme des hypothèses ou de guider à l'aide de règles basées sur la syntaxe la construction de la solution).

- La prise en compte de connaissances du domaine de type symbolique et numérique.

$\mathcal{SIAO}1$ est donc polyvalent et sollicite peu l'utilisateur en ce qui concerne des questions

assez techniques telles que les biais de recherche ou d'évaluation.

Chapitre 5

Parallélisme et répartition

5.1 Problématique

L'application des algorithmes d'apprentissage à des problèmes de grande taille [Dietterich97] est un thème de recherche qui s'est particulièrement développé ces dernières années. L'enjeu est en effet important car les applications de tels systèmes dans des domaines en développement rapide comme par exemple l'extraction de connaissances à partir des données ne sont envisageables dans la plupart des cas que s'il est possible d'exploiter des quantités importantes de données en un temps raisonnable.

Plus généralement, l'étude de la capacité des algorithmes d'apprentissage automatique à *changer d'échelle* est centrale dans beaucoup d'aspects de la recherche en apprentissage automatique [Provost et al.96].

Du point de vue du *changement d'échelle en espace*, le défi vient du fait que les techniques d'échantillonnage, bien qu'utiles, ont montré leurs limites dans de nombreux cas. En effet, ces dernières occasionnent la plupart du temps une perte globale de précision qui peut être expliquée par le fait que plus on supprime d'information, plus on augmente les risques de sur-apprentissage. Par ailleurs, ces techniques rendent plus difficile l'apprentissage des cas particuliers qu'elles réduisent (d'un point de vue statistique) au bruit.

Du point de vue du *changement d'échelle en temps*, de nouveaux domaines nécessitent des systèmes très rapides. En ce qui concerne l'apprentissage automatique interactif, dans lequel un humain et un système d'apprentissage automatique interagissent en temps réel, la contrainte provient du temps de réponse qu'un utilisateur est prêt à supporter. Dans d'autres domaines tels que la sélection automatique de biais, pour laquelle il est nécessaire d'obtenir des résultats statistiquement significatifs portant sur de nombreuses exécutions pour différents biais de plusieurs systèmes d'apprentissage, la contrainte provient de l'explosion combinatoire qui limite d'autant la durée d'un apprentissage.

La parallélisation de $\mathcal{SIAO}1$ a donc pour principal objectif de faire bénéficier autant que possible ce système des possibilités de changement d'échelle offertes par les machines

parallèles, afin d'étendre par exemple son utilisation aux applications nécessitant l'exploitation de volumes de données importants.

Ce chapitre commence par une présentation du domaine du parallélisme et introduit les définitions des concepts qui serviront de base à la description et à l'évaluation des différentes approches des travaux étudiés et réalisés. Un certain nombre de résultats théoriques généraux relatifs aux performances des algorithmes répartis permettront de préciser la problématique et de proposer des mesures destinées à quantifier la réussite (ou l'échec) d'une architecture au regard des objectifs fixés. Les algorithmes d'évolution répartis sont ensuite présentés à l'aide d'une classification générale des architectures couramment rencontrées, ainsi que par une description plus spécifique de différents travaux représentatifs. Le paragraphe 5.4 présente le cadre du travail qui a mené à l'implantation des deux architectures étudiées. Les résultats expérimentaux sont ensuite fournis, puis l'étude d'un modèle de prédiction du comportement des architectures est réalisée.

5.2 Concepts fondamentaux

5.2.1 Les sources de parallélisme

Ce paragraphe regroupe les principaux critères permettant de catégoriser l'architecture des machines parallèles. Ce sont ces critères qui vont guider l'approche choisie, celle-ci devant exploiter au mieux la structure de l'algorithme à paralléliser ainsi que les caractéristiques de la machine parallèle.

La classification de Flynn

Flynn a proposé dans [Flynn66] une classification des systèmes répartis prenant pour critère la façon dont les instructions et les données sont réparties sur les processeurs. Cette classification regroupe donc quatre catégories de machines, puisque les flots de données ou d'instructions peuvent être chacun unique ou multiple. Le cas le plus simple, **SISD** (*Single Instruction stream, Single Data stream*), dans lequel la machine travaille sur un unique flux de données et un unique flux d'instructions, correspond à la machine séquentielle classique de von Neumann. Les trois autres cas (non triviaux) sont :

SIMD *Single Instruction stream, Multiple Data stream*
Cette classe regroupe les machines possédant une unique unité de contrôle appelée séquenceur qui distribue les données aux processeurs, chaque processeur exécutant la même instruction. On parle alors de machine vectorielle (*array processor*), ces systèmes étant principalement destinés à effectuer des traitements sur les vecteurs.

MISD *Multiple Instruction stream, Single Data stream*
Les systèmes MISD implantent le parallélisme en faisant travailler les processeurs à la chaîne le long de l'unique flux de données. Ce fonctionnement est plus connu sous le nom évocateur de *pipeline*.

MIMD *Multiple Instruction stream, Multiple Data stream*
Ce type d'architecture est de loin la plus courante. Chaque processeur exécute son propre programme à partir de ses données, rangées dans une mémoire pouvant être locale ou au contraire commune, et les communications se font de manière asynchrone.

L'intérêt de cette classification est qu'elle fait ressortir également les différentes manières d'envisager l'exploitation du parallélisme intrinsèque d'un algorithme.

Autres critères

Il existe trois autres caractéristiques qui permettent de situer de manière encore plus précise l'approche de parallélisme envisagée :

1. **L'organisation de la mémoire (ou couplage)**
   Le couplage rend compte du degré d'interaction entre les différentes unités élémentaires. Par exemple on parle de systèmes fortement couplés lorsque les processeurs travaillent sur une mémoire partagée, c'est-à-dire une mémoire directement adressable par tout processeur. Inversement, on parle de systèmes faiblement couplés dans le cas des machines à mémoires distribuées, pour lesquelles chaque processeur dispose d'une mémoire propre.
2. **Le degré de parallélisme**
   Le degré de parallélisme est une évaluation du nombre de tâches élémentaires exécutées simultanément. Cette mesure permet d'estimer l'ordre de grandeur du changement d'échelle qu'on pourra envisager ; ainsi le degré de parallélisme maximal donne une borne supérieure au nombre de processeurs qu'on utilisera, alors que le degré de parallélisme moyen donne une idée du nombre de processeurs nécessaires afin de conserver un bon rapport performances / coût.
3. **Le grain de parallélisme**
   On le définit habituellement comme la taille moyenne des tâches élémentaires qui ont guidé la parallélisation [Gengler et al.96]. De ce point de vue idéal, qui envisage un utilisateur pouvant construire ou se procurer une machine dédiée à son projet, on peut détailler ce critère dans le tableau suivant :

Grain	*Tâches élémentaires*	*Échelle*
gros	programmes, procédures	système réparti
moyen	instructions, expressions	variable
fin	opérateurs, bits	microprocesseur

   Cependant, du point de vue plus courant d'un utilisateur qui doit travailler dans un environnement donné, et n'agit par conséquent qu'au niveau logiciel et non matériel, le grain est étroitement associé au degré de parallélisme. Ainsi, dans la littérature concernant les algorithmes d'évolution parallèles, le grain fin correspond aux machines massivement parallèles, dont le nombre important de processeurs est mis à profit la plupart du temps en répartissant la population sur

ceux-ci [Spiessens et al.91, Manderick et al.89], avec dans le cas extrême un individu par processeur. Le problème de surcoût dû aux communications devient alors très important.

Contexte actuel

La plupart des machines parallèles actuelles sont construites autour de processeurs ordinaires et utilisent des réseaux de communication à très haut débit (il s'agit donc de machines de type MIMD). Cette tendance, qui s'est confirmée de manière très significative ces dernières années peut se justifier en partie par le gain en termes de coût de production de la machine et de portabilité des programmes.

D'autre part, le développement effréné des réseaux a vu la multiplication de machines parallèles d'un type nouveau, constituées de stations de travail ou de micro-ordinateurs inter-connectés. Ces ordinateurs-réseaux (*network computers*), plus polyvalents et de coût plus faible que les machines spécialisées peuvent être classés dans la catégorie des machines MIMD hétérogènes[1] faiblement couplées. Compte tenu des débits assurés par les réseaux actuels, seul un parallélisme à grain épais est envisageable sur ce type de machines.

En conséquence, il n'est pas étonnant de constater qu'un grand nombre des travaux concernant les AE répartis sont validés, pour leur partie expérimentale, sur des machines MIMD. Pour des raisons à la fois scientifiques et pratiques, la parallélisation de $\mathcal{SIAO}1$ a donc été envisagée, puis réalisée et étudiée sur ce type de machine.

5.2.2 La loi d'Amdahl

La loi d'Amdahl est centrale en parallélisme car non seulement elle borne les performances des algorithmes répartis, mais surtout parce qu'elle fournit un cadre théorique simple permettant d'étudier et de modéliser leur comportement en pratique. C'est à ce cadre qu'il sera fait référence tout au long de ce chapitre lorsqu'il s'agira de quantifier l'intérêt d'une architecture parallèle. Dans un souci de simplification, on suppose par la suite que les caractéristiques matérielles des processeurs (machines élémentaires) sont strictement identiques.

Accélération et efficacité

Soit $T_p(n)$ le temps nécessaire à l'algorithme étudié pour résoudre une instance de taille n avec p processeurs, et $M_p(n)$ la quantité de mémoire requise.

Définition 5.1 *Facteur d'accélération d'un algorithme parallèle*
On appelle accélération le rapport $A(n,p) = \frac{T_1(n)}{T_p(n)}$.

[1] puisque les processeurs, les capacités mémoire et disque, voire même les systèmes d'exploitation varient souvent d'une machine à l'autre.

Définition 5.2 *Efficacité d'un algorithme parallèle*
On appelle efficacité le rapport $E(n,p) = \frac{A(n,p)}{p} = \frac{T_1(n)}{p.T_p(n)}$.

Propriétés

Un raisonnement souvent utile en parallélisme, appelé *argument de la simulation*, permet de borner grossièrement l'accélération et l'efficacité de tout algorithme réparti. Cet argument consiste à considérer la possibilité de simuler un algorithme réparti destiné à utiliser p processeurs identiques sur une unique machine monoprocesseur fonctionnant en temps partagé. En supposant que les temps de commutation entre processus sur la machine monoprocesseur sont inférieurs aux temps de communication entre processeurs sur la machine parallèle d'une part, et que les temps de traitement de la machine monoprocesseur et de l'un des p éléments de la machine parallèle sont identiques (processeurs identiques, mémoire suffisante) d'autre part, cet argument montre que l'accélération est toujours plus faible ou dans le cas idéal égale au nombre de processeurs. En effet, si un algorithme sur-linéaire existait, il suffirait de le simuler sous les conditions précitées pour montrer que tel n'est pas le cas. Sous les hypothèses (très raisonnables) de cet argument, on a donc les propriétés suivantes :

Propriété 5.1
L'accélération est au mieux linéaire en fonction du nombre de processeurs.

Propriété 5.2
L'efficacité ne peut dépasser l'unité.

La loi d'Amdahl

La loi d'Amdahl [Amdahl67] fait l'hypothèse que les algorithmes parallèles et séquentiels exécutent les mêmes instructions (hormis la surcharge due à la mise en parallèle que l'on peut négliger puisqu'on cherche une borne supérieure au gain de performance), chaque instruction étant exécutée en un temps unitaire.

Loi 5.1 *Amdahl*

Soit $T_1(n) = T_{Seq}(n) + T_{Par}(n)$ *; avec :*

- $T_{Seq}(n) = f(n).T_1(n)$ *; temps pris pour exécuter la partie de l'algorithme ne pouvant être parallélisée.*
- $T_{Par}(n) = (1 - f(n)).T_1(n)$ *; temps pris pour exécuter la partie de l'algorithme qui peut être parallélisée.*

Alors, le meilleur temps d'exécution possible sur p processeurs (noté T_p^**) est obtenu lorsqu'on répartit de manière optimale la partie de l'algorithme qui peut être parallélisée sur tous les processeurs disponibles :*

$$T_p^*(n) = T_{Seq}(n) + \frac{T_{Par}(n)}{p} = (f(n) + \frac{1-f(n)}{p}).T_1(n) \quad \textbf{(Amd 1)}$$

De plus, on a :

- $0 \leq A(n,p) \leq \frac{1}{f(n)+(1-f(n))/p}$ **(Amd 2)** *et* $\lim_{p\to\infty} A(n,p) = \frac{1}{f(n)}$.
- $0 \leq E(n,p) \leq \frac{1}{1+(p-1).f(n)}$ **(Amd 3)** *et pour n fixé* $\lim_{p\to\infty} E(n,p) = 0$.

Conséquence

Comme le montre la figure 5.1 ci-après, une des conséquences de la loi d'Amdahl est que le taux d'instructions ne pouvant être réparties sur la machine parallèle $f(n)$ doit être extrêmement faible dès que le nombre de processeurs devient assez important si l'on veut conserver une efficacité correcte.

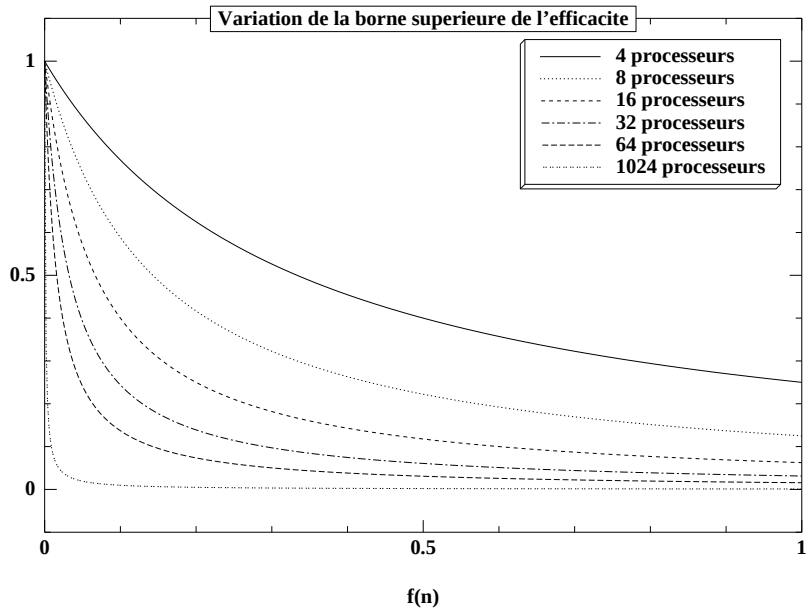


FIG. 5.1: Comportement de la borne supérieure de l'efficacité prédite par la loi d'Amdahl.

Plus précisément, en se plaçant dans le cas le plus favorable - lorsque la borne supérieure de l'inéquation **(Amd 3)** est atteinte - on remarque que pour conserver une efficacité constante, $f(n)$ doit être inversement proportionnelle au nombre de processeurs, soit $f(n) = \frac{1}{p-1}.(\frac{1}{E} - 1)$. Le tableau 5.1 illustre cette contrainte en donnant une estimation de $f(n)$ pour différentes valeurs de E lorsqu'on fait varier le nombre p de processeurs.

Cette contrainte très forte limite *a priori* toute approche de parallélisme massif aux algorithmes dont la quasi-totalité du code peut être reparti.

Processeurs	4	8	16	32	64	1024
$f(n)/E = 0,95$	$1,75\ 10^{-2}$	$7,52\ 10^{-3}$	$3,51\ 10^{-3}$	$1,70\ 10^{-3}$	$8,35\ 10^{-4}$	$5,14\ 10^{-5}$
$f(n)/E = 0,9$	$3,70\ 10^{-2}$	$1,59\ 10^{-2}$	$7,41\ 10^{-3}$	$3,58\ 10^{-3}$	$1,76\ 10^{-3}$	$1,09\ 10^{-4}$
$f(n)/E = 0,8$	$8,33\ 10^{-2}$	$3,57\ 10^{-2}$	$1,67\ 10^{-2}$	$8,06\ 10^{-3}$	$3,97\ 10^{-3}$	$2,44\ 10^{-4}$
$f(n)/E = 0,5$	$3,33\ 10^{-1}$	$1,43\ 10^{-1}$	$6,67\ 10^{-2}$	$3,22\ 10^{-2}$	$1,59\ 10^{-2}$	$9,77\ 10^{-4}$

TAB. 5.1: Variation du taux d'instructions non parallélisables maximum permettant de conserver une efficacité donnée en fonction du nombre de processeurs.

5.2.3 Conclusion

Cette brève introduction au parallélisme a été l'occasion de détailler les caractéristiques des systèmes parallèles, puis de définir plus formellement les concepts d'accélération et d'efficacité. L'argument de la simulation, qui envisage la possibilité de simuler un algorithme parallèle sur une machine monoprocesseur fonctionnant en temps partagé permet de poser[1] que la justification de toute parallélisation ne se valide que sur le critère de gain de performance (en termes d'accélération ou d'efficacité) ; de plus, il fournit[1] des bornes absolues concernant l'efficacité et l'accélération maximum. Enfin, la loi d'Amdahl montre comment calculer une borne supérieure plus précise à l'accélération et l'efficacité qu'on peut attendre d'un algorithme parallèle en fonction du taux d'instructions parallélisables et du nombre de processeurs. Bien que ne prenant pas en compte de nombreux paramètres, comme par exemple le surcoût dû aux communications, cette loi montre que le parallélisme massif ne peut être envisagé que pour une classe d'algorithmes particuliers, dont le taux d'instructions non parallélisables est infinitésimal.

5.3 Les algorithmes d'évolution répartis

Les algorithmes d'évolution étant utilisés intensivement dans des domaines relevant de l'optimisation, leur parallélisation a fait l'objet de nombreuses études dans le but d'augmenter la taille des problèmes traités ou de diminuer le temps de calcul. La structure générale de ces algorithmes contient en effet implicitement un fort parallélisme tant au niveau des opérations de manipulation et d'évaluation des individus qu'au niveau de la gestion de la population. De plus, l'implantation et l'utilisation de telles solutions ont été facilitées par le considérable développement ces dernières années, au sein des entreprises ou des laboratoires, de réseaux locaux permettant la mise en commun des moyens de calcul.

5.3.1 Une classification des architectures

Une vaste majorité des travaux dans le domaine des algorithmes d'évolution répartis concerne les algorithmes génétiques ; c'est pourquoi la classification proposée ci-dessous se base, comme beaucoup d'autres [Goldberg89, CP97b, Doorly95] sur une réorganisation

[1]Sous réserve que ses hypothèses (très peu contraignantes) soient vérifiées.

et une augmentation des quatre prototypes d'algorithmes génétiques parallèles envisagés par Grefenstette dans [Grefenstette81].

L'organisation proposée s'oriente tout d'abord en fonction du choix du type de gestion de la mémoire de l'algorithme. Dans le cas d'une mémoire (population) unique, on s'oriente vers une architecture centralisée de type maître-esclave, alors que lorsque chaque processeur travaille sur une mémoire qui lui est propre, on obtient une architecture en réseau. Ce choix est particulièrement important car il détermine si la version séquentielle de l'algorithme va être intégralement conservée (cas de l'architecture centralisée), ou si au contraire l'algorithme va évoluer en autre chose (cas de l'architecture en réseau). Dans ce dernier cas, une attention toute particulière doit être portée sur la méthodologie mise en œuvre pour étudier le nouvel algorithme obtenu.

5.3.2 Les architectures maître-esclave

Comme le montre la figure 5.2, ces architectures sont centrées autour d'un processus, appelé maître, qui gère l'unique population et distribue le travail aux autres processus appelés esclaves. La mémoire étant commune, les opérateurs d'évolution travaillent exactement dans les mêmes conditions que l'algorithme séquentiel. Le critère d'évaluation de ce type d'architecture est donc tout simplement l'accélération qu'elle produit, puisque le comportement de l'algorithme parallèle et celui de l'algorithme séquentiel sont strictement identiques.

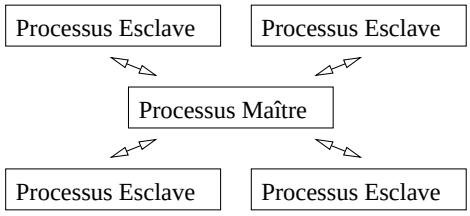


FIG. 5.2: L'architecture maître-esclave.

Dans ce cadre, la vitalité du processus maître est essentielle car si ce dernier perd pied, il ralentit l'ensemble du système. Cette architecture n'est donc envisageable que lorsque la durée de l'évaluation d'un individu est longue par rapport au temps pris pour l'envoyer sur le réseau et gérer la population.

Les sous-paragraphes suivants détaillent les trois types d'architectures maître-esclave que l'on peut rencontrer (du plus simple au plus évolué), et détaillent leurs avantages et inconvénients.

Maître-esclave synchrone

L'architecture maître-esclave synchrone est à la fois la plus simple et la plus répandue

des architectures maître-esclave. Au début de chaque génération, le processus maître sélectionne les individus pour la reproduction et leur applique les opérateurs d'évolution, puis il distribue ces derniers aux esclaves pour évaluation. Enfin, il attend que l'ensemble des résultats lui aient été retournés avant de recommencer. Il y a donc synchronisation sur le *top* du compteur de générations.

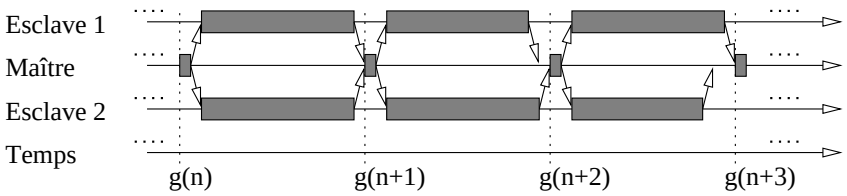


FIG. 5.3: Architecture maître-esclave synchrone.

Le temps est ici à l'échelle des durées de traitement et de transmission de la population.

La figure 5.3 détaille l'utilisation des processeurs d'un système maître - deux esclaves typique au cours du temps (porté en abscisse). Les rectangles grisés le long des axes dénotent un travail de la part du processus correspondant ; pour le processus maître, il s'agit de la sélection des individus et de l'application des opérateurs d'évolution afin de créer la nouvelle population, tandis que pour les processus esclaves, il s'agit de l'évaluation des individus transmis. Les flèches dénotent des communications entre processus (envoi et réception des individus et de leur qualité).

En contrepartie de sa simplicité d'implantation, cette approche montre certaines faiblesses du point de vue de son efficacité :

- Il est très difficile de répartir efficacement la charge de travail sur les différents esclaves. En effet, dans le cas d'un environnement partagé dynamique, d'autres utilisateurs peuvent à tout moment modifier le taux d'utilisation des processeurs du réseau lorsqu'ils effectuent d'autres travaux. De plus, à supposer que le système soit uniquement dédié à l'algorithme d'évolution, il faut également que la variance du temps d'évaluation des individus soit faible.

- Par ailleurs, comme le montre la figure 5.3, les esclaves sont inactifs pendant que le maître gère la population au début de chaque génération.

Comme on va le voir, les deux autres types d'architectures maître-esclave remédient à ces inconvénients.

Maître-esclave asynchrone

Les modifications permettant d'aboutir au mode asynchrone sont les suivantes :

- Les esclaves traitent les individus un par un et non plus par lots : dès qu'un individu est évalué, il est renvoyé au maître avec sa qualité mise à jour.
- De même, le maître gère les individus « à la volée » : dès qu'il reçoit un individu, le maître en renvoie un à l'esclave correspondant.

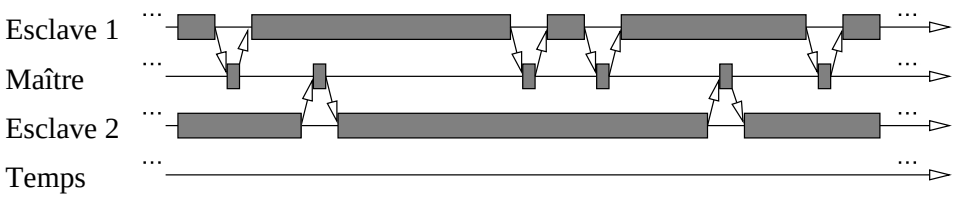


FIG. 5.4: Architecture maître-esclave asynchrone.

