Thèse de Doctorat de l'Université Pierre et Marie Curie
spécialité : Informatique

présenté par

Stephan GROLIMUND

pour obtenir le titre de Docteur de l'Université Paris 6

sujet de la thèse :

Apprentissage de Connaissances de Contrôle pour l'Optimisation Combinatoire :

Intégration du Raisonnement à Partir de Cas dans la Méthode Tabou

soutenue le 20 février 1997 devant le jury composé de :

A. DAVID	
J.G. GANASCIA	Directeur de la thèse
A. NAPOLI	
G. PLATEAU	Rapporteur
J.C. POMEROL	Président
J.L. WYBO	Rapporteur

Remerciements

Je tiens à remercier M. Jean-Gabriel Ganascia, Professeur à l'Université Pierre et Marie Curie, pour avoir accepté de diriger cette thèse et s'être tout au long de celle-ci employé à me faire bénéficier de son aide précieuse.

Toute ma reconnaissance va à M. Gérard Plateau, Professeur à l'Université de Paris Nord, et à M. Jean-Luc Wybo, Maître de Recherches à l'Ecole des Mines de Paris, pour avoir bien voulu en être les rapporteurs.

Je remercie M. Jean-Charles Pomerol, Professeur à l'Université Pierre et Marie Curie et Directeur du Laboratoire Reconnaissances de Formes et Intelligence Artificielle, où ce travail a été effectué, pour m'avoir accueilli avec bienveillance dans son laboratoire, et pour avoir accepté de présider mon jury de thèse.

M. Alain David, Assistant à l'Université Pierre et Marie Curie, trouvera ici le témoignage de ma vive gratitude pour les conseils qu'il m'a prodigué et pour m'avoir fait partager certaines de ses connaissances en méthode tabou. Je le remercie par ailleurs d'avoir accepté l'invitation à faire partie de mon jury de thèse.

Je tiens à remercier M. Amedeo Napoli, Chargé de recherches à l'INRIA Lorraine, pour les enrichissantes discussions que nous avons pu avoir, et pour sa participation à mon jury de thèse.

En outre, je remercie tous les membres de l'équipe de recherche ACASA ainsi que les membres du LAFORIA pour leur accueil et leur aide. De même, j'exprime ma gratitude à Marie-Pierre Anatole pour la relecture attentive de ce mémoire et lui demande d'excuser les fautes que je persiste à conserver.

En dernier lieu, que le Ministère de la Recherche français, le Fonds National Suisse, et là plus particulièrement M. Guggenbühl, Professeur à l'Ecole Polytechnique Fédérale de Zürich, trouvent ici le témoignage de ma reconnaissance pour le soutient matériel qu'ils m'ont apporté et sans lequel cette thèse n'aurait pu être soutenue.

Table des matières

Table des figures

Table des tableaux

Introduction

La gestion appropriée des ressources dont on dispose est d'une importance considérable tant au niveau professionnel, pour des motifs économiques, que sur le plan personnel, pour des raisons de confort. Ainsi, le choix judicieux de l'itinéraire lors d'un départ en vacances peut concourir de manière non négligeable à l'agrément, et au coût, du voyage. Malgré l'intérêt d'une bonne gestion des ressources dans la vie quotidienne, il n'en demeure pas moins que c'est essentiellement dans le secteur de l'industrie que sont évoqués les problèmes de gestion de ressources. L'impact d'une bonne gestion des moyens de production sur les résultats d'une entreprise est, en effet, loin d'être dérisoire. S'agissant d'une entreprise, ceux qui en assurent la direction tenteront sans doute d'accroître la productivité des outils de production. Alors que, pour reprendre l'exemple de notre vacancier celui-ci cherchera sans doute à écourter la durée de son trajet afin d'être au plus tôt sur son lieu de villégiature.

En dépit de leurs différences, ces deux situations présentent un point commun, dans les deux cas la problématique peut être formulée en termes de *problème d'optimisation*. Il s'agit en toute hypothèse d'identifier parmi plusieurs possibilités celle qui constituera, selon des critères préalablement établis, la meilleure solution au problème posé.

Depuis le début de la deuxième guerre mondiale, la *recherche opérationnelle* a pour vocation la formulation et la résolution de problèmes d'optimisation. Ces derniers présentent, en effet, l'inconvénient majeur d'être souvent très difficiles à résoudre, “difficile” au sens où l'entend la théorie de la complexité [Garey & Johnson 1979]. Au fil des ans, les approches ainsi développées et les applications qui en ont découlé sont devenues nombreuses. Il est, de manière schématique, possible de ranger ces méthodes dans deux grandes classes. L'une est celle des algorithmes exacts qui résolvent les problèmes d'optimisation de façon à ce que l'optimalité de la solution proposée puisse être mathématiquement démontrée. La seconde classe regroupe quant à elle des techniques dites *heuristiques*, qui sans démontrer l'optimalité des solutions avancées, n'en proposent pas moins des solutions de “bonne” qualité. Plus rapides dans l'exécution, et souvent plus aisées à mettre en oeuvre que les techniques exactes, les approches heuristiques se caractérisent par leur aptitude à s'appliquer à un plus large éventail de problèmes. Grâce à leur flexibilité il leur est possible de réaliser une modélisation des problèmes très proche de la réalité. Cette souplesse peut, dans une application réelle, largement compenser la moindre qualité des solutions obtenues, par rapport à celles qui pourraient résulter de l'emploi d'un algorithme exact, mais plus coûteux dans sa mise en oeuvre ainsi que dans son calcul.