Le temps est ici à l'échelle des durées de traitement et de transmission des individus.

Cette architecture fait donc disparaître l'attente bloquante de la terminaison des esclaves. Dans le cas des algorithmes générationnels évaluant toute la population, cette méthode permet d'équilibrer dynamiquement la charge en fonction du travail qu'est capable de fournir chaque processeur ; cependant l'attente bloquante de la terminaison du maître lorsque ce dernier crée la nouvelle population est toujours présente. Par contre, dans le cas des algorithmes incrémentaux, si le coût de gestion *d'un* individu est assez faible par rapport au temps nécessaire pour l'évaluer, alors il n'y a plus d'attente liée à la terminaison du maître (voir figure 5.4).

Il faut enfin remarquer que l'asynchronisme, qui permet d'un côté de lever des contraintes assez fortes au niveau de l'attente entre processus, introduit en contrepartie un « degré d'incertitude » supplémentaire d'une part, et un risque de déséquilibre d'exploitation entre individus d'autre part :

- Le *degré d'incertitude supplémentaire* se constate de la manière suivante : lorsqu'on fournit à un algorithme d'évolution à architecture séquentielle ou maître-esclave synchrone un même noyau lui permettant de générer une suite de nombres pseudo-aléatoire donnée, on obtient toujours des déroulements d'exécutions strictement identiques. Or cette propriété n'est plus valable dans le contexte asynchrone, puisque des éléments extérieurs, comme l'accroissement de la charge d'un processeur, peuvent modifier le comportement de l'algorithme. Cette modification, aussi insignifiante soit-elle (par exemple la réception - et donc l'insertion - d'un individu avant ou après un autre), a des répercussions sur l'état du système et donc sur son déroulement.
- Le *risque de déséquilibre d'exploitation* se produit lorsqu'il existe une très forte variance du temps d'évaluation des individus. En effet, cette variance introduit une distorsion qui provoque une meilleure exploitation de l'espace couvert par les individus évalués rapidement. Dans un cas extrême, ce déséquilibre pourrait causer

une modification du comportement de l'algorithme maître-esclave asynchrone par rapport au comportement de l'algorithme séquentiel original.

Maître-esclave asynchrone avec mémoire partagée

Il est possible d'accroître la proportion des instructions exécutées en parallèle de l'architecture asynchrone en transférant aux esclaves la charge d'appliquer les opérateurs d'évolution modifiant les individus (par exemple, dans le cas des algorithmes génétiques, il s'agit des opérateurs de mutation et de croisement). Le maître se comporte alors comme une mémoire partagée par les esclaves, et ne procède plus qu'à l'extraction et au remplacement des individus.

Bien que séduisant à première vue, ce type d'architecture a deux inconvénients qui doivent être examinés avant d'envisager toute implantation, et ce même sur les machines à mémoire partagée qui semblent particulièrement adaptées :

1. Dans le cadre des algorithmes générationnels, cette approche est souvent inutile voire pénalisante. En effet, d'une part la phase de création de la nouvelle population s'exécute à l'exclusivité de toute autre tâche, et d'autre part les opérateurs d'évolution modifiant les individus sont presque toujours extrêmement simples (il suffit de considérer par exemple la mutation et le croisement génétiques classiques). Le temps nécessaire pour envoyer et recevoir les individus par le réseau est alors bien supérieur au temps mis par le maître pour appliquer ces mêmes opérateurs.

2. Les opérateurs d'évolution plus complexes dépendent souvent de données autres que l'individu auquel ils s'appliquent, ce qui peut également être un inconvénient pour leur mise en parallèle.

5.3.3 Les architectures en réseau

D'après ce qui a été dit dans les paragraphes précédents d'une part, et la loi d'Amdahl d'autre part, les limites des architectures maître-esclave sont généralement atteintes :

- lorsque le temps d'évaluation d'un individu est trop court par rapport aux temps de communication et de gestion de la population ;

- lorsqu'on augmente le degré de parallélisme au delà des contraintes prévues (par exemple lors du portage du système sur des machines massivement parallèles).

Dans ces cas extrêmes, il faut essayer de paralléliser la partie non encore répartie de l'algorithme qui comprend la gestion de la mémoire centralisée via les opérations de sélection et de remplacement des individus. Il est cependant très difficile de répartir efficacement la sélection car cet opérateur nécessite le plus souvent des renseignements concernant l'ensemble de la population. Par exemple, dans le cas des algorithmes génétiques, la sélection proportionnelle à la qualité et ses nombreuses variantes nécessitent la connais-

sance de la somme des qualités des individus alors que les méthodes basées sur le rang nécessitent la connaissance de leur ordre.

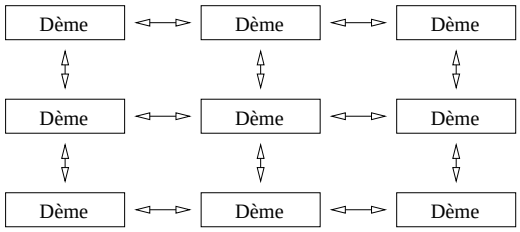


FIG. 5.5: L'architecture en réseau.

La topologie du réseau est ici une grille.

Ces contraintes ont conduit au développement des architectures en réseau, qui sont caractérisées par une répartition de la mémoire sur les processeurs grâce au déploiement de processus identiques gérant localement leur propre population, appelée *dème*. Dans ce contexte, la conservation du comportement de l'algorithme séquentiel n'est plus garantie, par exemple [De jong et al.95] puis [Sarma et al.97] montrent dans le cadre des algorithmes génétiques qu'à schéma de sélection identique, la pression sélective globale obtenue au niveau de la version parallèle est toujours inférieure à celle de la version séquentielle (ce phénomène peut être expliqué par le temps de propagation sur les dèmes du réseau des meilleurs individus). L'étude du nouvel algorithme obtenu doit alors prendre en compte deux caractéristiques liées à son implantation : la taille des dèmes et le type des communications.

Remarque méthodologique

Un certain nombre de travaux[2] concernant les AE répartis présentent des résultats faisant état d'accélérations sur-linéaires. Ces résultats, en contradiction apparente avec ce qui a été présenté antérieurement dans le cadre général des algorithmes répartis, peuvent être expliqués par un protocole expérimental ne respectant pas les contraintes relatives à la définition canonique de ce concept.

Plus généralement, il est important de faire attention aux points suivants lors de l'évaluation des performances d'une architecture :

1. Tout d'abord les questions d'efficacité et de qualité ne doivent jamais être traitées indépendamment. En effet, la définition de l'accélération suppose que l'algorithme parallèle et l'algorithme séquentiel travaillent sur le même problème et fournissent la même solution. En conséquence, l'étude de la variation du temps de terminaison de l'algorithme lorsque le nombre de processeurs varie doit se faire à qualité de solution obtenue constante.
2. De même, la question du paramétrage joue un rôle important. Par exemple, dans le cas des AE répartis suivant une architecture en réseau, les caractéristiques de l'algorithme dépendent étroitement du nombre de processeurs, ce qui fausse la signification des mesures. En effet, le cadre théorique ne permet pas de justifier la comparaison du même AE générique utilisant soit un processeur et un dème unique de n individus, soit p processeurs, et p dèmes de n individus ou encore p dèmes de n/p individus, puisqu'il s'agit alors d'algorithmes différents[3]. Une manière de procéder consisterait alors à considérer non une instance de l'algorithme dont les paramètres seraient fixés, mais la classe de toutes ces instances, ce qui implique qu'il faudrait trouver pour chaque configuration les meilleurs paramètres afin de procéder à une comparaison. Bien que des travaux théoriques [CP98] puissent être exploités à ces fins, ce point de vue n'a pas été mis en pratique.

[2] Ces derniers sont d'autant plus nombreux qu'ils sont anciens.

[3] Par contre l'argument de la simulation qui permet toujours de travailler à algorithme identique garantit encore que quels que soient les paramètres, l'accélération ne pourra jamais être sur-linéaire.

3. Enfin, les conditions expérimentales doivent également garantir autant que possible des éléments de calcul identiques en termes de puissance, d'espace mémoire et de capacité de communication. Ces facteurs doivent être constants au cours du temps, l'étude de la robustesse de l'architecture aux variations de charge, si elle a lieu, devant être faite dans un second temps.

En particulier, dans le cas des AE répartis suivant une architecture maître-esclave, il est souvent facile de montrer que le fonctionnement global de l'algorithme reste inchangé. En conséquence la qualité des solutions obtenues d'une part, et le paramétrage de l'algorithme d'autre part ne nécessitent pas d'étude particulière. Par contre, le cas des AE répartis suivant une architecture en réseau est différent et plus complexe.

En conclusion, il est important de s'assurer d'après la description du contexte expérimental que la définition normative des termes accélération et efficacité est effectivement utilisée lorsque l'on veut évaluer une nouvelle approche.

5.3.4 Modélisation

Parmi les travaux concernant la modélisation dans un cadre aussi général que possible du comportement des AE répartis, on peut citer dans le domaine des algorithmes génétiques [Pettey et al.89] qui étudie la conservation du théorème des schémas dans les architectures en réseau. Toujours dans le cadre des architectures en réseau [CP et al.97] modélise l'accélération obtenue pour deux topologies extrêmes (cas des dèmes totalement isolés ne communiquant pas, ou des dèmes totalement inter-connectés avec un taux de migration maximum). Ces travaux sont suivis dans [CP98] par l'étude des relations liant le degré de connexion, le taux de migration, la taille des dèmes et la qualité de la solution obtenue.

En ce qui concerne la modélisation des architectures maître-esclave dont il sera particulièrement question dans la suite, on présente brièvement l'analyse donnée par [CP97a] à laquelle il sera fait référence. Cette analyse se place dans le contexte – commun à la quasi-totalité des architectures maître-esclave développées pour les AE – de l'architecture maître-esclave synchrone, dans laquelle le maître, gérant un algorithme de type générationnel, répartit à chaque génération l'évaluation de la population sur les esclaves. On néglige le temps pris à chaque itération par le maître pour générer la nouvelle population. On note :

- p le nombre d'esclaves ;
- n la taille de la population ;
- α le temps d'évaluation d'un individu ;
- $t_{c(pop)}$ le temps de communication nécessaire pour transmettre la fraction de la population à évaluer (sens maître-esclave) ou évaluée (sens esclave-maître) ;

- t_{eval} le temps nécessaire à un esclave pour évaluer la fraction de la population qui lui a été transmise ; on a : $t_{eval} = \frac{n.\alpha}{p}$.

Le premier esclave reçoit les données à l'instant $t_{c(pop)}$ et termine à $t_{c(pop)} + t_{eval}$. Comme d'autre part le maître a fini l'envoi des données à l'instant $p.t_{c(pop)}$, on en déduit que la durée d'inactivité du maître est : $t_i = t_{c(pop)} + t_{eval} - p.t_{c(pop)}$, soit :

$$t_i = t_{eval} - (p-1).t_{c(pop)} \qquad \textbf{(M-E Sync 1)}$$

Ainsi, le temps total nécessaire au maître pour traiter une génération est : $t_{gen} = 2.p.t_{c(pop)} + t_i = 2.p.t_{c(pop)} + t_{eval} - (p-1).t_{c(pop)} = (p+1).t_{c(pop)} + t_{eval}$, soit :

$$t_{gen} = (p+1).t_{c(pop)} + \frac{n.\alpha}{p} \qquad \textbf{(M-E Sync 2)}$$

Cependant, le temps de communication n'est pas constant puisqu'on peut supposer qu'il dépend linéairement de la quantité d'information à transmettre, soit dans ce cas : $t_{c(pop)} = a.n.l/p + c$; où a et c sont des constantes dépendantes du matériel, et l est la longueur (en bits) du codage d'un individu. On obtient finalement :

$$t_{gen} = (p+1).(a.n.l/p + c) + \frac{n.\alpha}{p} \qquad \textbf{(M-E Sync 3)}$$

Le modèle a été validé expérimentalement, et prédit correctement les performances de ce type d'algorithme réparti une fois déterminés les multiples paramètres (α, l, a et c). Comme d'autre part à partir d'un certain nombre d'esclaves les performances vont décroître (puisque la diminution du temps de calcul au niveau des esclaves ne compensera plus le coût croissant des communications au niveau du maître), il est possible de l'utiliser pour estimer le nombre d'esclaves optimal, noté p^*, (en résolvant l'équation $\frac{\partial t_{gen}}{\partial p} = 0$) ; on obtient alors :

$$p^* = \sqrt{\frac{(a.l+\alpha).n}{c}} \qquad \textbf{(M-E Sync 4)}$$

5.4 Parallélisation de $\mathcal{SIAO}1$

5.4.1 Objectifs

Les buts de la mise en parallèle de $\mathcal{SIAO}1$ découlent logiquement des résultats présentés au paragraphe 5.2 qui concernent les algorithmes répartis en général, ainsi que de contraintes plus générales relatives à l'implantation et au domaine d'application. Ils permettent de préciser, au vu de ce qui a été présenté, la problématique du paragraphe 5.1.

L'objectif principal est de permettre l'utilisation de $\mathcal{SIAO}1$ sur des bases de données de grande taille

Puisque l'algorithme converge en un temps très raisonnable sur des bases comportant quelques centaines d'instances (par exemple, moins d'une minute est nécessaire sur un micro-ordinateur pour traiter la base *Iris*), la parallélisation a pour objectif une utilisation de l'algorithme dans le cadre de l'ECD sur des bases importantes. Il s'agit donc de permettre un changement *d'échelle* en temps ou en espace aussi performant que possible.

L'architecture doit conserver les propriétés de fonctionnement de l'algorithme séquentiel

Cette contrainte est centrale pour la cohérence de l'approche puisqu'il semble logique de vouloir conserver un algorithme élaboré pour être adapté au problème à résoudre. Le rappel de l'argument de la simulation appliqué à ce contexte précis, qui figure dans les lignes suivantes, permet une justification plus précise de cet objectif.

La parallélisation se valide uniquement par des critères de performance en temps ou en espace

Ce critère unique de validation provient directement de l'argument de la simulation. En effet, d'après cet argument, tout algorithme réparti[4] peut être simulé sur une machine monoprocesseur fonctionnant en temps partagé, et on peut donc en conclure que la comparaison d'un algorithme réparti dont le fonctionnement général serait différent de l'algorithme séquentiel servant de référence n'a pas lieu d'être. En effet, si l'algorithme séquentiel est moins bien adapté au problème que l'algorithme réparti, il y a alors une sorte de tromperie concernant la méthode de validation puisqu'il suffirait de simuler ce dernier pour obtenir un meilleur cas de référence. En conséquence, dans le cas où la qualité de la solution prime, et puisque l'algorithme de référence peut toujours être ajusté par l'argument de la simulation[1] sur le modèle de l'algorithme réparti, et donc fournir un résultat strictement identique, il ne reste comme critère d'évaluation qu'une comparaison des performances en temps ou en espace. En particulier, la notion de changement d'échelle, qui est au cœur de

[4] Les conditions d'application de cet argument étant extrêmement peu contraignantes, on suppose ici qu'elles sont réalisées.

la problématique de ce chapitre, doit être précisée et formalisée si possible.

L'architecture doit être aussi indépendante que possible de la machine physique

Une fois les contraintes de conservation des résultats de l'algorithme et de gain de performances en temps ou en espace remplies, on peut également souhaiter que l'architecture envisagée soit aussi indépendante que possible de la machine physique. Il est en effet préférable de favoriser l'adaptabilité de l'architecture à différentes machines MIMD, qu'il s'agisse d'un système homogène et spécialisé comme l'IBM SP2, ou plus simplement d'un réseau hétérogène de stations de travail. Pour répondre à ce besoin, des librairies de fonctions de gestion de processus et de leurs communications couplées à des interfaces ont été développées ces dernières années. PVM [Geist et al.94], qui a été utilisée dans le cadre de cette thèse, est à ce jour la librairie la plus répandue. De plus, il est préférable de rendre les paramètres de l'algorithme d'évolution (comme, par exemple, le nombre de dèmes) indépendants des caractéristiques de la machine disponible (comme, par exemple, le nombre de processeurs) afin qu'il soit possible d'exploiter au mieux les configurations matérielles disponibles et ce, quel que soit le problème à résoudre.

5.4.2 Formalisation de la notion de changement d'échelle

Changement d'échelle en espace et en temps

Le changement d'échelle est la mesure de la capacité d'un algorithme à travailler sur des problèmes de taille croissante lorsque l'on augmente les ressources en mémoire et en puissance de calcul mises à sa disposition. Dans le contexte des algorithmes répartis, l'accroissement des ressources correspond à l'augmentation du nombre p de machines identiques disponibles, ce qui conduit à proposer les définitions suivantes reprenant les notations établies au paragraphe 5.2.2 :

Définition 5.3 *Facteur de changement d'échelle en espace d'un algorithme parallèle*
On appelle facteur de changement d'échelle en espace le rapport $F_E(n,p) = \frac{T_1(n)}{T_p(p.n)}$.

Définition 5.4 *Facteur de changement d'échelle en temps d'un algorithme parallèle*
On appelle facteur de changement d'échelle en temps le rapport $F_T(n,p) = \frac{T_1(n)}{p.T_p(n)}$.

Tout comme le facteur d'accélération, le facteur de changement d'échelle définit un taux de rendement prenant comme référence une machine monoprocesseur. Dans le cas du facteur de changement d'échelle en espace, il s'agit d'étudier le comportement du programme parallèle lorsqu'on augmente la complexité du problème à résoudre proportionnellement au nombre de processeurs, alors que dans le cas du facteur de changement d'échelle en temps, on revient à la notion d'accélération qui se définit à problème constant.

Iso-efficacité

Une autre possibilité pour déterminer la capacité d'un algorithme à changer d'échelle consiste à étudier la relation existant entre la taille n du problème et le nombre p de processeurs lorsqu'on fixe l'efficacité à une constante donnée.

Définition 5.5 *Courbe d'iso-efficacité de niveau k_E*
On appelle courbe d'iso-efficacité de niveau k_E la courbe de niveau obtenue par l'intersection du plan d'équation $E = k_E$ avec la fonction $E(n,p)$ dans l'espace $\mathcal{E}(n,p,E)$.

Si il semble évident que lorsque le nombre de processeurs augmente, il est nécessaire d'accroître également la taille des problèmes afin de conserver une efficacité constante, différents algorithmes nécessiteront des accroissements plus ou moins importants. Or plus cet accroissement de la taille des instances sera important, plus l'algorithme étudié nécessitera des problèmes de taille importante pour fonctionner efficacement sur un grand nombre de processeurs. En conclusion la capacité de changement d'échelle de l'algorithme étudié sera d'autant plus réduite que l'accroissement de la taille des instances sera important sur la courbe d'iso-efficacité.

Point de vue expérimental

Un premier problème qui apparaît lors de la mesure expérimentale des grandeurs détaillées ci-dessus concerne la difficulté qu'il y a à définir en toute généralité la complexité n d'une instance. Une pratique, commune à de nombreux travaux du domaine et suivie ici, consiste à poser que n est proportionnelle à la taille de l'ensemble d'apprentissage. Ainsi, lorsqu'il devient nécessaire de faire varier n, on choisit une base d'apprentissage suffisamment grande , puis on obtient différentes valeurs de n en procédant à plusieurs échantillonnages.

D'autre part, il est important de noter que si, du point de vue de la faisabilité des mesures, les facteurs de changement d'échelle en espace et en temps peuvent être facilement déterminés, il en est tout autrement de l'iso-efficacité. En effet, la détermination des courbes d'iso-efficacité ne peut se faire qu'à l'aide d'un modèle suffisamment précis de $E(n,p)$, à moins de disposer de ressources permettant de faire un grand nombre de tests destinés à trouver pour chaque valeur de p la valeur de n associée qui permet de s'approcher autant que possible de l'efficacité recherchée.

Par ailleurs, une définition alternative de l'accélération, motivée et présentée dans [Gustafson88], pose également des problèmes quand à sa mise en œuvre. L'idée à la base de cette mesure consiste à prendre la réciproque exacte de l'accélération $\frac{T_1(n)}{T_p(n)}$ qui est définie à taille du problème n constante, alors qu'on fait varier le temps T d'exécution. On se place donc à temps d'exécution t constant, et l'on étudie le comportement de la taille N du problème que l'on peut traiter pendant la même durée avec p processeurs :

$$A'(t,p) = \frac{N_p(t)}{N_1(t)}$$

Bien que difficile d'application, la définition alternative du facteur de changement d'échelle en espace qui en découle est particulièrement adaptée à la problématique de ce chapitre :

$$F'_E(t,p) = \frac{N_p(t)}{p.N_1(t)}$$

En conclusion, on remarquera que le facteur de changement d'échelle en espace est un bon compromis puisqu'il est à la fois facile à mesurer et adapté au contexte de l'ECD dans lequel on dispose d'algorithmes séquentiels résolvant déjà en temps raisonnable des problèmes de taille raisonnable, et pour lesquels il s'agit de savoir si une fois répartis ils pourront résoudre en temps raisonnable des problèmes de taille importante.

5.4.3 Deux architectures adaptées à la fouille de données

Méthodologie générale

La démarche sous-tendant la conception de bien des algorithmes parallèles consiste à décomposer l'algorithme initial en sous-tâches à peu près équivalentes en termes de temps d'exécution. De plus, cette décomposition doit idéalement minimiser la partie séquentielle (qui ne pourra être distribuée) ainsi que les surcoûts liés aux communications (échanges d'informations entre les tâches et attentes bloquantes). Les paragraphes suivants présentent les deux architectures proposées et étudiées pour atteindre les objectifs exposés en 5.4.1.

Étude préliminaire

Une première étude du fonctionnement de l'algorithme à l'aide d'un optimiseur de code (`gprof`) montre d'une part que la grande majorité du temps d'exécution est passée à évaluer la couverture des définitions générées par l'algorithme, et d'autre part que la proportion du temps d'exécution total dédiée à cette évaluation augmente en même temps que la taille de la base d'apprentissage. Le tableau 5.2 détaille ce phénomène sur les bases *Iris* et *Mushrooms* en fonction de la taille $|\mathcal{E}|$ de la base d'apprentissage. L'accélération maximum envisageable d'après la loi d'Amdahl est également reportée pour un nombre variable de processeurs.

Si les résultats obtenus ci-dessus impliquent clairement que toute parallélisation de l'algorithme devra impérativement inclure dans la fraction du code répartie sur les processeurs le calcul de la couverture, le choix – encore vaste – du type d'architecture doit être précisé. Puisque les objectifs poursuivis posent comme contrainte d'assurer autant que possible la conservation des propriétés de fonctionnement de l'algorithme séquentiel ainsi que l'indépendance entre les architectures logicielles et physiques, seule une parallélisation se réduisant uniquement au calcul de la couverture peut être envisagée. En effet une parallélisation d'une plus grande partie de l'algorithme doit obligatoirement répartir au minimum soit des instructions (qui correspondront donc à l'application des opérateurs d'évolution), soit des données, c'est-à-dire les noyaux en cours de généralisation. Or ces

			Accélération maximale (Amdahl)				
Base	$\|\mathcal{E}\|$	$\%T_{couv}$	$p=2$	$p=4$	$p=8$	$p=16$	$p=32$
Iris	50	64.6%	1.47	1.93	2.30	2.53	2.67
Iris	100	77.6%	1.63	2.39	3.11	3.67	4.03
Iris	150	83.0%	1.71	2.65	3.65	4.50	5.10
Mushrooms	500	91.5%	1.84	3.18	5.01	7.03	8.80
Mushrooms	1000	94.9%	1.90	3.46	5.89	9.06	12.39
Mushrooms	1500	95.9%	1.92	3.56	6.21	9.90	14.09
Mushrooms	2000	96.7%	1.93	3.63	6.49	10.70	15.81
Mushrooms	4000	97.4%	1.94	3.71	6.76	11.51	17.71
Mushrooms	6000	97.7%	1.95	3.74	6.89	11.89	18.68
Mushrooms	8000	97.7%	1.95	3.74	6.89	11.89	18.68

TAB. 5.2: Proportion du temps d'exécution passée en tests de couverture lorsque l'on fait varier la taille de la base d'apprentissage.

deux cas de figure doivent être écartés, car le cas de la parallélisation des noyaux conduit à introduire une dépendance entre l'architecture matérielle (le nombre de processeurs) et l'architecture logicielle (le nombre de noyaux que l'on va répartir) ; alors que dans le cas où l'on envisage la répartition des opérateurs d'évolution, on aboutit à l'architecture de type *maître-esclave asynchrone avec mémoire partagée* décrite en 5.3.2, qui n'est pas adaptée à l'algorithme : en effet, les opérateurs d'évolution représentent une partie négligeable du temps d'exécution ; leur parallélisation n'apporte donc rien, d'autant plus que le temps nécessaire pour envoyer et recevoir les individus est bien supérieur au temps nécessaire à l'application de ces opérateurs.

Conclusion

L'étude préliminaire conduit à la sélection d'un type d'architecture dont le traitement réparti se limite à l'évaluation des règles générées. Cette architecture générique, schématisée figure 5.6, doit permettre d'assurer un degré de parallélisme important, comme l'ont montré les mesures effectuées sur l'algorithme séquentiel.

Parmi les décompositions possibles de l'architecture générique, la première idée venant à l'esprit consiste à considérer l'exploitation du parallélisme au niveau des données, ce qui correspond au schéma bien connu de l'AE de type maître-esclave exposé en 5.3.2. Cependant, cette approche n'est pas la seule : il est également possible d'envisager l'exploitation du parallélisme au niveau des instructions.

Les paragraphes suivants détaillent ces deux types d'architectures autant du point de vue théorique que de l'implantation pratique, cette dernière ayant parfois nécessité une adaptation aux contraintes du domaine. De plus, un modèle simple destiné à évaluer le comportement et les contraintes de chaque architecture est également présenté.

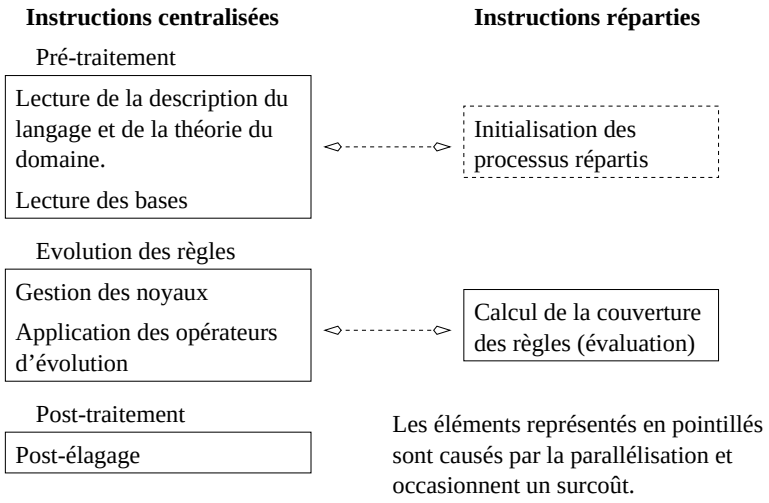


FIG. 5.6: Architecture générique proposée.

5.4.3.1 Parallélisme de données

L'architecture implantée

Bien que l'architecture maître-esclave synchrone soit la plus répandue et la plus facile à mettre en œuvre, la conception de $\mathcal{SIAO}1$ plaide en faveur d'une architecture asynchrone. En effet :

- l'algorithme étant incrémental, il est possible d'éviter le temps d'attente des esclaves pendant la gestion de la population ;
- de plus, l'asynchronisme permet d'adapter la charge de travail aux esclaves lorsque les temps d'évaluation varient d'un individu à l'autre.

On peut remarquer que l'équation provenant du modèle proposé par E. Cantù-Paz :

$$t_{gen} = (p+1).(a.n.l/p + c) + \frac{n.\alpha}{p} \qquad \textbf{(M-E Sync 3)}$$

ne permet pas de modéliser ces deux types de déperditions, puisque :

- le temps de gestion de la population par le processus maître est supposé négligeable (et n'est par conséquent pas pris en compte dans le modèle) ;
- le temps d'évaluation α d'un individu étant supposé constant, il n'y a aucune variance du temps d'évaluation $t_{eval} = \frac{n.\alpha}{p}$ de la fraction de la population envoyée à chaque esclave.

Par contre, cette équation fait ressortir le principal goulot d'étranglement de ce type d'architecture. En effet, le terme $\frac{n.\alpha}{p}$ correspond à la charge de travail incompressible, alors que le terme restant $(p+1).(a.n.l/p + c)$ est la partie du modèle correspondant

au surcoût de la parallélisation. Comme d'une part les expérimentations menées sur ce modèle ont montré qu'il était fiable, et que d'autre part son unique terme réductible décrit les temps de communication, une technique destinée à augmenter encore l'efficacité des esclaves a été proposée. Elle consiste à envoyer plusieurs individus pour évaluation lors de l'initialisation du processus (comme dans le cas de l'architecture maître- esclave synchrone, on utilisera une mémoire tampon, ici structurée en file d'attente). L'existence de cette file d'attente permet à l'esclave de travailler, même pendant les communications, du moment que le maître est capable de l'alimenter (c'est-à-dire qu'il ne perd pas pied, ce qui est l'hypothèse de base). Comme le montre la figure 5.7, cette amélioration doit permettre un meilleur taux d'utilisation des esclaves par rapport à ce que l'on peut attendre d'une architecture asynchrone classique (cf. figure 5.4).

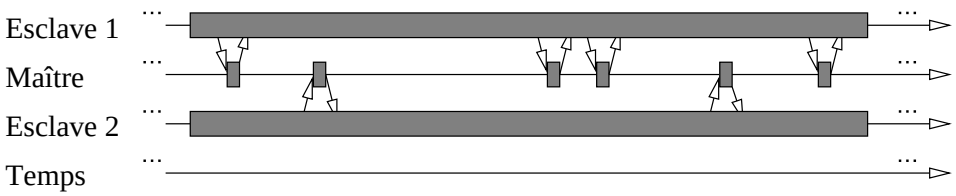


FIG. 5.7: Architecture maître-esclave asynchrone avec file d'attente.

Le temps est ici à l'échelle des durées de traitement
et de transmission des individus.

Modélisation

On suppose dans un premier temps que l'on peut négliger les contraintes liées au débit du réseau ainsi qu'à la charge du maître. Soit :

- p le nombre d'esclaves ;
- i le nombre d'individus traités ;
- t_c le temps de communication (en secondes) nécessaire pour transmettre un individu ;
- t_e le temps nécessaire (en secondes) pour évaluer (séquentiellement) un individu ;

La durée d'exécution de l'algorithme séquentiel de référence est :

$$T_{seq} = i.t_e \qquad \textbf{(T seq)}$$

Puisque les i individus sont répartis sur les p esclaves d'une part, et qu'une évaluation nécessite t_e secondes de la part d'un esclave d'autre part, la durée d'exécution de l'algorithme maître-esclave est donc[5] :

[5]Dans le cas de l'architecture maître-esclave asynchrone avec file d'attente, les seuls temps de communication non couverts par les durées d'évaluation des individus sont relatifs à l'envoi du premier et du dernier message.

$$T_p^{m-esc} = \frac{i.t_e}{p} + 2.t_c \qquad \textbf{(T m-esc)}$$

Pour que ces performances soient atteintes, il faut cependant que le temps t_m de gestion d'un individu par le maître et que le temps t_c de communication d'un individu (dépendant du débit du réseau) satisfassent les contraintes suivantes :

- tout d'abord il faut au maître $t_m + 2.t_c$ secondes pour servir un esclave ; or le maître doit servir p esclaves pendant t_e secondes, d'où $t_m + 2.t_c \leq \frac{t_e}{p}$, soit :
$$t_m \leq \frac{t_e}{p} - 2.t_c \qquad \textbf{(C1 m-esc)}$$
- enfin, il faut que le réseau puisse transmettre les $2.p$ messages pendant la même durée de base t_e, ce qui impose :
$$t_c \leq \frac{t_e}{2.p} \qquad \textbf{(C2 m-esc)}$$

5.4.3.2 Parallélisme de flux

L'architecture implantée

La seconde décomposition possible à partir de l'architecture générique détaillée figure 5.6 consiste à exploiter le parallélisme au niveau des instructions, ce qui conduit à répartir l'évaluation des individus sur les esclaves, ces derniers travaillant à la chaîne sur un unique flux de données. On obtient alors un fonctionnement en pipeline :

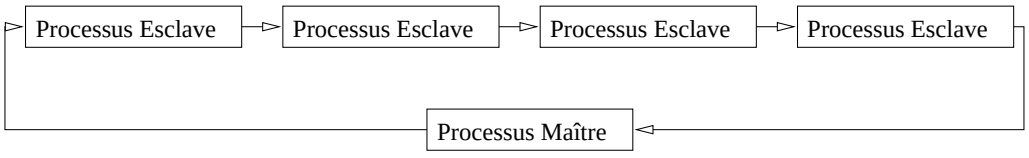


FIG. 5.8: L'architecture pipeline.

Le fait que ce type de décomposition se focalisant sur la fonction d'évaluation n'ait jamais été proposé pour les AE peut s'expliquer d'une part parce qu'il existe déjà des décompositions opérant à un niveau supérieur (que l'on pouvait donc supposer plus efficaces), et d'autre part parce que l'on considère habituellement l'évaluation d'un individu comme atomique au niveau de ces algorithmes, puisqu'il s'agit très souvent d'une opération rapide et difficile à répartir efficacement.

Cependant, cette décomposition est tout à fait adaptée au contexte de l'ECD, et plus particulièrement au changement d'échelle en espace. En effet, dans le cas de la fouille de grandes bases de données, la taille mémoire requise par l'algorithme devient proportionnelle au nombre d'exemples. Par conséquent, cette architecture permet de diviser l'espace mémoire nécessaire pour chaque processus esclave par le nombre de processeurs. On obtient alors, pour p suffisamment grand (quelques milliers d'exemples, cf tableau 5.4) :

$$M_p(n) \approx \frac{M_1(n)}{p}$$

Modélisation

En conservant les notations utilisées pour l'étude du comportement de l'architecture maître-esclave asynchrone avec file d'attente, on obtient dans le cas du pipeline :

$$T_p^{pipeline} = (p+1).t_c + t_e + \frac{(i-1).t_e}{p} \qquad \textbf{(T pipeline)}$$

En effet, la mise en marche du pipeline, qui correspond au temps que met le premier individu pour le traverser dure $(p+1).t_c + t_e$ secondes, puis tous les autres individus sortent un par un toutes les $\frac{t_e}{p}$ secondes.

Les contraintes sur t_m et t_c sont les suivantes :

- pour que le pipeline tourne à plein régime, le maître doit envoyer un individu toutes les $\frac{t_e}{p}$ secondes, soit :

$$t_m \leq \frac{t_e}{p} \qquad \textbf{(C1 pipeline)}$$

- de plus, le réseau doit pouvoir faire transiter (p+1) messages lors de chaque évaluation (de durée $\frac{t_e}{p}$ lorsque le pipeline tourne à plein régime), ce qui impose :

$$t_c \leq \frac{t_e}{p.(p+1)} \qquad \textbf{(C2 pipeline)}$$

Comparaison avec l'architecture maître-esclave

En contrepartie du gain en espace mémoire nécessaire réalisé qui peut être un critère décisif dans le choix d'une architecture lors de la fouille de bases de données importantes, le pipeline a quelques inconvénients qui le rendent moins performant en temps que l'architecture maître-esclave. En premier lieu le coût des communications est plus important dans le cas du pipeline puisque le nombre de messages nécessaires à l'évaluation d'un individu est égal au nombre total de processus (maître et esclaves), alors qu'il est constant dans le cas de l'architecture maître-esclave (deux messages suffisent). Ce coût plus important des communications peut être observé en comparant les contraintes portant sur t_c dans les deux modèles. D'autre part, contrairement à l'architecture maître-esclave asynchrone qui s'adapte aux variations de charge en environnement dynamique, le pipeline dépend pour son efficacité de chaque élément qui le compose. Par exemple, dans le cas où un processeur du système est ralenti (soit $c > 1$ le facteur de charge ralentissant ce processeur), l'architecture maître-esclave asynchrone avec file d'attente ne verra qu'un de ses processeurs ralenti, soit :

$$T_p^{m-esc\ ralenti} = (1 + \frac{c-1}{p}).\frac{i.t_e}{p} + 2.t_c \qquad \textbf{(m-esc ralenti)}$$

Dans le cas de l'architecture pipeline, ce même processeur ralenti diminuera le débit du flux d'individus, ce qui aura une conséquence pour tous les processeurs en introduisant des temps d'attente sur l'ensemble de la chaîne :

$$T_p^{pipeline\ ralenti} = (p+1).t_c + \frac{(p-1+c).t_e}{p} + c.\frac{(i-1).t_e}{p} \qquad \textbf{(pipeline ralenti)}$$

5.4.4 Du dépassement de la loi d'Amdahl

Pour conclure cette partie descriptive des architectures proposées, il convient de faire un retour sur la Loi d'Amdahl à la lumière de la problématique et de l'architecture générique proposées. En effet, le paragraphe 5.2.2 soulignait la très grande difficulté à atteindre une efficacité correcte dans le cas du parallélisme massif, puisque le taux d'instructions non parallélisables devait alors devenir infinitésimal (comme le montrait le tableau 5.1). Or l'argument à l'origine de cette assertion, qui est une conséquence mathématique directe de l'inéquation **(Amd 3)**, peut sembler contradictoire avec les résultats expérimentaux présentés dans les paragraphes suivants. Bien entendu, il n'en est rien.

La démarche présidant à la preuve de la loi d'Amdahl consiste à étudier les performances en temps d'un programme parallèle appliqué à une instance d'un problème de taille n fixée, que l'on borne (entre autres) en fonction du nombre de processeurs. Les inéquations obtenues contiennent donc l'hypothèse implicite que la taille n du problème à résoudre et le nombre p de processeurs sont indépendants, ce qui n'est pas le cas de la problématique de ce travail, le changement d'échelle consistant à augmenter la taille des problèmes traités de pair avec le nombre de processeurs.

En particulier on a pris soin d'inclure autant que possible les instructions dépendant de la taille du problème à traiter dans la partie de l'algorithme générique devant être répartie sur les processeurs. En conséquence, et contrairement à ce que suppose la loi d'Amdahl, le temps $T_{Par}(n)$ pris pour exécuter la partie de l'algorithme qui peut être parallélisée n'est plus une constante, puisque comme le montrent les mesures expérimentales cette valeur augmente avec la taille du problème. Ainsi, dans le cadre du changement d'échelle, les deux architectures parallèles proposées pour $\mathcal{SIAO}1$ sont de bonnes candidates au parallélisme massif (si elles sont appliquées à des bases de données importantes).

5.5 Résultats expérimentaux

5.5.1 Protocole de validation

Conditions d'expérimentation

Afin de quantifier aussi précisément que possible l'apport des deux architectures proposées, un environnement permettant de s'approcher autant que possible des conditions de mesure idéales a été recherché. Les expériences détaillées ci-dessous ont toutes été réalisées sur une machine IBM SP2 disposant de huit éléments identiques, construits autour de processeurs RS-6000 et disposant chacun de 256 Mo de mémoire. Bien que cette machine parallèle

possède un canal de communication à haut débit entre les éléments le constituant (le *switch*), le canal Ethernet classique (assurant un débit de 10Mo/s) a été préféré afin de se placer dans des conditions identiques à celles d'un réseau local standard, ce qui rend possibles des comparaisons avec d'autres travaux. Enfin, un système de supervision (appelé *loadleveller*) gère les travaux soumis à la machine et permet la réservation de processeurs dédiés à une application, ce qui garantit alors une puissance de calcul identique et constante des éléments utilisés.

Mesures effectuées

Il a été possible de vérifier expérimentalement que la qualité des solutions ainsi que la durée (en nombre de générations) des versions parallèle et séquentielle de l'algorithme étaient identiques. Cependant, à la mesure brute

$$T_p^{brute}(n) = \textit{Durée d'exécution du programme tournant sur p processeurs},$$

on a préféré une autre mesure :

$$T_p^{ratio}(n) = \frac{\textit{Durée d'exécution du programme tournant sur p processeurs}}{\textit{Nombre d'individus evalués}}.$$

En effet, l'algorithme n'étant pas déterministe, la variance du nombre de générations d'un test à l'autre provoque dans certains cas une variance non négligeable des durées d'exécution. De plus, le nombre de valeurs des paramètres d'une part et la durée de certains apprentissages d'autre part ne permettent pas de faire un nombre de tests suffisant pour estimer dans ces cas les valeurs moyennes avec une précision suffisante. Par contre, l'ajustement provoqué par l'évaluation de $T_p^{ratio}(n)$ permet de fournir avec certitude des résultats très précis équivalents car strictement proportionnels à $T_p^{brute}(n)$.

Les mesures étudiées sont donc :

- $T_p^{ratio}(n) = \frac{T_p^{brute}(n)}{\textit{Nombre d'individus evalues}}$;
- $M_p(n) = \textit{Taille mémoire (en Mo) des esclaves.}$

Il convient enfin de noter que –sauf mention du contraire– il a été affecté un processus esclave par processeur. La charge du processus maître étant en effet très peu importante, ce dernier partageait un processeur avec un processus esclave.

5.5.2 Premiers résultats

Une première série d'expériences[6] destinées à estimer grossièrement le gain que l'on peut obtenir grâce aux architectures proposées a été conduite sur deux bases dont les caractéristiques détaillées figurent en annexe B. Ce choix résulte ici simplement du fait que *Mushrooms* est une base de taille moyenne (0,37 Mo, plusieurs milliers d'exemples),

[6]Ces expérimentations préliminaires ont été réalisées avant que le mécanisme de mémoire tampon auquel il est fait référence au paragraphe 5.4.3.1 ne soit envisagé et implanté.

alors que *Census-income* est une base de grande taille (155 Mo, plusieurs centaines de milliers d'exemples), ce qui permet d'estimer les performances des deux architectures sur un problème habituel ainsi que du point de vue du changement d'échelle en espace.

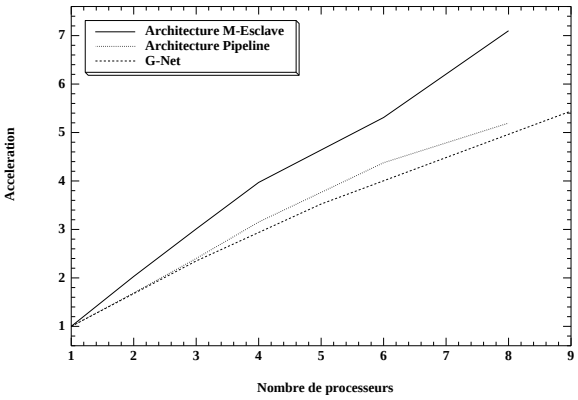


FIG. 5.9: Accélération (*Mushrooms*)

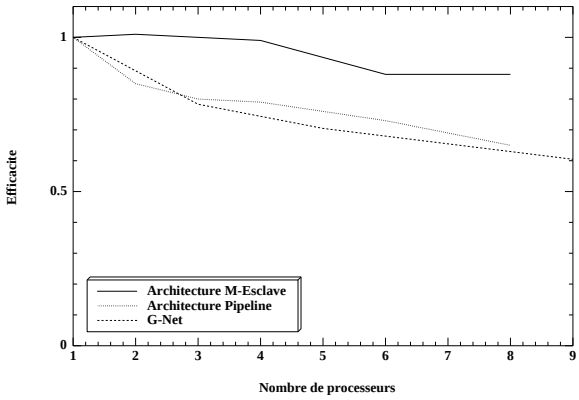


FIG. 5.10: Efficacité (*Mushrooms*)

Les figures 5.9 et 5.10 permettent de comparer les performances obtenues pour les deux architectures de $\mathcal{SIAO}1$ sur la base *Mushrooms* (4000 exemples d'apprentissage, 4128 exemples de test) avec des résultats concernant G-Net publiés dans [Angelano et al.97]. Le tableau suivant précise les conditions expérimentales qui ne sont pas exactement les mêmes :

Système	$\mathcal{SIAO}1$	**G-Net**
Machine élémentaire	RS-6000	Sparc20
Débit du Réseau	10 Mb/s	10 Mb/s
Base	Mushrooms	Thousand trains
Exemples d'apprentissage	4000 (attribut-valeur)	1000 (relationnels)

TAB. 5.3: Conditions d'expérimentation.

La seconde partie de l'expérimentation, dont les résultats sont montrés figures 5.11 et 5.12, concerne l'évaluation de l'architecture pipeline sur la base *Census-income*. Cette base est particulièrement intéressante car il s'agit d'un problème réel de grande taille (voir annexe B, section B.1).

Les résultats montrent que l'on obtient dans ce cas des performances quasi-optimales, alors que le pipeline est en théorie la plus lente des deux architectures étudiées.

Cette pré-étude ayant été réalisée alors que les nœuds de l'IBM SP2 ne disposaient que de 128 Mo de mémoire RAM, ces derniers se sont écroulés lors du test des versions

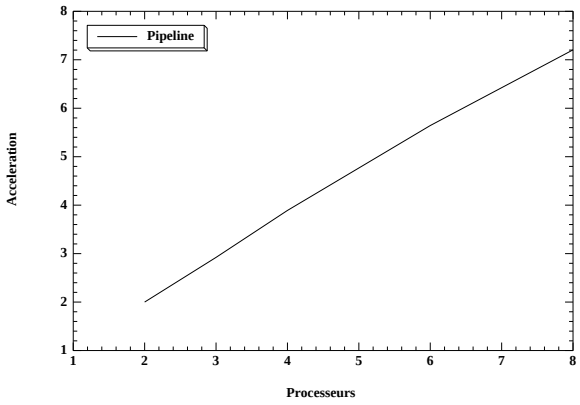


FIG. 5.11: Accélération

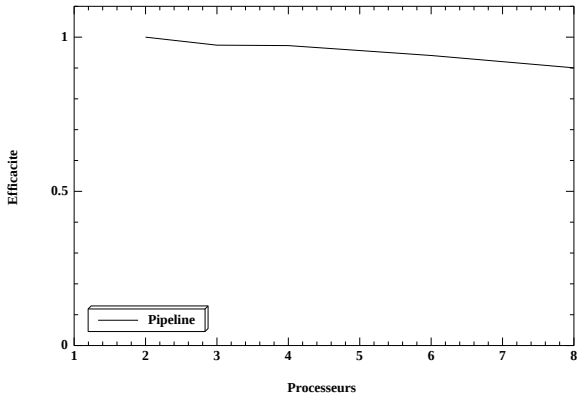


FIG. 5.12: Efficacité

séquentielles et maître-esclave du système. Bien qu'il ne soit pas possible de connaître avec certitude les causes de ce problème, il semble néanmoins qu'elles soient relatives à la gestion des fichiers de recouvrement (*swap*) par le système d'exploitation. L'architecture pipeline utilisant deux esclaves a donc servi de point de référence avec une efficacité fixée à l'unité et une accélération fixée à 2.

Enfin, une étude de la taille mémoire requise par les processus esclaves pour l'architecture pipeline, dont les résultats figurent dans le tableau 5.4, a été réalisée sur des PCs connectés en réseau. La décroissance de $M_p(n)$ est bien en $\frac{1}{p}$ à partir de $p = 2$ esclaves. Le fait que les mesures concernant le processus séquentiel dénotent une taille mémoire plus importante que prévu est sans doute en partie dû aux différentes structures prises en charge par le processus maître et non réparties sur les esclaves (populations, etc.).

Nombre d'esclaves Base	1[a]	2	3	4	5	6	7	8
Mushrooms	4048	1672	1400	1260	1164	1116	1080	1052
Census-income	143360	52008	34916	26308	21160	17732	15236	13396

[a]Taille de l'unique processus de la version séquentielle.

TAB. 5.4: Espace mémoire requis par les processus du pipeline (en Ko).

5.5.3 Étude sur les bases *Mushrooms* et *Iris*

Problématique

Les résultats précédents ont permis la validation des architectures en prouvant leurs très bonnes performances sur différents types de problèmes. Cependant, cette première validation, succincte et empirique, soulève également la question de savoir s'il est possible

d'évaluer plus précisément les capacités de changement d'échelle du système. Comme l'a montré le paragraphe 5.4.2, la plupart des grandeurs caractérisant la capacité d'un algorithme à changer d'échelle nécessitent un très grand nombre de mesures, ce qui est non seulement coûteux en terme d'allocation de ressources informatiques, mais également difficile à justifier en pratique.

Une solution à ce problème consiste à trouver un modèle du comportement des architectures, idéalement aussi simple à mettre en œuvre et précis que possible. Le modèle de [CP97a] déjà décrit au paragraphe 5.3.4 étant spécifique aux architectures maître-esclaves synchrones sans files d'attentes ne peut être utilisé, et ce d'autant plus que c'est l'étude des termes réductibles de l'équation **(M-E Sync 3)** de ce même modèle qui a guidé la conception des architectures implantées. D'autre part, les équations **(T m-esc)** et **(T pipeline)** des paragraphes 5.4.3.1 et 5.4.3.2 sont beaucoup trop optimistes puisque la charge étant répartie de manière optimale sur les processeurs, et les coûts de communications étant presque totalement éliminés grâce au mécanisme des files d'attente, on a :

$$lim_{i\to\infty} \frac{T\ m-esc}{i} = lim_{i\to\infty} \frac{T\ pipeline}{i} = \frac{t_e}{p}$$

ce qui, du point de vue d'un modèle et au vu des résultats obtenus ne semble pas réaliste même lorsque les contraintes respectives **(C1 m-esc)** et **(C2 m-esc)** ou **(C1 pipeline)** et **(C2 pipeline)** – qui portent entre autres sur p – sont vérifiées.

Construction du modèle

Une étude plus complète est donc nécessaire afin de caractériser les nouveaux facteurs de ralentissement. Ces derniers devant être d'autant plus visibles que les temps de traitement des esclaves sont courts, la base *Mushrooms* a été utilisée. Les résultats, détaillés en annexe C sont résumés par les graphiques ci-après. Les acronymes RCD et RCS font respectivement référence aux architectures à Répartition de Charge Dynamique (il s'agit alors du maître-esclave asynchrone) et à Répartition de Charge Statique (c'est le pipeline). On s'est non seulement attaché à faire varier le nombre de processeurs, mais également à étudier l'influence de la taille du tampon de mémorisation des messages (Norm. dénote une communication sans utilisation d'aucune mémoire tampon, et L(i) l'utilisation d'une liste de taille i). Chaque point représente la valeur moyenne obtenue pour cinq exécutions, et les barres l'entourant ont leurs extrémités à une distance de un écart-type de la valeur moyenne[7].

Comme attendu, on remarque que d'une part l'architecture RCD est légèrement plus performante que l'architecture RCS, et d'autre part que la décroissance de l'accélération, bien que légère, est notable et ne peut être expliquée par les modèles mentionnés auparavant. Par contre, l'incidence de la variation de la taille des tampons de mémorisation des

[7]Sur les graphiques détaillant les résultats obtenus sur la base *Mushrooms*, ces dernières sont trop petites pour pouvoir être vues avec discernement

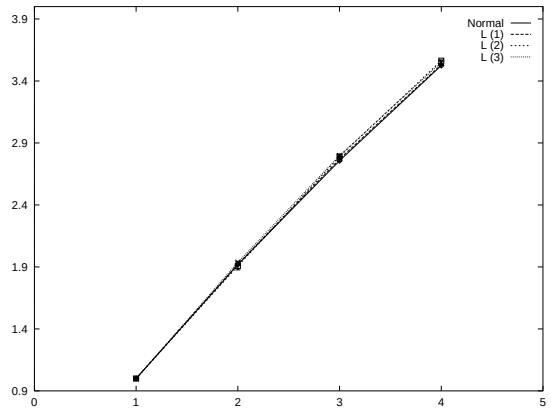


FIG. 5.13: Accélération RCD

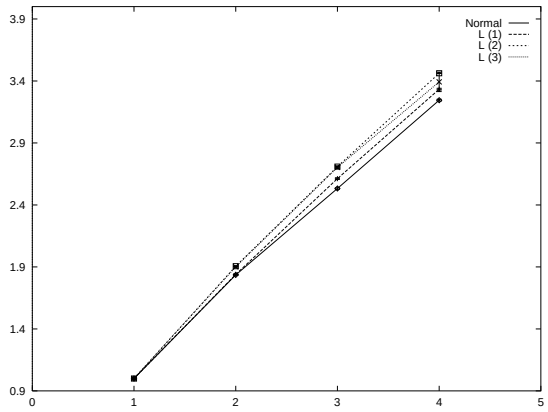


FIG. 5.14: Accélération RCS

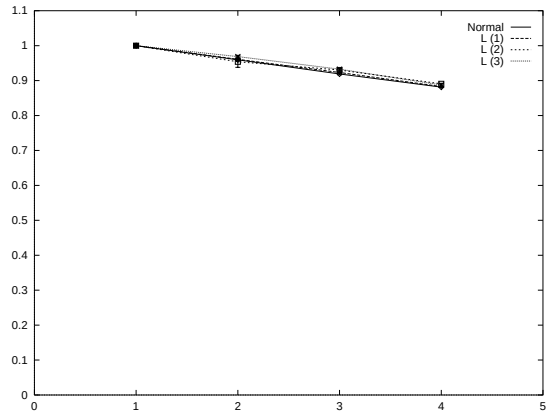


FIG. 5.15: Efficacité RCD

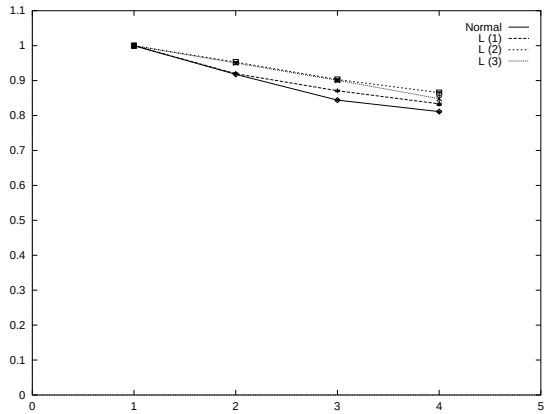


FIG. 5.16: Efficacité RCS

messages est nulle sur les performances de l'architecture RCD et très faible dans le cas de l'architecture RCS. En fait, une observation plus détaillée de la charge des processeurs montre que la partie logicielle du système de gestion des messages (en l'occurrence le processus-demon `pvmd`) nécessite une puissance de calcul de moins en moins négligeable lorsque le nombre d'esclaves s'accroît.

Contrairement à la loi d'Amdahl, qui pose $T_1(n) = T_{Seq}(n) + T_{Par}(n)$ puis en déduit que de manière optimale – en négligeant le surcoût dû aux communications – $T_p^*(n) = T_{Seq}(n) + \frac{T_{Par}(n)}{p}$, les observations expérimentales conduisent ici à négliger $T_{Seq}(n)$ mais à ajouter un terme correcteur lié au surcoût logiciel causé par la gestion des communications $T_{Com}(n,p)$. Contrairement à $T_{Seq}(n)$, $T_{Com}(n,p)$ dépend bien évidemment du nombre de processeurs. On pose donc :

$$T_p(n) = \frac{1}{p}(T_{Com}(n,p) + T_1(n)) \quad \textbf{(Eq-M 1)}$$

De plus, on déduit des conditions initiales en $p = 1$ que $T_{Com}(n,1) = 0$. Ainsi, on a :

$$T_{Com}(n,p) = p.T_p(n) - T_1(n) \quad \textbf{(Eq-M 2)}$$

ce qui permet d'obtenir immédiatement à partir des mesures le comportement de T_{Com}, fourni également en annexe C et résumé par les graphes suivants :

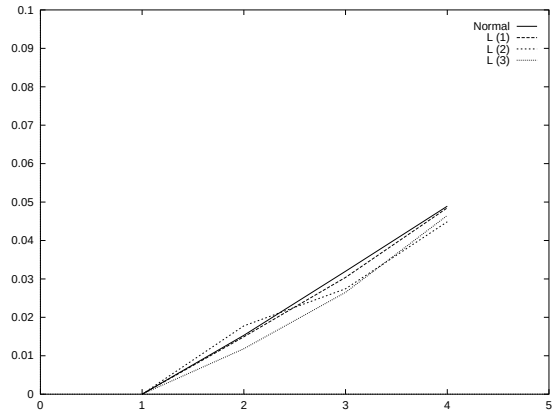


FIG. 5.17: T_{Com} RCD

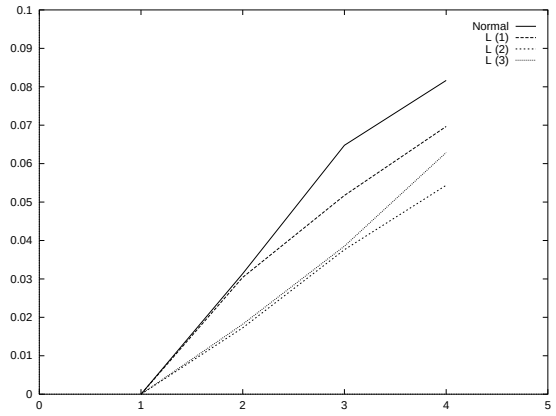


FIG. 5.18: T_{Com} RCS

Bien que T_{Com} ne soit connu qu'avec une moindre précision puisqu'il est obtenu par différence de deux mesures, il semble possible au vu des résultats de postuler une relation linéaire entre ce paramètre et le nombre de processeurs : $T_{Com}(n,p) = T_{Com}(p) = a.p + b$. En effet :

- T_{Com} ne peut dépendre que du nombre de processeurs, à l'exclusion de la complexité du problème, puisque les seules données échangées sur le réseau au cours de l'évolution sont les messages représentant les individus. Or la taille de ces derniers reste petite par rapport à la taille de la base contenant les exemples. Ceci se remarque en particulier au sujet du protocole d'expérimentation proposé ici et qui consiste, lorsque l'on veut faire varier n, à procéder à des échantillonnages de la base d'apprentissage : on conserve ainsi exactement la taille des individus qui devient donc une constante.
- T_{Com} modélisant la surcharge logicielle (et donc locale à chaque processeur) due aux communications, il semble cohérent de postuler que, si l'on prend en considération le maître – dont la charge en terme de gestion de messages est égale (architecture RCS) ou supérieure (architecture RCD) à celle des esclaves – ce terme dépend linéairement du nombre d'esclaves (et donc de processeurs).

Validation du modèle

L'annexe C contient les coefficients de corrélation r, l'équation des droites de régression, ainsi que la probabilité α que la valeur de r calculée soit dépassée en valeur absolue pour deux variables aléatoires indépendantes. Ces tables montrent que les deux variables sont certainement liées (on a toujours $\alpha \leq 10^{-2}$). D'autre part, l'utilisation conjointe du modèle décrit ci-dessus et des estimations de T_{Com} à l'aide des droites de régression fournissent des estimations très fidèles (l'erreur maximale est de 0.6% pour l'architecture RCD et de 1.5% pour l'architecture RCS) :

	RCD Norm.			RCD L(1)			RCD L(2)			RCD L(3)		
p	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.
1	1,000	1,001	0,1%	1,000	1,001	0,1%	1,000	0,997	0,3%	1,000	1,005	0,5%
2	0,959	0,958	0,1%	0,960	0,959	0,1%	0,953	0,959	0,6%	0,968	0,964	0,4%
3	0,919	0,918	0,1%	0,923	0,920	0,3%	0,930	0,924	0,6%	0,932	0,926	0,6%
4	0,881	0,881	0,0%	0,882	0,884	0,2%	0,890	0,892	0,2%	0,887	0,892	0,5%

	RCS Norm.			RCS L(1)			RCS L(2)			RCS L(3)		
p	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.
1	1,000	0,992	0,8%	1,000	0,990	1,0%	1,000	1,000	0,0%	1,000	1,004	0,4%
2	0,917	0,919	0,2%	0,919	0,929	1,0%	0,952	0,950	0,2%	0,950	0,947	0,3%
3	0,844	0,857	1,5%	0,870	0,875	0,5%	0,903	0,905	0,2%	0,900	0,896	0,4%
4	0,811	0,802	1,1%	0,833	0,827	0,7%	0,865	0,864	0,1%	0,848	0,851	0,3%

TAB. 5.5: Performance du modèle appliqué au calcul de l'efficacité (*Mushrooms*).

Puisque le comportement de l'algorithme parallèle est correctement capturé par le modèle, la dernière étape de validation concerne l'étude de sa capacité de généralisation, c'est-à-dire de prédiction dans des cas non pris en compte lors de son élaboration. Pour ce faire, deux types de variations ont été prises en compte :

1. *Variation de la taille de la base d'apprentissage.*

   Ce paramètre, particulièrement important pour toute étude de changement d'échelle, peut être immédiatement pris en compte en appliquant un coefficient représentant l'accroissement (ou la diminution) de charge sur T_1, le comportement de T_{Com} restant inchangé. Les résultats ci-dessous montrent que le modèle obtenu précédemment pour un apprentissage sur 4000 exemples de la base *Mushrooms* s'adapte parfaitement à des apprentissages sur 2000 ou 6000 exemples de cette même base.

Archi.	Com.	$\lvert\mathcal{E}\rvert$	p	T_p^{ratio} prédit	T_p^{ratio} observé	Erreur
RCD	Norm.	2000	3	0,0712	0,0679	4,8%
RCD	Norm.	6000	2	0,280	0,283	1,0%

2. *Variation du nombre de processeurs.*

   Le second paramètre crucial pour toute étude de changement d'échelle est bien entendu le nombre p de processeurs. Le tableau ci-après compare les résultats expérimentaux observés sur 5 et 6 processeurs avec les résultats prédits à l'aide du modèle élaboré à partir des données pour 1 à 4 processeurs. Dans ce cas également, la précision obtenue est tout-à-fait satisfaisante.

Archi.	Com.	$\lvert\mathcal{E}\rvert$	p	T_p^{ratio} prédit	T_p^{ratio} observé	Erreur
RCD	Norm.	4000	5	0,0856	0,0864	0,9 %
RCD	Norm.	4000	6	0,0740	0.0752	1,6 %

Cas de la base *Iris*

Afin d'être complète, l'étude du comportement des architectures parallèles proposées pour $\mathcal{SIAO}1$ et du modèle de prédiction de leurs performances a été également réalisée dans un cas limite. Pour ce faire, la base *Iris*, ne contenant que 150 exemples, a été utilisée. Bien que n'ayant aucun intérêt pratique (puisqu'un apprentissage sur cette base dure moins d'une minute sur un IBM PC classique, ce qui retire tout intérêt à une parallélisation), cette étude de cas permet de se placer dans des conditions extrêmes : le temps d'évaluation d'un individu est si peu important que d'une part la charge du maître (gestion de la population, ...) n'est plus du tout négligeable, et que d'autre part le débit du réseau n'est plus suffisant pour assurer des communications fluides lorsqu'on augmente le nombre de processeurs.

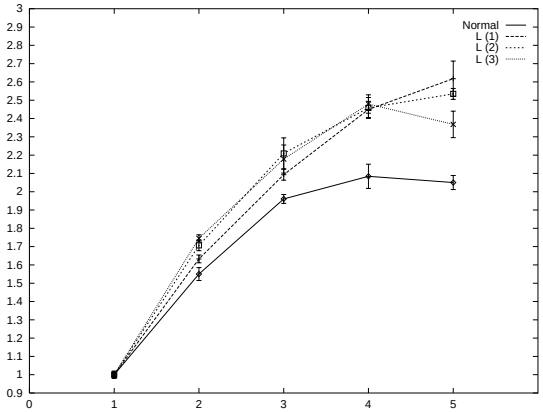


FIG. 5.19: Accélération RCD

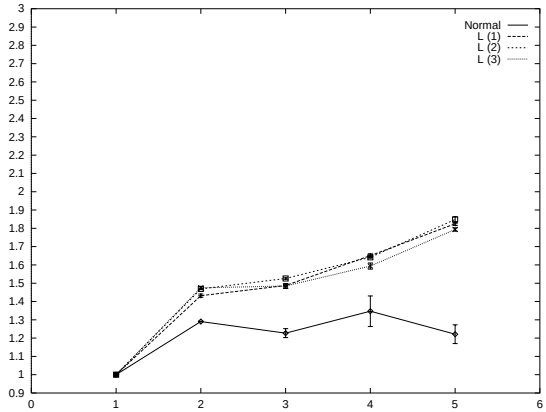


FIG. 5.20: Accélération RCS

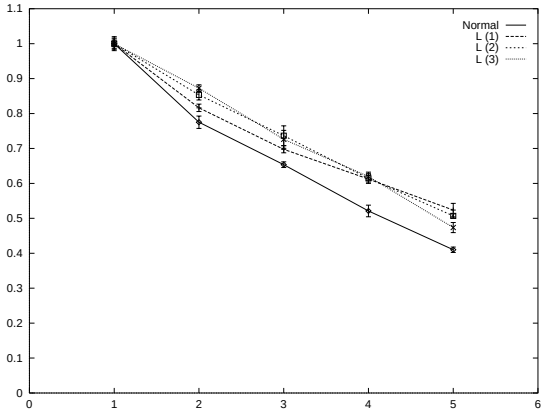


FIG. 5.21: Efficacité RCD

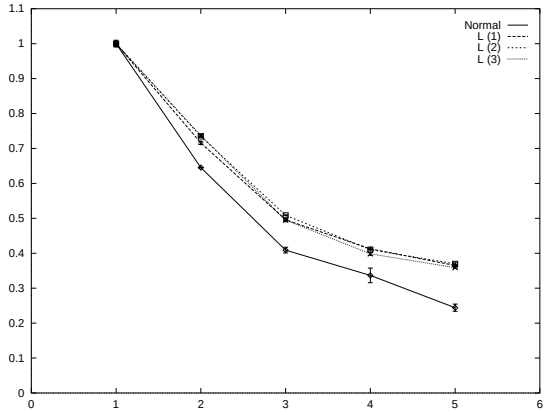


FIG. 5.22: Efficacité RCS

Les figures 5.19, 5.20, 5.21 et 5.22 illustrent bien le fait que le maître perd peu à peu pied, l'accélération n'étant plus du tout linéaire et l'efficacité très mauvaise. Contrairement à ce qui se passait dans l'exemple précédent, on peut remarquer deux faits nouveaux :

- Les performances de l'architecture RCS sont considérablement plus dégradées que celles de l'architecture RCD. En effet, l'accélération semble se stabiliser à 2,5 dans le cas de l'architecture RCD au delà de 4 processeurs, alors qu'elle reste inférieure à 1,9 dans le cas de l'architecture RCS sans connaître de progression notable au delà de 2 processeurs.

- L'ajout d'un tampon mémorisant un message accroît notablement les performances, cependant aucune amélioration n'est constatée si l'on continue d'augmenter la taille du tampon. Malgré la plus grande incertitude sur les résultats (observable aux écarts-types mesurés), ce constat est valable quelle que soit l'architecture utilisée.

Par contre, comme le montrent intuitivement les figures 5.23 et 5.24 (ceci est confirmé par les résultats détaillés de l'annexe C), on peut toujours postuler une dépendance entre T_{Com} et p, et il semble que cette dépendance puisse encore être modélisée (plus grossièrement cette fois) par une droite. Bien que le modèle obtenu en ajustant T_{Com} sur sa droite

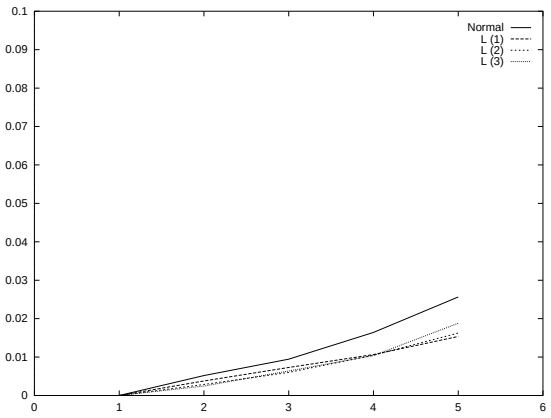


FIG. 5.23: T_{Com} RCD

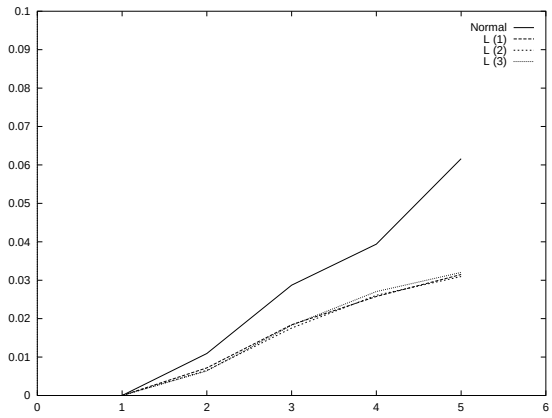


FIG. 5.24: T_{Com} RCS

	RCD Norm.			RCD L(1)			RCD L(2)			RCD L(3)		
p	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.
1	1.000	1.070	7.0%	1.000	1.005	0.5%	1.000	1.056	5.6%	1.000	1.097	9.7%
2	0.775	0.777	0.2%	0.816	0.821	0.6%	0.853	0.842	1.2%	0.872	0.846	2.9%
3	0.653	0.611	6.4%	0.697	0.693	5.7%	0.736	0.700	4.8%	0.726	0.689	5.0%
4	0.521	0.503	3.4%	0.612	0.600	1.9%	0.614	0.599	2.4%	0.619	0.581	6.1%
5	0.410	0.427	4.1%	0.523	0.529	1.1%	0.506	0.524	3.5%	0.473	0.502	6.1%

	RCS Norm.			RCS L(1)			RCS L(2)			RCS L(3)		
p	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.	réel	modèle	err.
1	1.000	1.125	12.5%	1.000	0.986	1.4%	1.000	1.003	0.3%	1.000	1.011	1.1%
2	0.645	0.605	6.2%	0.715	0.682	4.6%	0.734	0.691	5.8%	0.737	0.683	7.3%
3	0.409	0.413	0.9%	0.495	0.522	5.4%	0.508	0.527	3.7%	0.494	0.516	4.4%
4	0.336	0.314	6.5%	0.412	0.422	2.4%	0.410	0.426	3.9%	0.398	0.415	4.2%
5	0.244	0.253	3.6%	0.365	0.355	2.7%	0.368	0.358	2.7%	0.358	0.347	3.0%

TAB. 5.6: Performance du modèle appliqué au calcul de l'efficacité (*Iris*).

de régression soit moins précis qu'auparavant (cf. tableau 5.6), on peut observer que ce paramètre, ainsi que les hypothèses concernant sa variation, permettent encore de rendre compte de manière correcte de l'évolution de l'efficacité. En conclusion, l'étude du cas limite (qui a lieu lorsque le temps de gestion de la population par le maître n'est plus négligeable devant le temps d'évaluation des individus) montre que le ralentissement dû à la partie logicielle de la gestion des messages (modélisée par le paramètre T_{Com}) est le facteur limitant de plus grande importance. La modélisation de ce facteur s'avère être suffisante pour rendre compte et prévoir les performances du système.

Conclusion

Un modèle simple des performances des architectures maître-esclave et pipeline a été proposé. Il a été montré expérimentalement que le comportement du principal paramètre, T_{Com}, peut être décrit par une loi linéaire, que l'on peut déterminer par régression à partir d'un très petit nombre de tests, soit :

$$T_{Com} = \alpha.p + \beta \quad (\Delta)$$

Il a été également observé que cette relation permet de généraliser correctement le comportement de T_{Com} pour des valeurs de p voisines non prises en compte lors de la construction du modèle. D'autre part, l'impact de la variation de la complexité du problème (supposée ici proportionnelle à sa taille, c'est-à-dire au nombre d'exemples d'apprentissage) est également prédite correctement. En conclusion, la relation

$$T_p(m) = \tfrac{1}{p}.\left[\alpha.p + \beta + \tfrac{m}{n}.T_1(n)\right] \quad \textbf{(Eq-M 3)}$$

est un modèle prédictif fiable.

5.5.4 À propos du changement d'échelle de $\mathcal{SIAO}1$

Une application immédiate concerne l'estimation des capacités de changement d'échelle des architectures implantées pour le problème *Mushrooms* sur la machine IBM-SP2.

La figure 5.25 montre l'efficacité prédite par le modèle dans le cas de l'architecture maître-esclave utilisant une file d'attente de taille un. La complexité de la tâche varie de 2.000 à 8.000 exemples d'apprentissage et le nombre p de processeurs de 1 à 8. Il est intéressant de noter que si la conservation d'une efficacité presque parfaite (par exemple 90%) semble difficile (le nombre d'exemples à soumettre au système s'accroît beaucoup trop rapidement), il est néanmoins possible d'obtenir de bonnes performances (comme le montre, par exemple, la région $E \in [0,8..0,9]$) pour un facteur de changement d'échelle très raisonnable.

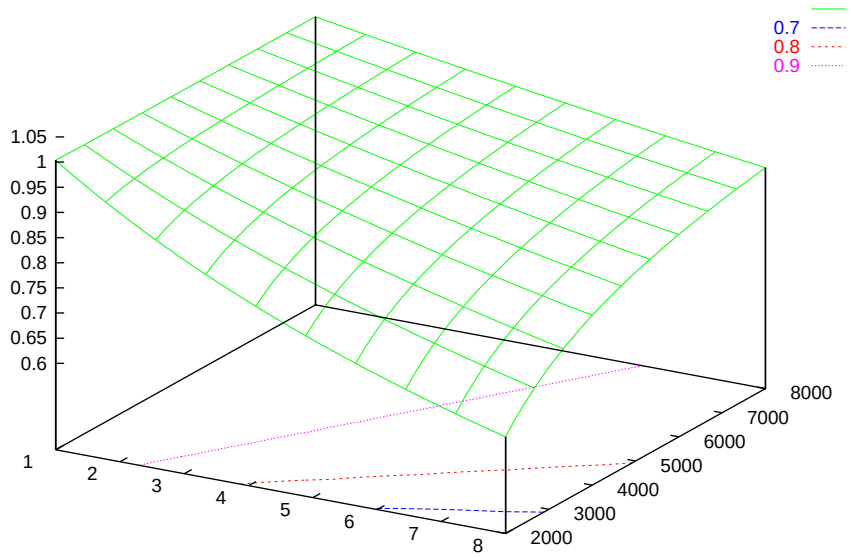


FIG. 5.25: Surface $E(n,p)$ calculée d'après le modèle pour le problème *Mushrooms*. Les courbes d'iso-efficacité associées figurent sur le plan (n,p).

5.6 Bilan

Ce chapitre a permis la validation de deux architectures parallèles implantées afin de permettre à $\mathcal{SIAO}1$ de changer d'échelle tout en conservant intégralement le fonctionnement de l'algorithme.

Dans le cas où l'accroissement de la complexité est dû à une forte composante relationnelle du problème, ou lorsque la base d'apprentissage est de taille moyenne (500 à 20 000 exemples), l'architecture de type maître-esclave doit être préférée. En effet, d'une part son asynchronisme lui permet de s'adapter à la variation des durées d'évaluation, et d'autre part il a été montré qu'elle est plus performante dans ce contexte d'utilisation.

Par contre, lorsqu'on se place dans une perspective de montée en charge afin de traiter de grandes bases d'apprentissage, dont la taille est éventuellement supérieure à la mémoire disponible sur une machine, on utilisera l'architecture basée sur un pipeline puisqu'elle répartit les données sur les processeurs. De plus, les temps d'évaluation devenant très importants, le problème du débit du réseau n'est plus un obstacle et les performances obtenues approchent la limite théorique prédite par la loi d'Amdahl.

Enfin, un modèle simple et précis des performances des architectures a été proposé, et sa mise en œuvre a permis d'évaluer plus précisément les capacités de changement d'échelle de $\mathcal{SIAO}1$ par le biais du comportement de l'iso-efficacité.

Chapitre 6

Conclusion

6.1 Bilan

Cette thèse a présenté la conception, l'implantation et la validation du système d'apprentissage automatique supervisé symbolique (AS) $\mathcal{SIAO}1$ fondé sur les algorithmes d'évolution. La première caractéristique de ce système est sa polyvalence, puisqu'il peut s'appliquer dans des contextes extrêmement variés sans pour autant nécessiter la spécification de nombreux biais :

- les données peuvent être de nature symbolique, numérique ou mixte ;
- le langage de description des concepts peut être de type attribut-valeur ou relationnel ;
- dans le cas de l'induction de concepts relationnels, la forme de la solution s'adapte au cours de l'évolution[1] ;
- la taille des bases d'apprentissage peut varier de quelques dizaines à plusieurs centaines de milliers d'instances.

Parmi les diverses problématiques abordées, deux directions de recherche sont centrales pour les domaines de l'AS et de l'extraction de connaissances à partir des données.

La première direction est issue du dilemme liant l'expressivité des langages de description des instances et des hypothèses au coût de l'induction. L'approche adoptée, qui met en œuvre des techniques de propagation de contraintes, montre qu'il est possible – y compris dans le cas des problèmes relationnels réels complexes – de relaxer certains biais de recherche ou de représentation classiques par l'utilisation de procédures optimisées adaptées au langage considéré.

La seconde direction concerne la problématique du changement d'échelle des méthodes

[1] En particulier, il n'est pas nécessaire de fournir la forme de l'hypothèse ou de spécifier de biais de recherche décrivant le nombre et le type des liens qui la constituent.

d'AS, résolue ici en parallélisant l'algorithme $\mathcal{SIAO}1$. Ce travail a permis de montrer qu'une première alternative, commune à de nombreux travaux du domaine, qui consiste à répartir les évaluations des hypothèses suivant un schéma maître-esclave, peut être améliorée et complétée, lorsque le volume des données est important, par une architecture répartissant les instances d'apprentissage et de test sur les différentes machines afin de diminuer les besoins en mémoire. Un modèle prédictif permettant d'évaluer le gain procuré par les architectures proposées a également été présenté et validé.

Un autre apport de cette thèse a trait à l'application des algorithmes évolutionnaires (en tant que procédures d'optimisation) aux problèmes d'AS. $\mathcal{SIAO}1$ prend en effet le parti d'intégrer nettement dans la stratégie de recherche les deux principales caractéristiques qui distinguent les tâches d'apprentissage des autres problèmes d'optimisation. En effet, d'une part la relation de généralité qui structure l'espace des hypothèses est un élément central de la stratégie de recherche puisqu'elle est ascendante, et d'autre part la base d'apprentissage est mieux exploitée puisque les opérateurs d'évolution sont, contrairement aux approches classiques, dirigés par les données.

6.2 Perspectives

Au stade actuel de son évolution, $\mathcal{SIAO}1$ pourrait bénéficier de deux types d'extensions.

1. Une architecture réalisant un pont avec un système de gestion de bases de données (SGBD). L'ajout de cette fonctionnalité est un travail complexe car il pose – entre autres – le problème de l'accès aux données. Ces dernières peuvent en effet être gérées soit par le SGBD ($\mathcal{SIAO}1$ évaluant chaque hypothèse au moyen d'une requête), soit par $\mathcal{SIAO}1$ (le pont se limite alors à un système de traduction).

2. Une interface gérant l'ensemble du cycle visualisation des données / apprentissage / étude et modification du modèle et permettant également un interaction directe avec le système en cours d'apprentissage.

Ces deux extensions feraient de $\mathcal{SIAO}1$ un système complet d'extraction des connaissances à partir des données.

Une autre orientation possible concerne la modification du mécanisme d'évolution simultanée de plusieurs noyaux afin d'augmenter les interactions entre les différentes populations. À ce titre, les stratégies mises en œuvre dans [Angelano et al.97] et [Torre99] pourraient être adaptées au contexte de $\mathcal{SIAO}1$ qui est caractérisé par un petit nombre de populations et une recherche ascendante. De telles stratégies permettraient également à l'utilisateur de ne plus avoir à spécifier le nombre de noyaux devant évoluer.

Annexe A

Notations - Définitions

Cette annexe contient une table récapitulant les diverses notations et abréviations utilisées au cours de cette thèse, ainsi que certaines définitions non rappelées dans le corps du document concernant :

1. Les ensembles ordonnés et les treillis ;

2. La construction et la manipulation des formules de la logique du premier ordre.

A.1 Notations

Ensembles :

$\in$	élément de l'ensemble.
$\subset$	sous-ensemble strict.
$\subseteq$	sous-ensemble.
$\cap$	intersection.
$\cup$	union.
$\setminus$	différence ensembliste.
$\emptyset$	ensemble vide.
$\lvert A \rvert$	cardinal de l'ensemble A.
$\overline{A}$	complément de l'ensemble A.
$\aleph$	l'ensemble des nombres entiers, ou, par abus, son approximation par l'ensemble des entiers représentables en machine.
$\Re$	l'ensemble des réels, ou, par abus, son approximation par l'ensemble des nombres flottants représentables en machine.

Logique :

$\wedge$	conjonction (et).
$\vee$	disjonction (ou).
$\neg$	négation (non).
$\forall$	quantificateur universel pour tout .
$\exists$	quantificateur existentiel il existe .
$\leftarrow$	implication (dans le cadre de la notation des clauses).
$\rightarrow$	implication (notation usuelle, de gauche à droite).
$\models$	implication logique (non décidable en logique d'ordre 1).
$\vdash$	déduction syntaxique (par des règles d'inférence, par la théorie du domaine).
$\leftrightarrow$	équivalence.
$\mathcal{C}$	ensemble fini de symboles de constantes.
$\mathcal{F}$	ensemble fini de symboles de fonctions.
$\mathcal{P}$	ensemble fini de symboles de prédicats.
$\mathcal{T}$	ensemble des termes du langage de la logique du premier ordre.
$\mathcal{V}$	ensemble infini de symboles de variables.

Algorithmique :

$\leftarrow$	opérateur d'affectation.
A_n^p	nombre d'arrangements (ens. ordonnés) de p objets pris parmi n.
	$Def : A_n^p = \frac{n!}{(n-p)!}$
C_n^p	nombre de combinaisons (ens. non ordonnés) de p objets pris parmi n.
	$Def : C_n^p = \frac{n!}{p!(n-p)!}$
$f = O(g)$	f est dominée asymptotiquement par g.
	$Def : f = O(g)\ ssi\ \exists c \in \Re^{+*}, \exists n_0 \in \aleph / \forall n > n_0, f(n) \leq c.g(n)$
$f = \Theta(g)$	f et g ont même ordre de grandeur asymptotique.
	$Def : f = \Theta(g)\ ssi\ f = O(g)\ et\ g = O(f)$

Apprentissage :

$\leq_g$	relation de généralité définie sur $\mathcal{L}_h$ (c'est un ordre partiel).
$\vdash_\theta$	relation de θ-subsomption (voir chapitre 3)
$\mathcal{B}$	base d'apprentissage de cardinal $\|\mathcal{B}\|$.
$\mathcal{B}^+$	l'ensemble des exemples (du concept-cible) de la base d'apprentissage.
$\mathcal{B}^-$	l'ensemble des contre-exemples de la base d'apprentissage.
Ce	un contre-exemple du concept-cible, exprimé dans le langage $\mathcal{L}_e$.
Ex	un exemple du concept-cible, exprimé dans le langage $\mathcal{L}_e$.
$\mathcal{L}_e$	langage de description des exemples.
$\mathcal{L}_h$	langage de description des hypothèses.
$\mathcal{T}$	base de test de cardinal $\|\mathcal{T}\|$.
$\mathcal{T}h$	théorie du domaine.

A.2 Definitions

A.2.1 Ensembles ordonnés - Treillis

Définition A.1 *Relation d'ordre, relation d'ordre strict, relation d'ordre large*

Une relation d'ordre (notée $\mathcal{R}$) est une relation d'ordre large ou une relation d'ordre strict. Une relation d'ordre large (notée $\geq$) est une relation réflexive, antisymétrique et transitive. Une relation d'ordre strict (notée $>$) est une relation irréflexive, antisymétrique et transitive.

Définition A.2 *Ordre total et ordre partiel*

Si la relation d'ordre $\mathcal{R}$ permet de comparer deux éléments quelconques (ie. si elle vérifie $\forall\, e, e' \in E, e \neq e' \Rightarrow (e\ \mathcal{R}\ e'$ ou $e'\ \mathcal{R}\ e)$), alors $\mathcal{R}$ est un ordre total, sinon $\mathcal{R}$ est un ordre partiel.

Définition A.3 *Ensemble ordonné*

Un ensemble E muni d'une relation d'ordre $\geq$ est dit ordonné.

Définition A.4 *Borne supérieure et borne inférieure*

Un élément x est la borne supérieure d'une partie E' d'un ensemble ordonné E si et seulement si x est le plus petit majorant de E' : $(\forall y \in E', x \geq y)$ et $(\forall z \in E, ((\forall y \in E', z \geq y) \Rightarrow z \geq x))$.
Un élément x est la borne inférieure d'une partie E' d'un ensemble ordonné E si et seulement si x est le plus grand minorant de E'.

Propriété A.1

Les bornes supérieures et inférieures n'existent pas toujours dans le cas d'ensembles ordonnés quelconques. Par contre, s'il existe une borne supérieure ou inférieure, alors elle est unique.

Définition A.5 *Sup-demi treillis et inf-demi treillis*

Un ensemble ordonné est un sup-demi (resp. inf-demi) treillis si et seulement si toute paire d'éléments admet une borne supérieure (resp. inférieure).

Définition A.6 *Treillis*

Un ensemble ordonné est un treillis si et seulement si toute paire d'éléments admet une borne supérieure et une borne inférieure.

A.2.2 Le langage de la logique relationnelle

Cette partie définit les éléments nécessaires à la construction des formules de la logique du premier ordre. La figure A.1 donne une vue d'ensemble sur l'imbrication des différents éléments du langage.

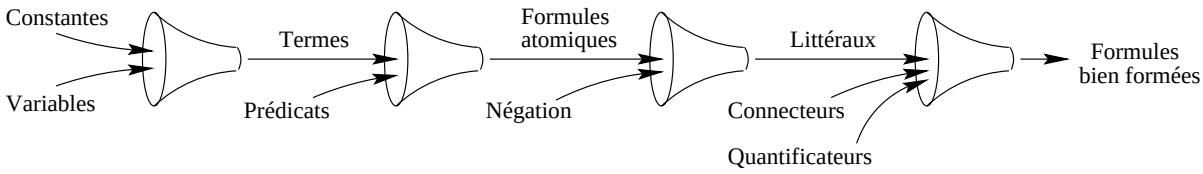


FIG. A.1: Construction des formules de la logique relationnelle.

Définition A.7 *Symboles du langage de la logique du premier ordre*
Le langage de la logique du premier ordre est composé d'un alphabet contenant :

- *les connecteurs* $\neg, \wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow$ *;*
- *les parenthèses* $(,)$ *;*
- *les quantificateurs universels* $\forall$ *et existentiels* $\exists$ *;*

ainsi que les ensembles de symboles suivants :

- *un ensemble infini* $\mathcal{V}$ *de symboles de variables ;*
- *un ensemble fini* $\mathcal{C}$ *de symboles de constantes ;*
- *un ensemble fini* $\mathcal{F}$ *de symboles de fonctions ;*
- *un ensemble fini* $\mathcal{P}$ *de symboles de prédicats.*

À chaque élément de $\mathcal{P}$ *(resp.* $\mathcal{F}$*), est associé un entier positif (resp. strictement positif) appelé arité, représentant le nombre d'arguments du prédicat (resp. de paramètres de la fonction).*

Définition A.8 *Terme et terme clos (ou fermé)*

- *L'ensemble* $\mathcal{T}$ *des termes du langage est le plus petit ensemble satisfaisant :*
  1. *tout symbole de constante ou de variable est un terme ;*
  2. *si* f *est un symbole de fonction d'arité* k *et si* $t_1, t_2, ..., t_k$ *sont des termes alors* $f(t_1, t_2, ..., t_k)$ *est également un terme.*
- *Un terme ne comportant aucun symbole de variable est dit clos (ou encore fermé).*

Définition A.9 *Formule atomique ou atome*
On appelle atome un prédicat appliqué à un nombre correct de termes : si P *est un symbole de prédicat d'arité* k *et* $t_1, t_2, ..., t_k$ *sont des termes, alors* $P(t_1, t_2, ..., t_k)$ *est un atome. Une formule constituée uniquement d'un et un seul atome est dite atomique.*

Définition A.10 *Littéral, littéral positif, littéral négatif*

- *Un littéral est une formule atomique ou la négation d'une formule atomique.*
- *Un littéral positif est un atome.*
- *Un littéral négatif est la négation d'un atome.*

Définition A.11 *Formule (bien formée)*

L'ensemble des formules (bien formées) de la logique du premier ordre est le plus petit ensemble satisfaisant :

- *toute formule atomique est une formule ;*
- *si F est une formule, alors $\neg F$ est une formule ;*
- *si F, G sont des formules, alors $(F \wedge G), (F \vee G), (F \rightarrow G), (F \leftarrow G), (F \leftrightarrow G)$ sont également des formules ;*
- *si F est une formule et v une variable, alors $\forall v F$ et $\exists v F$ sont des formules.*

Définition A.12 *Clause, clause de Horn, clause définie, requête*

- *Une clause est une disjonction finie de littéraux universellement quantifiés. On notera que si $Q_1, ..., Q_n$ et $T_1, ..., T_m$ sont des atomes, alors la clause $\neg Q_1 \vee ... \vee \neg Q_n \vee T_1 \vee ... \vee T_m$ peut s'écrire $T_1 \vee ... \vee T_m \leftarrow Q_1 \wedge ... \wedge Q_n$.*
- *Une clause est dite de Horn si elle comporte au plus un littéral positif.*
- *Une clause est dite définie si c'est une clause de Horn comportant exactement un littéral positif, elle est alors notée : $T \leftarrow Q_1 \wedge ... \wedge Q_n$, T est la tête de la clause, l'ensmble des $\{Q_i\}, 1 \leq i \leq n$ est la queue de la clause.*
- *Une clause unitaire est une clause définie ne comportant aucun littéral négatif et sera notée : $T \leftarrow$.*
- *Une requête est une clause définie dont la tête est vide et peut donc être mise sous la forme : $\leftarrow Q_1 \wedge ... \wedge Q_n$.*

Définition A.13 *Expression*

Une expression peut être :

- *un terme ;*
- *un littéral ;*
- *une conjonction ou une disjonction de littéraux.*

Entre autres, les clauses sont des expressions.

Définition A.14 *Occurrence d'une variable, variable libre, variable liée, formule close*

- *Une occurrence d'une variable dans une formule est un couple constitué de cette variable et d'une place effective, c'est-à-dire tout endroit dans la formule où figure cette variable autre qu'immédiatement après un quantificateur.*

- *Une occurrence d'une variable v dans une formule F est une occurrence libre si elle ne se trouve dans aucune sous-formule de F qui commence par une quantification $\forall v$ ou $\exists v$.*
- *Une variable est libre dans une formule si elle a au moins une occurrence libre dans cette formule. Dans le cas contraire elle est dite liée.*
- *Une formule close est une formule ne contenant aucune variable libre.*

Définition A.15 *Substitution, substitution de renommage, application d'une substitution*

- *Une substitution θ est un ensemble fini de la forme $\{x_1/t_1, ..., x_n/t_n\}, n \geq 0$ où les x_i sont des variables distinctes et les t_i sont des termes; on dit alors que x_i est substitué par t_i.*
- *Soit E une expression et V l'ensemble des variables de E. Une substitution de renommage sur E est une substitution telle que les x_i appartiennent à V et les t_i soient des variables distinctes n'appartenant pas à l'ensemble des variables de E non substituées.*
- *Soit $\theta = \{x_1/t_1, ..., x_n/t_n\}$ une substitution et E une expression. On note $E\theta$ l'instanciation de E par θ, c'est-à-dire l'expression obtenue en remplaçant dans E toutes les occurrences des variables x_i par les termes associés t_i.*

Annexe B

Bases d'apprentissage utilisées

Cette annexe présente les principales bases d'apprentissage sur lesquelles $\mathcal{SIAO}1$ a été testé au chapitre 4. Elles ont été choisies afin de montrer la grande variété de problèmes sur lesquels ce système peut travailler avec succès :

- certains problèmes sont très fortement relationnels alors que d'autres sont définis à l'aide d'une représentation attribut-valeur ;
- certains concepts peuvent être caractérisés de manière exacte alors que d'autres sont particulièrement bruités ou flous ;
- la taille des bases varie de quelques centaines à plusieurs centaines de milliers d'instances ;
- les problèmes peuvent être de nature symbolique, numérique, ou mixte ;
- certaines bases peuvent comporter des valeurs manquantes.

B.1 La base *Census-Income*

Cette base a été extraite par T. Lane et R. Kohavi à partir de données de 1994 et 1995 provenant du bureau du recensement américain (`http ://www.census.gov/ftp/pub/DES/www/welcome.html`) afin de conduire des études sur l'application de techniques d'apprentissage à de gros volumes de données bruitées. La tâche consiste à caractériser les individus dont les revenus sont supérieurs à 50 000 $ US, et en conséquence toutes les données fortement corrélées à ce concept ont été retirées de la base originale (`AGI (adjusted gross income)`, `PEARNVAL (total person earnings)`, `PTOTVAL (total person income)`, `TAXINC (taxable income amount)`, etc.).

Census-Income comprend 299 285 instances réparties en 199 523 instances d'apprentissage et 99 762 instances de test. Ces données occuppent environ 155 Mo. Le nombre d'instances multiples potentiellement conflictuelles s'élève à 46 716 dans la base d'apprentissage et 20 936 dans la base de test. Chacune des instances est décrite à l'aide de 40 attributs, dont 33 sont symboliques et 7 numériques. Les attributs symboliques (par exemple `ADTIND`) ont

jusqu'à 52 valeurs différentes, alors que les attributs numériques peuvent en avoir plusieurs milliers. Le tableau ci-dessous résume les données contenues dans la base :

Contenu du champ	Code	Contenu du champ	Code
age	AAGE	class of worker	ACLSWKR
industry code	ADTIND	occupation code	ADTOCC
education	AHGA	wage per hour	AHRSPAY
enrolled in edu inst last wk	AHSCOL	marital status	AMARITL
major industry code	AMJIND	major occupation code	AMJOCC
mace	ARACE	hispanic Origin	AREORGN
sex	ASEX	member of a labor union	AUNMEM
reason for unemployment	AUNTYPE	full or part time employment stat	AWKSTAT
capital gains	CAPGAIN	capital losses	CAPLOSS
divdends from stocks	DIVVAL	federal income tax liability	FEDTAX
tax filer status	FILESTAT	region of previous residence	GRINREG
state of previous residence	GRINST	detailed household and family stat	HHDFMX
detailed household summary in household	HHDREL	migration code-change in msa	MIGMTR1
migration code-change in reg	MIGMTR3	migration code-move within reg	MIGMTR4
live in this house 1 year ago	MIGSAME	migration prev res in sunbelt	MIGSUN
num persons worked for employer	NOEMP	family members under 18	PARENT
country of birth father	PEFNTVTY	country of birth mother	PEMNTVTY
country of birth self	PENATVTY	citizenship	PRCITSHP
own business or self employed	SEOTR	fill inc quest for veteran's admin	VETQVA
veterans benefits	VETYN	weeks worked in year	WKSWORK

TAB. B.1: Les 40 attributs de la base *census-income*.

B.2 La base *Iris*

Cette base, qui fait référence dans le domaine de la reconnaissance des formes, décrit 150 exemples de fleurs de la famille des iris à l'aide de 4 attributs numériques représentant la longueur et la largeur des pétales et des sépales. Les données ont été récoltées en 1936 par R. A. Fisher [Fisher36]. Trois classes de fleurs sont représentées : Iris Setosa, Iris Versicolor et Iris Virginica. S'il est facile de caractériser les Iris Setosa grâce à la largeur de leurs sépales, les classes Iris Versicolor et Iris Virginica sont plus difficiles à distinguer car elles ne sont pas linéairement séparables.

B.3 La base *Mushrooms*

Mushrooms (également connue sous le nom *agaricus-lepiota*) décrit un échantillon de 23 espèces de champignons à lamelles appartenant aux familles des lépiotes et des amanites. Les données, générées à partir de [Lincoff81], comprennent 8 124 instances représentées à l'aide de 23 attributs symboliques. 2 480 valeurs sont manquantes. Le but de l'apprentissage consiste à distinguer les champignons comestibles des champignons empoisonnés ou non recommandés. Ce problème fait référence dans le cadre de l'apprentissage disjonctif, car il n'existe pas, dans le formalisme attribut-valeur, de règle unique déterminant la classe d'un champignon.

B.4 La base *Mutagenesis*

Cette base issue de données réelles provient d'une recherche conjointe menée par A. Srinivasan et S.H. Muggleton de l'Oxford University Computing Laboratory d'une part, et R.D. King et M.J.E Sternberg du Biomolecular Modelling Laboratory de l'Imperial Cancer Research Fund de Londres d'autre part, dont le but était d'étudier l'application de méthodes de PLI dans le cadre de la chimie moléculaire [Srinivasan et al.94].

Le problème *mutagenesis* consiste à déterminer parmi les molécules nitro-aromatiques que l'on trouve dans les gaz d'échappement des voitures celles dont le logarithme du degré mutagène est positif (on sait que l'activité mutagène d'une molécule est liée à son effet cancérigène). L'étude porte sur 230 molécules, dont 138 sont classées mutagènes.

Dans un premier temps, les auteurs de [Debnath et al.91] ont proposé un modèle pour le logarithme du degré mutagène obtenu par regression linéaire sur quatre attributs jugés pertinents et extraits des données dont ils disposaient :

1. $logP$ est le logarithme du ratio octane/eau, qui donne une mesure de l'hydrophobicité ;
2. ε_{LUMO} est l'énergie de la plus basse orbitale non saturée des composés de la molécule (obtenu à l'aide d'un modèle de mécanique moléculaire quantique) ;
3. I_1 est un indicateur vrai pour toutes les molécules comportant trois cycles ou plus ;
4. I_a est un indicateur vrai pour :

   > " ... five examples of acenthrylenes and shows that these are much less active than expected for some unknown reason" ([Srinivasan et al.94] p. 219, citant [Debnath et al.91] p.788).

Le modèle est obtenu à l'aide de 188 molécules intégrables dans la régression, les 42 autres étant des points isolés et éloignés inadéquats.

Dans un second temps [Srinivasan et al.94] ont entrepris de décrire les molécules de cette base en logique relationnelle à l'aide du système de modélisation moléculaire QUANTA, ce qui a permi l'obtention rapide d'une description basée sur les prédicats `bond(compound, atom1, atom2, bondtype)` et `atm(compound, atom, element, atomtype, charge)` où :

- `compound`, `atom`, `atom1` et `atom2` sont des variables permettant de lier les éléments de la structure ;
- `bondtype` est l'un des 8 types de liens générés automatiquement par QUANTA ;
- `element` correspond au symbole de l'atome correspondant dans le tableau de la classification périodique ;

- `atomtype` correspond au numéro atomique de l'atome ;
- `charge` correspond à la charge électrique.

La base obtenue comporte 12 203 faits décrivant les 230 molécules.

B.5 La base *1000-trains*

Cette base a été créée dans le but d'étendre le fameux problème des trains de Michalski [Michalski80]. La tâche consiste à distinguer deux concepts (les trains allant à l'est et les trains allant à l'ouest), le concept-cible étant uniquement trains allant à l'est . Chaque train est un objet composé de parties (les wagons) liées par une relation de succession. La base comporte 1000 exemples d'apprentissage et 1000 exemples de test, les concepts étant parfaitement équilibrés.

La représentation utilisée dans $\mathcal{SIAO}1$ met en œuvre deux symboles de prédicat :

- `Coach(pos, n_pos, shape, color, length, nwheel, load)`
- `Next(pos, pos)`

Ce codage est sémantiquement équivalent à celui de REGAL et G-Net.

Annexe C

Détail des résultats expérimentaux

C.1 Parallélisation de $\mathcal{SIAO}1$

Les tableaux ci-après, dont sont extraits les graphiques du paragraphe 5.5.3, contiennent les résultats qui proviennent des exécutions du programme réparti sur des processeurs dédiés de la machine IBM SP2. Ils détaillent donc le comportement expérimental de T_p^{ratio} en fonction du type d'architecture ("RCD", acronyme de "Répartition de Charge Dynamique" fait référence à l'architecture maître-esclave asynchrone, alors que "RCS", acronyme de "Répartition de Charge Statique" fait référence à l'architecture pipeline), du nombre p de processeurs et du type de communication ("Norm." dénote une communication sans utilisation d'aucune mémoire tampon, et "L(i)" l'utilisation d'une liste de taille i). A partir des estimations obtenues pour T_p^{ratio}, il a été possible de dériver des estimations pour l'efficacité (notée Eff) et pour $p.T_p^{ratio} - T_1^{ratio}$. Les données figurant sur chaque ligne du tableau correspondent à cinq essais ; on note E les valeurs moyennes et σ les écarts-types.

C.2 Etude de T_p^{ratio} sur la base *Mushrooms*

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCD	Norm.	1	0.363097	0.00201095	1	0.00551486	0
RCD	Norm.	2	0.189208	0.00068182	0.959528	0.0034527	0.015320
RCD	Norm.	3	0.131698	0.00033530	0.919022	0.0023392	0.031996
RCD	Norm.	4	0.103006	0.00008547	0.881254	0.0007308	0.048926
RCD	L(1)	1	0.365462	0.00072737	1	0.0019887	0
RCD	L(1)	2	0.190203	0.00061168	0.960724	0.0030886	0.014944
RCD	L(1)	3	0.131952	0.00027620	0.923227	0.0019296	0.030392
RCD	L(1)	4	0.103485	0.00010841	0.882889	0.0009259	0.048477

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCD	L(2)	1	0.366237	0.00185994	1	0.0050636	0
RCD	L(2)	2	0.192010	0.00324938	0.953966	0.0160544	0.017782
RCD	L(2)	3	0.131220	0.00049326	0.930288	0.0034790	0.027423
RCD	L(2)	4	0.102776	0.00011465	0.890865	0.0009939	0.044866
RCD	L(3)	1	0.367054	0.00113307	1	0.0030866	0
RCD	L(3)	2	0.189446	0.00070741	0.968768	0.0036121	0.011838
RCD	L(3)	3	0.131185	0.00032556	0.932668	0.0023064	0.026501
RCD	L(3)	4	0.103392	0.00054071	0.887557	0.0046471	0.046512

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCS	Norm.	1	0.3507960	0.00243458	1	0.0069406	0
RCS	Norm.	2	0.1910900	0.00014345	0.917882	0.0006892	0.0313839
RCS	Norm.	3	0.1385230	0.00035440	0.844140	0.0021615	0.0647731
RCS	Norm.	4	0.1081100	0.00026616	0.811205	0.0019965	0.0816446
RCS	L(1)	1	0.3486020	0.00099018	1	0.0028351	0
RCS	L(1)	2	0.1895110	0.00050533	0.919746	0.0024529	0.0304205
RCS	L(1)	3	0.1334410	0.00034971	0.870809	0.0022779	0.0517202
RCS	L(1)	4	0.1045690	0.00023443	0.833433	0.0018693	0.0696726

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCS	L(2)	1	0.3502370	0.00135740	1	0.0038696	0
RCS	L(2)	2	0.1837590	0.00022265	0.952982	0.0011537	0.0172803
RCS	L(2)	3	0.1292810	0.00029074	0.903044	0.0020302	0.0376054
RCS	L(2)	4	0.1011530	0.00007348	0.865611	0.0006287	0.0543758
RCS	L(3)	1	0.3499780	0.00102326	1	0.0029204	0
RCS	L(3)	2	0.1841050	0.00072569	0.950501	0.0037247	0.0182313
RCS	L(3)	3	0.1295010	0.00034476	0.900847	0.0023918	0.0385235
RCS	L(3)	4	0.1032150	0.00246166	0.848159	0.0195525	0.0628813

C.3 Etude de T_{Com} sur la base *Mushrooms*

Le tableau ci-dessous fournit pour chaque type d'architecture : le coefficient de corrélation r calculé à partir des différentes valeurs de T_{Com} calculées plus haut ($T_{Com}(n,p) = p.T_p(n) - T_1(n)$), la probabilité α que cette valeur de r soit dépassée en valeur absolue pour deux variables aléatoires indépendantes, ainsi que l'équation Δ de la droite de régression obtenue.

Archi.	Com.	r	α	Δ
RCD	Norm.	0.999735	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0163454\ p\ -\ 0.016803$
RCD	L(1)	0.998961	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0160879\ p\ -\ 0.0167665$
RCD	L(2)	0.993933	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0144239\ p\ -\ 0.013542$
RCD	L(3)	0.992919	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0154199\ p\ -\ 0.017337$
RCS	Norm.	0.991108	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0278323\ p\ -\ 0.0251303$
RCS	L(1)	0.992446	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0230317\ p\ -\ 0.019626$
RCS	L(2)	0.999334	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0183452\ p\ -\ 0.0185477$
RCS	L(3)	0.997812	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0208936\ p\ -\ 0.022325$

C.4 Etude de T_p^{ratio} sur la base *Iris*

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCD	Norm.	1	0.0178160	0.00015164	1	0.0084945	0
RCD	Norm.	2	0.0114988	0.00026217	0.775095	0.0176325	0.00518160
RCD	Norm.	3	0.0090883	0.00011115	0.653538	0.0080934	0.00944896
RCD	Norm.	4	0.0085565	0.00027230	0.521067	0.0165671	0.01641000
RCD	Norm.	5	0.0086933	0.00016213	0.410018	0.0076883	0.02565095
RCD	L(1)	1	0.0168298	0.00034286	1	0.0200593	0
RCD	L(1)	2	0.0103092	0.00013362	0.816387	0.0106519	0.00378860
RCD	L(1)	3	0.0080433	0.00011496	0.697602	0.0100120	0.00730037
RCD	L(1)	4	0.0068722	0.00012439	0.612437	0.0110522	0.01066124
RCD	L(1)	5	0.0064373	0.00023442	0.523581	0.0192572	0.01535695

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCD	L(2)	1	0.0166946	0.00022074	1	0.0132025	0
RCD	L(2)	2	0.0097839	0.00016595	0.853410	0.0146600	0.00287332
RCD	L(2)	3	0.0075666	0.00029314	0.736534	0.0281995	0.00600547
RCD	L(2)	4	0.0067959	0.00016023	0.614476	0.0143032	0.01048916
RCD	L(2)	5	0.0065874	0.00595656	0.506931	0.0059565	0.01624260
RCD	L(3)	1	0.0168944	0.00026722	1	0.0160129	0
RCD	L(3)	2	0.0096817	0.00010739	0.872591	0.0096505	0.00246916
RCD	L(3)	3	0.0077640	0.00027975	0.726247	0.0256306	0.00639781
RCD	L(3)	4	0.0068169	0.00013658	0.619824	0.0125552	0.01037344
RCD	L(3)	5	0.0071408	0.00021935	0.473624	0.0144765	0.01880960

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCS	Norm.	1	0.0198672	0.00007902	1	0.0039718	0
RCS	Norm.	2	0.0153950	0.00002710	0.645249	0.0011330	0.01092280
RCS	Norm.	3	0.0161954	0.00033298	0.409075	0.0081835	0.02871900
RCS	Norm.	4	0.0148133	0.00101485	0.336726	0.0209228	0.03938600
RCS	Norm.	5	0.0162968	0.00068873	0.244249	0.0102168	0.06161680
RCS	L(1)	1	0.0181054	0.00002710	1	0.0014936	0
RCS	L(1)	2	0.0126440	0.00008131	0.715998	0.0046013	0.0071826
RCS	L(1)	3	0.0121727	0.00005224	0.495803	0.0021143	0.0184127
RCS	L(1)	4	0.0109674	0.00006762	0.412724	0.0025406	0.0257642
RCS	L(1)	5	0.0099197	0.00005347	0.365047	0.0019769	0.0314935

Archi.	Com.	p	$E(T_p^{ratio})$	$\sigma(T_p^{ratio})$	$E(Eff)$	$\sigma(Eff)$	$p.E(T_p^{ratio}) - E(T_1^{ratio})$
RCS	L(2)	1	0.0181553	0.00017989	1	0.0097963	0
RCS	L(2)	2	0.0123599	0.00003248	0.734447	0.0019282	0.0065645
RCS	L(2)	3	0.0118997	0.00003231	0.508567	0.0013795	0.0175438
RCS	L(2)	4	0.0110592	0.00003180	0.410415	0.0011785	0.0260815
RCS	L(2)	5	0.0098239	0.00008069	0.369638	0.0030214	0.0309644
RCS	L(3)	1	0.0179156	0.00003231	1	0.0018050	0
RCS	L(3)	2	0.0121504	0.00006463	0.737264	0.0039301	0.0063852
RCS	L(3)	3	0.0120661	0.00010619	0.494968	0.0043889	0.0182827
RCS	L(3)	4	0.0112421	0.00011964	0.398451	0.0042238	0.0270528
RCS	L(3)	5	0.0099890	0.00005126	0.358715	0.0018379	0.0320296

C.5 Etude de T_{Com} sur la base *Iris*

Archi.	Com.	r	α	Δ
RCD	Norm.	0.987943	$\alpha \leq 10^{-2}$	$T_{Com} = 0.00625303\ p\ -\ 0.00742079$
RCD	L(1)	0.998292	$\alpha \leq 10^{-2}$	$T_{Com} = 0.00375865\ p\ -\ 0.00385453$
RCD	L(2)	0.988587	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0040101\ p\ -\ 0.0049082$
RCD	L(3)	0.97305	$\alpha \leq 10^{-2}$	$T_{Com} = 0.00455235\ p\ -\ 0.00604704$
RCS	Norm.	0.992565	$\alpha \leq 10^{-2}$	$T_{Com} = 0.0151697\ p\ -\ 0.0173801$
RCS	L(1)	0.994053	$\alpha \leq 10^{-2}$	$T_{Com} = 0.00815686\ p\ -\ 0.00789998$
RCS	L(2)	0.993009	$\alpha \leq 10^{-2}$	$T_{Com} = 0.00814458\ p\ -\ 0.0082029$
RCS	L(3)	0.991683	$\alpha \leq 10^{-2}$	$T_{Com} = 0.00847268\ p\ -\ 0.00866798$

Annexe D

Interface graphique de $\mathcal{SIAO}1$

L'algorithme $\mathcal{SIAO}1$ présenté dans cette thèse a été implanté en C++ et pourvu d'une interface graphique appelée EvoMiner. La première section de cette annexe décrit comment utiliser cette interface pour effectuer un apprentissage, et éventuellement modifier les paramètres par défaut de l'algorithme. La seconde section décrit un certain nombre de fonctionnalités graphiques offertes par EvoMiner. Cette extension permet à l'utilisateur d'obtenir visuellement un grand nombre d'informations sur la base d'apprentissage en cours d'exploitation.

D.1 Exemple d'apprentissage sur la base des Iris

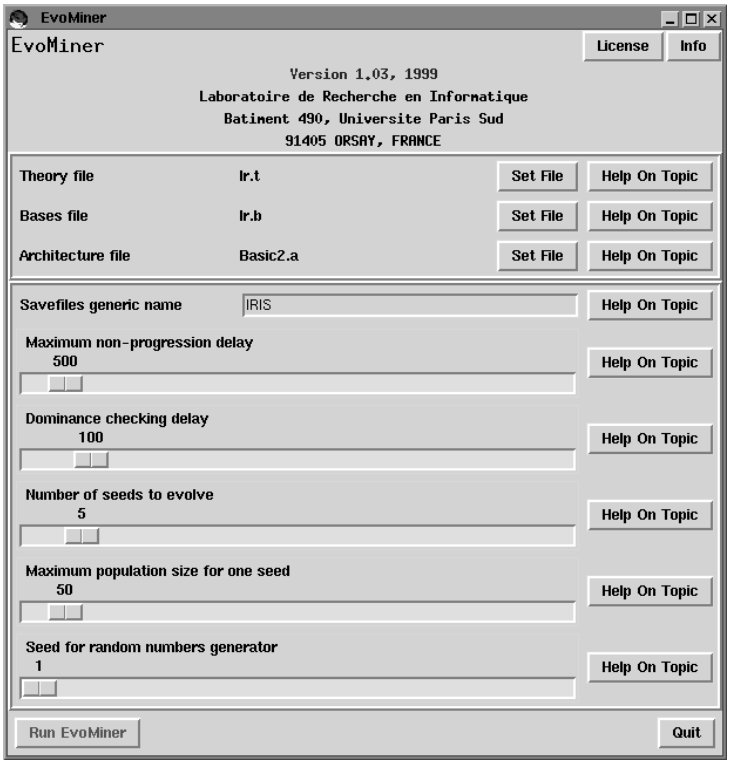


FIG. D.1: Menu de lancement de $\mathcal{SIAO}1$.

Le menu de lancement (figure D.1) permet à l'utilisateur de spécifier les données fournies en entrée et les paramètres de $\mathcal{SIAO}1$:

- Un fichier (dont l'extension est `.t`) décrit la grammaire du langage (les prédicats et leurs arités, les différents types des constantes, les relations de généralité entre prédicats et entre constantes, etc.).
- Un fichier (dont l'extension est `.b`) contient la description de toutes les instances. Différents mots-clé permettent soit de délimiter une base d'apprentissage et une base de test figées, soit d'extraire une base de test (par tirage aléatoire respectant ou non les pondérations des différentes classes), soit de spécifier un schéma de validation spécifique (par exemple la k-validation croisée).
- Un fichier (dont l'extension est `.a`) n'est utilisé que lors d'une session distribuée et fournit les noms des machines exploitées. Le reste de l'architecture (maître-esclave ou pipeline, taille – éventuellement nulle – de la mémoire tampon) est par contre fixé dès la compilation de $\mathcal{SIAO}1$.
- Les paramètres spécifiques à l'algorithme (nombre de noyaux par classe-cible, taille maximum des populations, etc.) peuvent également être modifés à cette étape.

Une fois le menu de lancement validé, l'éditeur (dont certaines fonctionnalités seront décrites dans la section suivante) apparaît. Entrer `Quit` (ou `Q`) dans cet éditeur permet de lancer immédiatement l'algorithme d'apprentissage. Au cours de l'exécution, une fenêtre rend compte de l'état d'avancement de l'algorithme (évolution des noyaux, couverture des concepts, taux d'utilisation des opérateurs d'évolution, etc.).

EvoMiner Status Window

EvoMiner Evolution Monitor

General information

Generation number	5400	Time (s)	21

Classes coverage for seeds that have converged

Classes	1	2	3
Coverage (occurences)	49	47	25
Coverage (weights)	49.0	47.0	25.0

Sub-populations status

No	Class	Pop. Size	Best Fit.	Mut. Rate	Cv. Rate	Status
1	1	36	0.750001	61.2936	100	STOPPED
2	2	50	0.909092	27.6515	3.4	RUNNING
3	3	43	0.500001	63.2438	100	STOPPED
4	1	46	0.666668	64.0281	100	STOPPED
5	2	50	0.959185	18.8871	100	STOPPED
6	3	50	0.955557	21.228	84.2	RUNNING
7	1	41	0.714287	62.6727	100	STOPPED
8	2	50	0.882354	55.3558	100	STOPPED
9	3	50	0.868422	34.2107	10.8	RUNNING
10	1	48	0.600001	64.5397	100	STOPPED
11	2	1	0.500001	50.0	100	STOPPED
12	3	50	0.777779	18.9988	100	STOPPED
13	1	50	0.961539	30.5909	100	STOPPED
14	2	50	0.925927	29.8831	38.6	RUNNING
15	3	50	0.904763	67.4373	100	STOPPED

FIG. D.2: Fenêtre d'état de $\mathcal{SIAO}1$.

Lorsque l'apprentissage est terminé, l'éditeur est mis à jour avec les résultats obtenus, qui peuvent être facilement extraits par copier-coller pour utilisation à l'extérieur ou dans l'éditeur lui-même. Ces derniers comprennent :

- *Le résultat de l'apprentissage :* les règles, ainsi que leur qualité, leur cohérence et leur complétude.
- *Des mesures de la qualité du résultat de l'apprentissage :* la précision et les matrices de confusion sur les bases d'apprentissage et de test sont également fournies.
- *Des traces concernant l'apprentissage :* il est par exemple possible de tracer le processus d'élagage des règles. Pour des raisons évidentes de performance, les traces désirées doivent être spécifiées lors de la compilation du code de $\mathcal{SIAO}1$.
- *Des mesures de performance :* ces dernières comprennent la durée d'exécution de l'algorithme et différents indices de performance comme par exemple le nombre d'appariements par seconde ou encore la performance des différents filtres (filtre simple, filtre tabou). Dans le cas de la version parallèle de l'algorithme, des mesures globales et détaillées (machine par machine) sont également fournies.

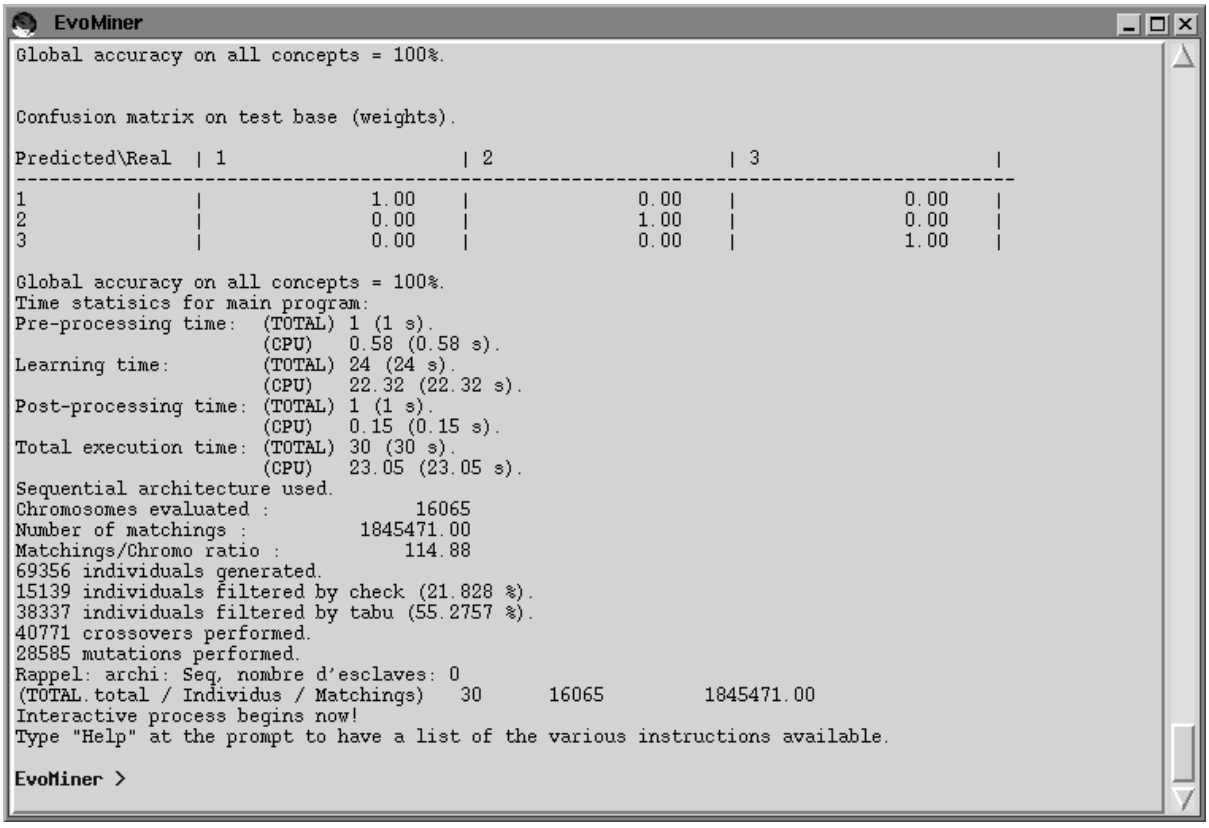


Fig. D.3: Le résultat de l'apprentissage figure dans l'éditeur.

D.2 Les fonctionnalités de visualisation des données

L'éditeur d'EvoMiner a été enrichi d'une interface de visualisation des données permettant à l'utilisateur de prendre connaissance de nombreuses informations concernant :

- la base en cours de traitement ;
- le fonctionnement de $\mathcal{SIAO}1$.

En effet, bien que le cycle d'extraction de la connaissance à partir des données dans son ensemble (tel qu'il est décrit figure 1.2) soit un sujet très – trop – vaste, il est rapidement apparu que le développement d'un environnement autour de l'algorithme facilite considérablement son utilisation dans des contextes très variés. Les deux principales fonctions exploratoires sont :

- `ShowNumericType` (ou `SNT`) affiche les principales caractéristiques –relevées sur la base d'apprentissage– d'un type numérique. La fenêtre `Statistics` montre sur l'ensemble de la base d'apprentissage ou pour chaque classe-cible les bornes minimum et maximum des valeurs, leur moyenne et leur écart-type, alors que la fenêtre `Stats View` fournit cette information de manière graphique et intuitive.

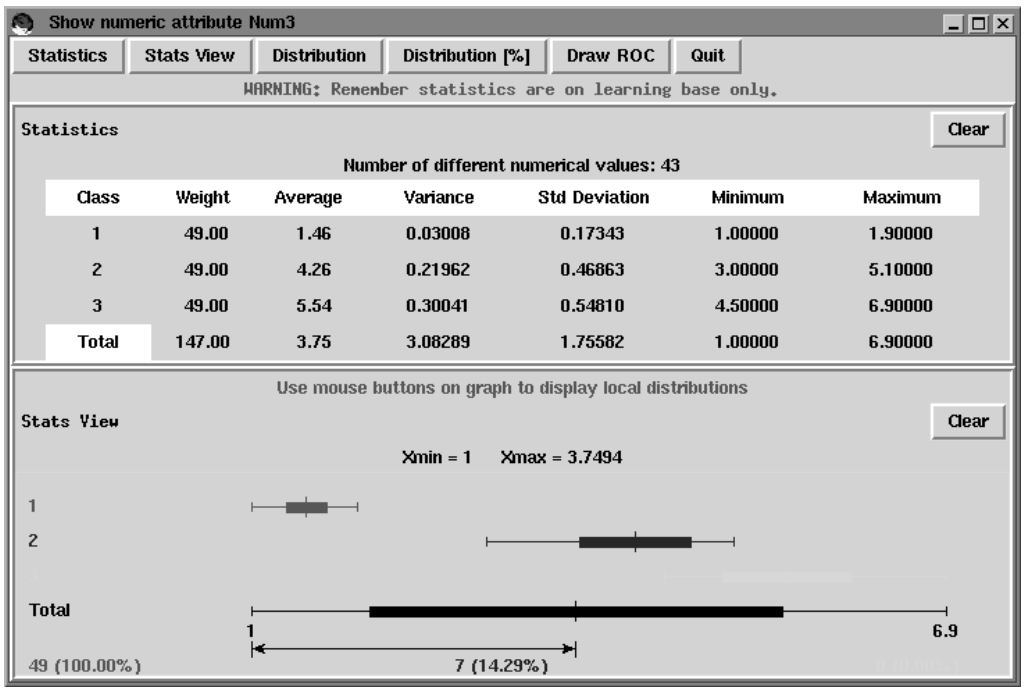


FIG. D.4: Fenêtre de description d'un type numérique.

La figure D.4 montre que les fonctionnalités exploratoires comportent également des mécanismes interactifs : dans le cas présent l'utilisateur peut générer à l'aide de la souris une plage de valeurs (dans la partie inférieure de la fenêtre `Stats View`) et obtenir des informations sur la répartition des classes dans cet ensemble. Cette

partie de l'interface permet également d'observer la distibution absolue ou relative des valeurs (l'utilisateur fixe le nombre de segments désirés), ou encore d'obtenir une courbe ROC montrant le comportement d'un classeur primitif n'exploitant que le type numérique sélectionné pour classer les instances de la base.

– `ShowSymbolicType` (ou `SST`) affiche les principales caractéristiques –relevées sur la base d'apprentissage– d'un type symbolique :

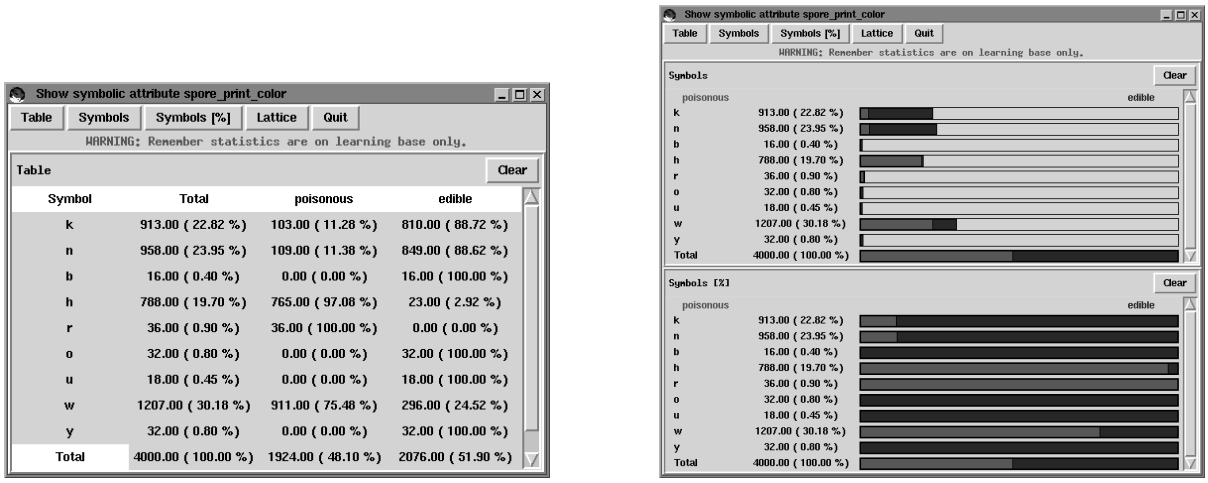


FIG. D.5: Fenêtre de description d'un type numérique.

On mentionnera également quelques autres commandes utiles :

– `ShowNumericGraph` (ou `SNG`) permet de visualiser une relation possible entre deux types numériques.

– `HyperplaneAnalysis` (ou `HA`) fournit des informations sur les attributs numériques introduits lors de la phase de complétion numérique par la méthode d'analyse discriminante.

– `FitnessStatistics` (ou `FS`) permet d'analyser la qualités des individus au cours de l'exécution, entre autres en fonction des opérateurs génétiques qui les ont générés. L'analyse des graphiques produits par cette commande a motivé l'auto-adaptation des taux de mutation et de croisement.

– `FilterHitMissStats` (ou `FHMS`) permet de visualiser et analyser le comportement des filtres (simple et tabou).

L'utilisateur se rapportera à l'aide de l'éditeur (`Help` ou `H`) pour obtenir une liste exhaustive des commandes (test d'hypothèses sur la base d'apprentissage (`TestDefinition` ou `TD`), etc.).

Bibliographie

[Adriaans et al.96] Adriaans (P.) et Zantinge (D.). – *Data Mining.* – Addison-Wesley, 1996.

[Aler et al.98] Aler (R.), Borrajo (D.) et Isasi (P.). – Genetic programming and deductive-inductive learning : a multi-strategy approach. *In : Proceedings of the fifteenth international conference on machine learning ICML'98*, pp. 10–18.

[Alphonse et al.99] Alphonse (É.) et Rouveirol (C.). – Selective propositionalization for relational learning. *In : Third European Conference on Principles of Data Mining and Knowledge Discovery PKDD'99, Prague, Czech Republic, J.M. Zytkow et J. Rauch (Eds), LNAI 1704*, pp. 271–276.

[Amdahl67] Amdahl (G.M.). – Validity of the single-processor approach to achieving large scale computing capabilities. *In : AFIPS Conference Proceedings.* pp. 483–485. – AFIPS Press, Reston, VA.

[Angelano et al.97] Angelano (C.), Giordana (A.), Lo Bello (G.) et Saitta (L.). – A network genetic algorithm for concept learning. *In : Proceedings of the seventh International Conference on Genetic Algorithms ICGA'97*, pp. 434–441.

[Angelano et al.98] Angelano (C.), Giordana (A.), Lo Bello (G.) et Saitta (L.). – An experimental evaluation of coevolutive concept learning. *In : Proceedings of the fifteenth international conference on machine learning ICML'98*, pp. 19–27.

[Antonisse89] Antonisse (J.). – A new interpretation of schema notation that overturns the binary encoding constraint. *In : Proceedings of the third International Conference on Genetic Algorithms ICGA'89, J.D. Schaffer (Ed), Morgan Kaufmann*, pp. 86–91.

[Atmar92] Atmar (W.). – On the rules and nature of simulated evolutionary programming. *In : Proceedings of the first Annual Conference on Evolutionary Programming EP'92, D.B. Fogel et W. Atmar (Eds), Evolutionary Programming Society*, pp. 17–26.

[Augier et al.95] Augier (S.), Venturini (G.) et Kodratoff (Y.). – Learning first order logic rules with a genetic algorithm. *In : KDD-95 Proceedings of The First International Conference on Knowledge Discovery & Data Mining, Montréal (Canada), U.M. Fayyad and R. Uthurusamy (Eds)*, pp. 21–26.

[Augier et al.96] Augier (S.) et Venturini (G.). – SIAO1, a first order logic machine learning system using genetic algorithms. *In : ICML'96 13th International Conference on Machine Learning, Bari (Italy), Proceedings of the pre-conference workshop on Evolutionary Algorithms and Machine Learning*, pp. 9–16.

[Augier99] Augier (S.). – Changement d'échelle d'un algorithme d'apprentissage automatique. *In : Actes de la Conférence d'Apprentissage CAP'99, Plate-forme AFIA'99, Palaiseau, Ecole Polytechnique, M. Sebag (Ed), 1999*, pp. 235–242.

[Back et al.96] Bäck (T.) et Schwefel (H.-P.). – Evolutionary computation : An overview. *In : Proceedings of the third IEEE Conference on Evolutionary Computation ICEC'96, IEEE Press, Piscataway NJ*, pp. 20–29.

[Back96] Bäck (T.). – *Evolutionary Algorithms in Theory and Practice.* – Oxford University Press, 1996.

[Badea et al.99] Badea (L.) et Stanciu (M.). – Refinement operators can be (weakly) perfect. *In : Inductive Logic Programming, Ninth International Workshop ILP'99, LNAI 1634*, pp. 21–32.

[Baker87] Baker (J.E.). – Reducing bias and inefficiency in the selection algorithm. *In : Genetic Algorithms and their Applications : Proceedings of the second International Conference ICGA'87, J.J. Grefenstette (Ed), Lawrence Erlbaum Associates, Hillsdale, NJ*, pp. 14–21.

[Bauer et al.99] Bauer (E.) et Kohavi (R.). – An empirical comparison of voting classification algorithms : Bagging, boosting and variants. *Machine Learning*, vol. 36, 1999, pp. 105–139.

[Baxter77] Baxter (L.D.). – The NP-completeness of subsumption. *In : manuscrit souvent cité mais non publié.*

[Belding95] Belding (T.C.). – The distributed genetic algorithm revisited. *In : Proceedings of the sixth International Conference on Genetic Algorithms ICGA'95, L.J. Eshelman (Ed), Morgan Kaufmann*, pp. 114–121.

[Blockeel et al.98] Blockeel (H.) et De Raedt (L.). – Top-down induction of first-order logical decision trees. *Artificial Intelligence*, vol. 101 (1-2), 1998, pp. 285–297.

[Blockeel et al.99] Blockeel (H.), De Raedt (L.), Jacobs (N.) et Demoen (B.). – Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, vol. 3 (1), 1999, pp. 59–93.

[Blumer et al.87] Blumer (A.), Ehrenfeucht (A.), Haussler (D.) et Warmuth (M.K.). – Occam's razor. *Information Processing Letters*, no24, 1987, pp. 377–380. – Édité également dans Readings in Machine Learning, J.W. Shavlik et T.G. Dietterich (Eds), Morgan Kaufmann, 1990.

[Breimann96] Breimann (L.). – Bagging predictors. *Machine Learning*, vol. 24, 1996, pp. 123–140.

[Brezellec99] Brézellec (P.). – Sondage et apprentissage. *In : Actes de la Conférence d'Apprentissage CAP'99, Plate-forme AFIA'99, Palaiseau, Ecole Polytechnique, M. Sebag (Ed), 1999*, pp. 63–68.

[Brunet93] Brunet (T.). – Le problème de la résistance aux valeurs inconnues dans l'induction : une forêt de branches. *In : Actes des huitièmes journées francophones sur l'apprentissage JFA'93, Saint Raphaël.* – Les pages du recueil ne sont pas numérotées.

[Brunet94] Brunet (T.). – Présentation de StatLog (projet ESPRIT 5170). *In : Actes des* $4^{èmes}$ *Journées sur l'Induction Symbolique / Numérique*, pp. 145–148.

[BT et al.96] Brunie-Taton (A.) et Cornuéjols (A.). – Classification en programmation génétique. *In : 11èmes Journées Françaises d'Apprentissage (JFA'96), Sète, France*, pp. 303–316.

[Cerf94] Cerf (R.). – *Une théorie asymptotique des algorithmes génétiques.* – PhD thesis, Université de Montpellier II, 1994.

[Clark et al.89] Clark (P.) et Niblett (T.). – The CN2 induction algorithm. *Machine Learning*, vol. 3 (4), 1989, pp. 261–283.

[Clark et al.91] Clark (P.) et Boswell (R.). – Rule induction with CN2 : Some recent improvements. *In : Proceedings of the fifth European Working Session on Learning EWSL'91, Y. Kodratoff (Ed), Springer-Verlag, LNAI 482*, pp. 151–163.

[Cochran77] Cochran (W.G.). – *Sampling Techniques.* – Wiley, 1977.

[Collins et al.91] Collins (R.J.) et Jefferson (D.R.). – Selection in massively parallel genetic algorithms. *In : Proceedings of the fourth International Conference on Genetic Algorithms ICGA'91, R.K. Belew et L.B. Booker (Eds), Morgan Kaufmann*, pp. 249–256.

[CP et al.97] Cantù-Paz (E.) et Goldberg (D.E.). – Predicting speedups of idealized bounding cases of parallel genetic algorithms. *In : Proceedings of the seventh International Conference on Genetic Algorithms ICGA'97*, pp. 113–120.

[CP97a] Cantù-Paz (E.). – *Designing Efficient Master-Slave Parallel Genetic Algorithms.* – Technical Report 97004, Nashville, TN, Illinois Genetic Algorithms Laboratory, May 1997. IlliGAL Report.

[CP97b] Cantù-Paz (E.). – *A Survey of Parallel Genetic Algorithms.* – Technical Report 97003, Nashville, TN, Illinois Genetic Algorithms Laboratory, May 1997. IlliGAL Report.

[CP98] Cantù-Paz (E.). – *Designing Scalable Multi-Population Parallel Genetic Algorithms.* – Technical Report 98009, Nashville, TN, Illinois Genetic Algorithms Laboratory, May 1998. IlliGAL Report.

[Cramer85] Cramer (N.L.). – A representation for the adaptive generation of simple sequential programs. *In : Proceedings of the first International Conference on Genetic Algorithms and their applications ICGA'85, J.J. Grefenstette (Ed), Lawrence Erlbaum Associates Publishers*, pp. 183–187.

[Davis89] Davis (L.). – Adapting operator probabilities in genetic algorithms. *In : Proceedings of the third International Conference on Genetic Algorithms ICGA'89, J.D. Schaffer (Ed), Morgan Kaufmann*, pp. 61–69.

[De jong et al.91] De Jong (K.A.) et Spears (W.M.). – Learning concept classification rules using genetic algorithms. *In : Proceedings of the Twelfth International Joint Conference on Artificial Intelligence IJCAI'91*, pp. 651–656.

[De jong et al.93] De Jong (K.A.), Spears (W.M.) et Gordon (D.F.). – Using genetic algorithms for concept learning. *Machine Learning*, vol. 13, 1993, pp. 161–188.

[De jong et al.95] De Jong (K.A.) et Sarama (J.). – On decentralizing selection algorithms. *In : Proceedings of the sixth International Conference on Genetic Algorithms ICGA'95, L.J. Eshelman (Ed), Morgan Kaufmann*, pp. 17–23.

[De jong90] De Jong (K.A.). – *Machine Learning - An Artificial Intelligence Approach, Volume III*, chap. Genetic-Algorithm-Based Learning, pp. 611–638. – Morgan Kaufmann, 1990.

[De jong92] De Jong (K.A.). – Genetic algoritms are NOT function optimizers. *In : Foundations of Genetic Algorithms 2 FOGA'92, L.D. Whitley (Ed), Morgan Kaufmann*, pp. 5–18.

[De raedt97] De Raedt (L.). – Logical settings for concept-learning. *Artificial Intelligence*, vol. 95, 1997, pp. 187–201.

[Debnath et al.91] Debnath (A.K.), de Compadre (R.L. Lopez), Debnath (G.), Schusterman (A.J.) et Hansch (C.). – Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, vol. 34(2), 1991, pp. 786–797.

[Diday et al.00] Diday (E.), Kodratoff (Y.), Brito (B.) et Moulet (M.). – *Induction symbolique-numérique à partir de données.* – Cépaduès-Édtions, 2000.

[Dietterich97] Dietterich (T.G.). – Machine learning research - four current directions. *AI Magazine*, vol. 18 (4), Winter 1997, pp. 97–136.

[Doorly95] Doorly (D.). – *Genetic Algorithms in Engeneering and Computer Science*, chap. Parallel Genetic Algorithms for Optimization in CFD, pp. 251–270. – Wiley, 1995.

[Ehrenburg96] Ehrenburg (H.). – Improved directed acyclic graph evaluation and the combine operator in genetic programming. *In : Genetic Programming 1996 : Proceedings of the first annual conference GP'96, J.R. Koza, D.E. Goldberg, D.B. Fogel et R.L. Riolo (Eds), MIT Press*, pp. 285–291.

[Fisher36] Fisher (R. A.). – The use of multiple measurements in taxonomic problems. *Annual Eugenics*, no7, Part II, 1936, pp. 179–188.

[Fleury96] Fleury (L.). – *Mesure de la Qualité d'une Règle, Elimination des Redondances, Proposition d'Algorithmes.* – PhD thesis, Université de Nantes, Ecole Doctorale, 1996. N^o : 210.

[Flynn66] Flynn (M.J.). – Very high speed computing systems. *Proceedings IEEE*, vol. 54 (12), 1966, pp. 1901–1909.

[Frawley et al.91] Frawley (W.J.), Piatetsky-Shapiro (G.) et Matheus (C.J.). – *Knowledge Discovery in Databases*, chap. Knowledge Discovery in Databases : An Overview. – AAAI Press, 1991.

[Freund et al.96] Freund (Y.) et Schapire (R.E.). – Experiments with a new boosting algorithm. *In : Proceedings of the thirteenth International Conference on Machine Learning ICML'96, Bari (Italy), July 3-6th*, pp. 148–156.

[Geist et al.94] Geist (A.), Beguelin (A.), Dongarra (J.), Jiang (W.), Manchek (R.) et Sunderam (V.). – *PVM : Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing.* – The MIT Press, 1994.

[Gengler et al.96] Gengler (M.), Ubéda (S.) et Desprez (F.). – *Initiation au Parallélisme. Concepts, Architectures et Algorithmes.* – Masson, 1996.

[Giordana et al.92] Giordana (A.) et Sale (C.). – Genetic algorithms for learning relations. *In : ICML'92*, pp. 169–178.

[Giordana et al.97] Giordana (A.), Neri (F.), Saitta (L.) et Botta (M.). – Integrating multiple learning strategies in first order logics. *Machine Learning*, vol. 27 (3), 1997, pp. 209–240.

[Glover et al.93] Glover (F.), Taillard (E.) et de Werra (D.). – A user's guide to tabu search. *Annals of Operations Research*, vol. 41, 1993, pp. 3–28.

[Goldberg et al.87] Goldberg (D.E.) et Richardson (J.). – Genetic algorithms with sharing for multimodal function optimization. *In : Genetic Algorithms and their Applications : Proceedings of the second International Conference ICGA'87, J.J. Grefenstette (Ed), Lawrence Erlbaum Associates, Hillsdale, NJ*, pp. 41–49.

[Goldberg87] Goldberg (D.E.). – Simple genetic algorithms and the minimal, deceptive problem. *In : Genetig Algorithms and Simulated Annealing*, éd. par Davis (L.). pp. 74–88. – Morgan Kaufmann.

[Goldberg89] Goldberg (D.E.). – *Genetic Algorithms in Search, Optimization and Machine Learning.* – Addison-Wesley, 1989.

[Gottlob et al.93] Gottlob (G.) et Fermüller (C.G.). – Removing redundancy from a clause. *Artificial Intelligence*, vol. 61, 1993, pp. 263–289.

[Gottlob87] Gottlob (G.). – Subsumption and implication. *Information processing letters*, vol. 24(2), 1987, pp. 109–111.

[GR et al.92] Germain-Renaud (C.) et Sansonnet (J.P.). – *Les ordinateurs Massivement Parallèles.* – Armand Collin, Paris, 1992.

[Greene et al.92] Greene (D.P.) et Smith (S.F.). – COGIN : Symbolic induction with genetic algorithms. *In : Proceedings of the tenth national conference on artificial intelligence AAAI'92*, pp. 111–116.

[Greene et al.93] Greene (D.P.) et Smith (S.F.). – Competition-based induction of decision models from examples. *Machine Learning*, vol. 13, 1993, pp. 229–257.

[Grefenstette81] Grefenstette (J.J.). – *Parallel Adaptive Algorithms for Function Optimization.* – Technical Report CS-81-19, Nashville, TN, Vanderbilt University, Computer Science Department, 1981.

[Grefenstette95] Grefenstette (J.). – *Virtual Genetic Algorithms : First Results.* – Technical Report AIC-95-013, NCARAI, 1995. Internal Report.

[Gustafson88] Gustafson (J.L.). – Reevaluating Amdahl's Law. *Communications of the ACM*, vol. 31 (5), 1988, pp. 532–533.

[Haussler88] Haussler (D.). – Quantifying inductive bias : AI learning algorithms and valiant's learning framework. *Artificial Intelligence*, vol. 36 (2), 1988, pp. 177–221.

[Haynes et al.96] Haynes (T.D.), Schoenefeld (D.A.) et Wainwright (R.L.). – Advances in Genetic Programming 2, K.E. Kinnear Jr and P.J. Angeline (Eds), 1996. Chapter 18 : Type Inheritance in Strongly Typed Genetic Programming.

[Haynes96] Haynes (T.). – Duplication of coding segments in genetic programming. *In : Proceedings of the thirteenth national conference on artificial intelligence AAAI'96, AAAI Press/The MIT Press*, pp. 344–349.

[Hekanaho97] Hekanaho (J.). – GA-based rule enhancement in concept-learning. *In : Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, KDD'97*, éd. par Heckerman (D.), Mannila (H.) et Uthurusamy (R.), pp. 183–186.

[Hekanaho98] Hekanaho (J.). – DOGMA : A GA-based relational learner. *In : Inductive Logic Programming, Eighth International Conference ILP'98, D. Page (Ed), Madison, Wisconsin, USA, LNAI 1446.*

[Holland75] Holland (J.H.). – *Adaptation in Natural and Artificial Systems.* – Ann Arbor, University of Michigan Press, 1975.

[Holland86] Holland (J.H.). – *Machine Learning - An Artificial Intelligence Approach, Volume II*, chap. Escaping Brittleness : the Possibilities

of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems, pp. 593–623. – Morgan Kaufmann, 1986.

[Holmes et al.00] Holmes (J.H.), Lanzi (P.L.), Stolzmann (W.) et Wilson (S.W.). – *Learning Classifier Systems : New Models, Successful Applications.* – Technical Report 2000-44, Politecnico di Milano, 2000.

[Horn et al.94] Horn (J.), Goldberg (D.E.) et Deb (K.). – Long path problems. *In : Proceedings of the third Conference on Parallel Problems Solving from Nature PPSN'94, Y. Davidor, H.-P. Schwefel et R. Manner (Eds), Springer Verlag, LNCS 866*, pp. 149–158.

[Janikow93] Janikow (C.Z.). – A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, vol. 13, 1993, pp. 189–228.

[Jaynes96] Jaynes (E.T.). – *Probability Theory : The Logic of Science.* – 1996. Fragmentary Edition of March 1996.

[Jones95] Jones (T.). – Crossover, macromutation, and population-based search. *In : Proceedings of the sixth International Conference on Genetic Algorithms ICGA'95, L.J. Eshelman (Ed), Morgan Kaufmann*, pp. 73–80.

[Kaufman et al.99] Kaufman (K.A.) et Michalski (R.S.). – *Learning in an inconsistent world : rule selection in AQ18.* – Technical Report MLI 99-2, P99-2, George Mason University, Fairfax, VA 22030-4444, 1999. Machine Learning and Inference Laboratory.

[Kietz et al.94] Kietz (J-U.) et Lübbe (L.). – An efficient subsumption algorithm for inductive logic programming. *In : Proceedings of the Eleventh International Conference on Machine Learning ML'94, W.W. Cohen et H. Hirsh (Eds), 1994*, pp. 130–138.

[Kim et al.92] Kim (B.M.) et Cho (J.W.). – A new subsumption method in the connection graph proof procedure. *Theoretical Computer Science*, vol. 103, 1992, pp. 283–309.

[Kodratoff et al.91] Kodratoff (Y.) et Diday (E.). – *Induction symbolique et numérique.* – Cépaduès-Édtions, 1991.

[Kodratoff94] Kodratoff (Y.). – Guest Editor's Introduction (The Comprehensibility Manifesto). *AI Communications*, vol. 7 (2), 1994, pp. 83–85.

[Kodratoff95] Kodratoff (Y.). – *Algorithmic Learning Theory*, chap. Technical and Scientific Issues of KDD (or : Is KDD a Science ?). – Springer-Verlag, 1995, *Lecture Notes in Artificial Intelligence*, volume 997.

[Kodratoff98] Kodratoff (Y.). – *IBM Data-Mining Course (Liège).* – 1998.

[Kodratoff00] Kodratoff (Y.). – chap. Applications de l'apprentissage automatique et de la fouille de données. – 2000. à paraître.

[Kohavi95] Kohavi (R.). – A study of cross-validation and bootstrap for accuracy estimation and model selection. *In : Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), Montréal, Québec, Canada*, pp. 1137–1143.

[Koza89] Koza (J.R.). – Hierarchical genetic algorithms operating on populations of computer programs. *In : Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI'89, N.S. Sridharan (Ed), Morgan Kaufmann*, pp. 768–774.

[Koza92] Koza (J.R.). – *Genetic Programming.* – MIT Press, 1992.

[Koza94] Koza (J.R.). – *Genetic Programming 2.* – MIT Press, 1994.

[Lavrac et al.99] Lavrač (N.), Džeroski (S.) et Numao (M.). – Inductive logic programming for relational knowledge discovery. *New Generation Computing*, vol. 17 (1), 1999, pp. 3–23.

[Lincoff81] Lincoff (G.H.). – *The Audubon Society Field Guide to North American Mushrooms.* – New York : A.A. Knopf, 1981.

[Lo bello98] Lo Bello (G.). – *A Coevolutionary Distributed Approach to Learning Classification Programs (preliminary version).* – PhD thesis, Dipartimento di Informatica, Università di Torino, 1998.

[Maloberti00] Maloberti (J.). – *Apports de la Satisfaction de Contraintes à la Programmation Logique Inductive : un nouvel algorithme de θ-subsomption.* – Technical report, LRI, 5 septembre 2000. Rapport de Stage de DEA.

[Manderick et al.89] Manderick (B.) et Spiessens (P.). – Fine-grained parallel genetic algorithms. *In : Proceedings of the third International Conference on Genetic Algorithms ICGA'89, J.D. Schaffer (Ed), Morgan Kaufmann*, pp. 428–433.

[Merz99] Merz (C.J.). – Using correspondance analysis to combine classifiers. *Machine Learning*, vol. 36, 1999, pp. 33–58.

[Michalski80] Michalski (R.). – Pattern recognition as a rule-guided inductive inference. *IEEE transactions on Pattern Analysis and Machine Intelligence, PAMI-2*, 1980, pp. 349–361.

[Michalski84] Michalski (R.S.). – *Machine Learning : An AI Approach, Volume 1*, chap. A theory and methodology of inductive learning, pp. 83–129. – Springer Verlag, 1984.

[Mitchell et al.92] Mitchell (M.), Forrest (S.) et Holland (J.H.). – The royal road for genetic algorithms : Fitness landscapes and GA performance. *In : Toward a Practice of Autonomous Systems : Proceedings of the First European Conference on Artificial Life*, éd. par Varela (F.J.) et Bourgine (P.). – MIT Press.

[Mitchell77] Mitchell (T.M.). – Version spaces : A candidate elimination approach to rule learning. *In : Fifth International Joint Conference on Artificial Intelligence IJCAI'77*, pp. 305–310.

[Mitchell80] Mitchell (T.M.). – *The need for biases in learning generalizations.* – Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, May 1980. Édité également dans Readings in

Machine Learning, J.W. Shavlik et T.G. Dietterich (Eds), Morgan Kaufmann, 1990.

[Mitchell82] Mitchell (T.M.). – Generalization as search. *Artificial Intelligence*, vol. 18 (2), 1982, pp. 203–226.

[Mitchell97] Mitchell (T.M.). – *Machine Learning.* – McGraw-Hill, 1997.

[Montana95] Montana (D.J.). – Strongly typed genetic programming. *Evolutionary Computation*, vol. 3 (2), 1995, pp. 199–230.

[Muggleton et al.88] Muggleton (S.) et Buntine (W.). – Machine invention of first order predicates by inverting resolution. *In : Proceedings of the Fifth international Machine Learning Workshop, Morgan Kaufmann*, pp. 339–352.

[Muggleton et al.90] Muggleton (S.) et Feng (C.). – Efficient induction of logic programs. *In : Proceedings of the first conference on Algorithmic Learning Theory ALT'90, Tokyo, Ohmsha.*

[Muggleton et al.92] Muggleton (S.) et Buntine (W.). – Machine invention of first-order predicates by inverting resolution. *In : Inductive Logic Programming*, éd. par Muggleton (S.), pp. 261–280. – Associated Press, 1992.

[Muggleton et al.94] Muggleton (S.) et De Raedt (L.). – Inductive logic programming : Theory and methods. *Journal of Logic Programming*, vol. 19, 1994, pp. 629–679.

[Muggleton90] Muggleton (S.). – Inductive logic programming. *New Generation Computing*, vol. 8 (4), 1990, pp. 295–318.

[Muggleton92] Muggleton (S.). – *Inductive Logic Programming.* – Academic Press, 1992. The A.P.I.C. Series, No 38.

[Muggleton95] Muggleton (S.). – Inverse entailment and Progol. *New Generation Computing*, vol. 13 (3-4), 1995, pp. 245–286.

[Munteanu96] Munteanu (P.). – *Extraction de Connaissances dans les Bases de Données Parole : Apport de l'Apprentissage Symbolique.* – PhD thesis, Institut National Polytechnique de Grenoble, 1996.

[NC et al.97] Nienhuys-Cheng (S.-H.) et de Wolf (R.). – *Foundations of Inductive Logic Programming.* – Springer, LNAI 1228, 1997.

[Nedellec et al.96] Nédellec (C.), Rouveirol (C.), Adé (H.), Bergadano (F.) et Tausend (B.). – *Advances in Inductive Logic Programming*, chap. Declarative Bias in Inductive Logic Programming, pp. 82–103. – IOS Press, 1996.

[Neri97] Neri (F.). – *First Order Logic Concept Learning by Means of a Distributed Genetic Algorithm.* – PhD thesis, Dipartimento di Informatica, Università di Torino, 1997.

[Pei et al.97] Pei (M.), Goodman (E.D.) et Punch (W.F.). – Pattern discovery from data using genetic algorithms. *In : First Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD'97.*

[Periaux et al.95] Périaux (J.), Sefrioui (M.), Stoufflet (B.), Mantel (B.) et Laporte (E.). – *Genetic Algorithms in Engeneering and Computer Science*, chap. Robust Genetic Algorithms for Optimization Problems in Aerodynamic Design, pp. 371–396. – Wiley, 1995.

[Pettey et al.89] Pettey (C.C.) et Leuze (M.R.). – A theoretical investigation of a parallel genetic algorithm. *In : Proceedings of the third International Conference on Genetic Algorithms ICGA'89, J.D. Schaffer (Ed), Morgan Kaufmann*, pp. 398–405.

[Plotkin70] Plotkin (G.D.). – *Machine Intelligence, Volume 5*, chap. A Note on Inductive Generalization, pp. 153–163. – Edinburgh University Press, 1970.

[Provost et al.96] Provost (F.J.) et Aronis (J.M.). – Scaling up inductive machine learning with massive parallelism. *Machine Learning*, vol. 23, 1996, pp. 33–46.

[Provost et al.97] Provost (F.) et Fawcett (T.). – Analysis and visualization of classifier performance : Comparison under imprecise class and cost distributions. *In : Proceedings of the Third International Conference on Knowledge Discovery and Data Mining KDD'97.*

[Quinlan et al.93] Quinlan (J.R.) et Cameron-Jones (R.M.). – FOIL : A midterm report. *In : European Conference on Machine Learning*, pp. 3–20.

[Quinlan et al.95] Quinlan (J.R.) et Cameron-Jones (R.M.). – Induction of logic programs : FOIL and related systems. *New Generation Computing*, vol. 13 (3-4), 1995, pp. 287–312.

[Quinlan86] Quinlan (J.R.). – Induction of decision trees. *Machine Learning*, vol. 1 (1), 1986, pp. 81–106.

[Quinlan87] Quinlan (J.R.). – Generating production rules from decision trees. *In : Proceedings of the Tenth International Joint Conference on Artificial Intelligence IJCAI'87*, pp. 304–307. – Volume 1.

[Quinlan93] Quinlan (J.R.). – *C4.5 - Programs For Machine Learning.* – Morgan Kaufmann, 1993.

[Radcliffe91] Radcliffe (N.J.). – Forma analysis and random respectful recombination. *In : Proceedings of the fourth International Conference on Genetic Algorithms ICGA'91, R.K. Belew et L.B. Booker (Eds), Morgan Kaufmann*, pp. 222–229.

[Radcliffe92] Radcliffe (N.J.). – Genetic set recombination. *In : Foundations of Genetic Algorithms 2 FOGA'92, L.D. Whitley (Ed), Morgan Kaufmann*, pp. 203–219.

[Rakotomalala97] Rakotomalala (R.). – *Graphes d'Induction.* – PhD thesis, Université Claude-Bernard Lyon I, 1997.

[Ravise et al.96] Ravisé (C.) et Sebag (M.). – An advanced evolution should not repeat its past errors. *In : Proceedings of the thirteenth International*

Conference on Machine Learning ICML'96, Bari (Italy), July 3-6th, pp. 400–408.

[Raymer et al.96] Raymer (M.L.), Punch (W.F.), Goodman (E.D.) et Kuhn (L.A.). – Genetic programming for improved data mining – application to the biochemistry of protein interactions. *In : Genetic Programming 1996 : Proceedings of the first annual conference GP'96, J.R. Koza, D.E. Goldberg, D.B. Fogel et R.L. Riolo (Eds), MIT Press*, pp. 375–380.

[Rechenberg73] Rechenberg (I.). – *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* – Frommann-Holzboog, Stuttgart, 1973.

[Reiser et al.99] Reiser (P.G.K) et Riddle (P.J.). – Evolution of logic programs : Part-of-speech tagging. *In : Proceedings of the Congress on Evolutionary Computation CEC'99, Washington DC*, pp. 1338–1345.

[Richards et al.92] Richards (B.L.) et Mooney (R.J.). – Learning relations by pathfinding. *In : Proceedings of the tenth national conference on artificial intelligence AAAI'92*, pp. 51–55.

[Robinson65] Robinson (J.). – A machine-oriented logic based on the resolution principle. *Journal of the ACM*, vol. 12(1), 1965, pp. 23–41.

[Rouveirol91] Rouveirol (C.). – *ITOU : Induction de Théories en Ordre Un. Extensions de l'inversion de la résolution appliquée à la complétion de théories.* – PhD thesis, Université Paris-Sud - Centre d'Orsay, 1991. n^o d'ordre : 1536.

[Rouveirol94] Rouveirol (C.). – Flattening and Saturation : Two Representation Changes for Generalization. *Machine Learning*, vol. 14, 1994, pp. 219–232.

[Rudolph97] Rudolph (G.). – *Convergence Properties of Evolutionary Algorithms.* – Hamburg : Kovac, 1997.

[Sarma et al.97] Sarma (J.) et De Jong (K.A.). – An analysis of local selection algorithms in a spatially structured evolutionary algorithm. *In : Proceedings of the seventh International Conference on Genetic Algorithms ICGA'97*, pp. 181–186.

[Schaffer et al.91] Schaffer (J.D.), Eshelman (L.J.) et Offut (D.). – Spurious correlations and premature convergence in genetic algorithms. *In : Foundations of Genetic Algorithms 1 FOGA'90, G.J.E. Rawlins (Ed), Morgan Kaufmann*, pp. 102–112.

[Schaffer94] Schaffer (C.). – A conservation law for generalization performance. *In : Proceedings of the Eleventh International Conference on Machine Learning ML'94, W.W. Cohen et H. Hirsh (Eds), 1994*, pp. 259–265.

[Scheffer et al.96] Scheffer (T.), Herbrich (R.) et Wysotzki (F.). – Efficient algorithms for θ-subsumption. *In : Inductive Logic Programming, 6th International Workshop, LNAI1314*, pp. 212–228.

[Schoenauer et al.97] Schoenauer (M.) et Michalewicz (Z.). – Evolutionary computation. *Control and Cybernetics*, vol. 26 (3), 1997, pp. 307–338.

[Sebag et al.97a] Sebag (M.) et Rouveirol (C.). – Tractable induction and classification in first order logic via stochastic matching. *In : Fifteenth International Joint Conference on Artificial Intelligence IJCAI'97, M.E. Pollack (Ed), Nagoya, Japan, Morgan Kaufmann*, pp. 888–893.

[Sebag et al.97b] Sebag (M.), Schoenauer (M.) et Ravisé (C.). – Inductive learning of mutation step-size in evolutionary parameter optimization. *In : 6^{th} Annual Conference on Evolutionary Programming EP'97, Indiana (USA), LNCS 1213*, pp. 247–261.

[Sebag94] Sebag (M.). – Using constraints to building version spaces. *In : ECML'94, European Conference on Machine Learning, Catania, Italy, F. Bergadano et L. De Raedt (Eds), 1994*, pp. 257–271.

[Sebag96] Sebag (M.). – Delaying the choice of bias : A disjunctive version space approach. *In : Proceedings of the thirteenth International Conference on Machine Learning ICML'96, Bari (Italy), July 3-6th*, pp. 444–452.

[Sebag98] Sebag (M.). – A stochastic simple similarity. *In : Inductive Logic Programming, Eighth International Conference ILP'98, D. Page (Ed), Madison, Wisconsin, USA, LNAI 1446*, pp. 95–105.

[Silverstein et al.91] Silverstein (G.) et Pazzani (M.J.). – Relational clichés : Constraining constructive induction during relational learning. *In : Proceedings of the Eight International Workshop on Machine Learning ML'91*, pp. 203–207.

[Smith83] Smith (S.F.). – Flexible learning of problem solving heuristics through adaptive search. *In : Proceedings of the Eighth International Joint Conference on Artificial Intelligence IJCAI'83, A. Bundy (Ed), distributed by William Kaufmann, Inc.*, pp. 422–425.

[Sommer95] Sommer (E.). – An approach to quantifying the quality of induced theories. *In : Proceedings of IJCAI'95 Workshop on Machine Learning and Comprehensibility*, éd. par Nedellec (C.).

[Spears et al.93] Spears (W.M.), De Jong (K.A.), Bäck (T.), Fogel (D.B.) et de Garis (H.). – An overview of evolutionary computation. *In : Sixth European Conference on Machine Learning ECML'93, Vienna Austria, April 5-7th, P.B. Brazdil (Ed), LNAI 667*, pp. 443–459.

[Spears91] Spears (W.). – Adapting crossover in a genetic algorithm. *In : Proceedings of the fourth International Conference on Genetic Algorithms ICGA'91, R.K. Belew et L.B. Booker (Eds), Morgan Kaufmann*, pp. 166–173.

[Spiessens et al.91] Spiessens (P.) et Manderick (B.). – A massively parallel genetic algorithm. *In : Proceedings of the fourth International Conference*

on Genetic Algorithms ICGA'91, R.K. Belew et L.B. Booker (Eds), Morgan Kaufmann, pp. 279–286.

[Srinivasan et al.94] Srinivasan (A.), Muggleton (S.H.), King (R.D.) et Sternberg (M.J.E). – Mutagenesis : ILP experiments in a non-determinate biological domain. *In : Proceedings of the Fourth International Workshop on Inductive Logic Programming ILP'94, Stefan Wrobel (Ed), GMD-Studien Nr. 237*, pp. 217–232.

[SS88] Schmidt-Schauss (M.). – Implication of clauses is undecidable. *Theoretical Computer Science*, vol. 59, 1988, pp. 287–296.

[Syswerda89] Syswerda (G.). – Uniform crossover in genetic algorithms. *In : Proceedings of the third International Conference on Genetic Algorithms ICGA'89, J.D. Schaffer (Ed), Morgan Kaufmann*, pp. 2–9.

[TN et al.00] Tamaddoni-Nezhad (A.) et Muggleton (S.). – Searching the subsumption lattice by a genetic algorithm. *In : Inductive Logic Programming, Tenth International Conference ILP'2000, London, UK, J. Cussens et A. Frisch (Eds), LNAI 1866*, pp. 243–252.

[Torre99] Torre (F.). – Les vraizamis. *In : Actes de la Conférence d'Apprentissage CAP'99, Plate-forme AFIA'99, Palaiseau, Ecole Polytechnique, M. Sebag (Ed), 1999*, pp. 177–184.

[Tsang93] Tsang (E.). – *Foundations of Constraint Satisfaction.* – Academic Press, 1993.

[Utgoff86] Utgoff (P.E.). – *Machine Learning - An Artificial Intelligence Approach, Volume II*, chap. Shift of bias for inductive concept learning, pp. 107–148. – Morgan Kaufmann, 1986.

[Vapnik98] Vapnik (V.). – *Advances in Kernel Methods - Support Vector Learning*, chap. Three Remarks on the Support Vector Method of Function Estimation, pp. 25–41. – The MIT Press, 1998.

[Venturini93] Venturini (G.). – SIA : a supervised inductive algorithm with genetic search for learning attributes based concepts. *In : Sixth European Conference on Machine Learning ECML'93, Vienna Austria, April 5-7th, P.B. Brazdil (Ed), LNAI 667*, pp. 280–296.

[Venturini94a] Venturini (G.). – Analyzing french justice with a genetic-based inductive algorithm. *Applied Artificial Intelligence*, vol. 8 (4), 1994, pp. 565–577.

[Venturini94b] Venturini (G.). – *Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique.* – PhD thesis, Université de Paris-Sud - Centre d'Orsay, 1994. n^o d'ordre : 3050.

[Venturini95] Venturini (G.). – Towards a genetic theory of easy and hard functions. *In : Artificial Evolution - European Conference AE'95, J.M. Alliot, E. Lutton, E. Ronald, M. Schoenauer et D. Snyers (Eds), Springer, LNCS 1063*, pp. 54–66.

[Venturini96] Venturini (G.). – Algorithmes génétiques et apprentissage. *Revue d'Intelligence Artificielle*, vol. 10 (2-3), 1996, pp. 345–387.

[Vose92] Vose (M.D.). – Modeling simple genetic algorithm. *In : Foundations of Genetic Algorithms 2 FOGA'92, L.D. Whitley (Ed), Morgan Kaufmann*, pp. 63–73.

[Whitley91] Whitley (L.D.). – Fundamental principles of deception in genetic search. *In : Foundations of Genetic Algorithms 1 FOGA'90, G.J.E. Rawlins (Ed), Morgan Kaufmann*, pp. 221–241.

[Wilson85] Wilson (S.W.). – Knowledge growth in an artificial animal. *In : Proceedings of the first International Conference on Genetic Algorithms and their applications ICGA'85, J.J. Grefenstette (Ed), Lawrence Erlbaum Associates Publishers*, pp. 16–23.

[Wilson87] Wilson (S.W.). – Classifier systems and the animat problem. *Machine Learning*, vol. 2 (3), 1987, pp. 199–228.

[Wilson91] Wilson (S.W.). – GA-easy does not imply steepest-ascent optimizable. *In : Proceedings of the fourth International Conference on Genetic Algorithms ICGA'91, R.K. Belew et L.B. Booker (Eds), Morgan Kaufmann*, pp. 85–89.

[Wilson98] Wilson (S.W.). – Generalization in the XCS classifier system. *In : Genetic Programming 1998 : Proceedings of the Third Annual Conference*, éd. par et al. (J. Koza). – Morgan Kaufmann.

[Wilson99] Wilson (S.W.). – *State of XCS Classifier System Research.* – Technical Report 99.1.1, Prediction Dynamics, 1999. Concord, MA 01742 USA.

[Winston75] Winston (P.H.). – *The Psychology of Computer Vision*, chap. Learning Structural Descriptions from Examples, pp. 157–209. – McGraw-Hill, 1975.

[Winston92] Winston (P.H.). – *Artificial Intelligence (third edition).* – Addison-Wesley, 1992.

[Wolpert et al.96a] Wolpert (D.H.) et Macready (W.G.). – *No Free Lunch Theorems for Optimization.* – Technical report, IBM Almaden Research Center / The Santa Fe Institute, December 31, 1996.

[Wolpert et al.96b] Wolpert (D.H.) et Macready (W.G.). – *No Free Lunch Theorems for Search.* – Technical Report SFI-TR-95-02-010, The Santa Fe Institute, February 23, 1996.

[Wong et al.95] Wong (M.L.) et Leung (K.S.). – The genetic logic programming system. *IEEE Expert*, vol. 10 (5), 1995, pp. 68–76.

[Zheng et al.00] Zheng (Z.) et Webb (G.I.). – Lazy learning of bayesian rules. *Machine Learning*, vol. 41 (1), 2000, pp. 53–84.

[Zighed et al.00] Zighed (D.A.) et Rakotomalala (R.R.). – *Graphes d'Induction - Apprentissage en Data Mining.* – Hermes, 2000.

[Zitzler99] Zitzler (E.). – *Evolutionary Algorithms for Multiobjective Optimization : Methods and Applications.* – PhD thesis, Institut für Technische Informatik und Kommunikationsnetze, 1999. TIK NR. 30, Diss. ETH No. 13398.

[Zucker et al.96] Zucker (J.-D.) et Ganascia (J.-G.). – Representation changes for efficient learning in structural domains. *In : Proceedings of the thirteenth International Conference on Machine Learning ICML'96, Bari (Italy), July 3-6th*, pp. 543–551.

[Zucker et al.98] Zucker (J.-D.) et Ganascia (J.-G.). – Learning structurally indeterminate clauses. *In : Inductive Logic Programming, Eighth International Conference ILP'98, D. Page (Ed), Madison, Wisconsin, USA, LNAI 1446*, pp. 235–244.

Liste des figures

Liste des tableaux

Table des matières détaillée