De multiples techniques heuristiques d'optimisation ont ainsi été proposées. Parmi elles figure la *recherche locale* dont la popularité tient sans doute à la simplicité de son application. Elle présente toutefois l'inconvénient majeure de construire par trop souvent des solutions médiocres. En effet, la technique explore de manière itérative, en passant d'une solution à l'autre, un espace de solutions. En choisissant à chaque étape la solution “successeur” qui améliore le plus le critère d'optimisation, elle identifie en général assez rapidement une solution qui, pour sa part, ne possède plus de successeur valable, c'est à dire améliorant. Une telle solution constitue ainsi un *optimum local*, qui hélas peut être d'une qualité bien inférieure au minimum global que cherche la méthode.

Trois variantes de la recherche locale classique ont eu du succès récemment : le *recuit simulé* [Kirkpatrick & al. 1983], les *algorithmes génétiques* [Goldberg 1989] et la *méthode tabou* [Glover 1989, 1990]. Chacune apporte sa propre solution au problème des optima locaux. Ainsi le recuit simulé établit une analogie avec la physique thermique et ne choisit pas nécessairement la solution successeur qui améliore le plus le critère d'optimisation. Les algorithmes génétiques, pour leur part, exploitent un parallèle avec la génétique et basent la recherche sur toute une famille de solutions, dont des recombinaisons entre les membres produisent de nouvelles solutions. La méthode tabou, quant à elle, utilise le mécanisme des listes tabou, simple dans son principe, pour éviter la visite répétée des mêmes solutions. De cette manière, l'algorithme peut franchir le puits autour de l'optimum local, et ainsi continuer l'exploration du reste de l'espace des solutions.

Si nous nous sommes intéressés dans cette thèse à la méthode tabou, ce n'est cependant pas pour sa manière particulière de pallier au problème des optima locaux, mais parce qu'elle fait appel, pour ce qui est du contrôle de l'exploration de l'espace des solutions, de concepts qui vont au delà de la sélection simple du successeur paraissant être le meilleur. En effet, la méthode tabou a pour hypothèse de base qu'il est possible d'améliorer la sélection des solutions futures en analysant le passé de la même optimisation. Elle envisage, ainsi, l'identification de régions de l'espace de recherche qui sont susceptibles de contenir des solutions de bonne qualité, ou alors des régions qui n'ont pas encore été explorées. Il devient alors intéressant de guider l'optimisation vers ces régions, soit en focalisant la recherche vers les régions prometteuses, soit en quittant de force une région pour entrer dans une autre, moins parcourue encore. Le but de ce contrôle de l'optimisation est d'améliorer les résultats en profitant de *l'expérience* même de l'optimisation, laquelle est obtenue par une analyse des traces de la recherche antérieure.

En conséquence, la méthode tabou fait appel à des techniques d'apprentissage et de mémorisation des traits caractéristiques de la recherche antérieure. C'est cette notion d'*apprentissage* que nous avons plus particulièrement étudiée dans cette thèse. En effet, il s'est avéré qu'une technique d'apprentissage doit être spécialement conçu pour un problème d'optimisation pour contribuer de manière significative à la construction de solutions excellentes. La construction d'une technique d'apprentissage à la fois générique, c'est à dire qui s'applique à divers problèmes d'optimisation, et performante, dans le sens où elle améliore considérablement la qualité des solutions, est un problème important. Elle a fait l'objet de maintes recherches, mais, encore aujourd'hui, reste une question ouverte [Glover 1995].

L'objectif principal de cette thèse est donc de proposer, et d'évaluer, une technique d'apprentissage en méthode tabou qui, étant définie de manière indépendante d'un problème particulier, permet la construction de solutions de bonne qualité à travers diverses familles de problèmes d'optimisation.

Si l'on examine le processus de l'exploration de l'espace des solutions d'un peu plus prés, il apparaît que la sélection de l'état successeur constitue une étape importante, et fortement répétitive, dans la recherche. En effet, cette sélection détermine le prochain optimum local atteint, et ainsi sa qualité, et précise si la recherche focalise plutôt vers une région prometteuse, ou vers une région encore peu explorée. En conséquence, il nous parait judicieux d'étudier une technique d'apprentissage permettant de modifier directement, en tenant compte de l'expérience, le choix de l'état successeur à chaque étape de la recherche.

Pour ce faire, nous recourons à une technique d'apprentissage originaire de *l'intelligence artificielle*, une discipline qui semble de prime abord assez éloignée de la recherche opérationnelle. Cependant, les systèmes de “génération de plans de première génération” procèdent, de manière comparable à la recherche locale en optimisation, à une recherche itérative dans un espace d'états, dans le but d'identifier une solution au problème qui leur est posé [Tate & al. 1990]. Plus précisément, étant donné une description d'un état initial, un ensemble d'opérateurs et un but, le générateur de plans construit une séquence d'opérateurs, appelée *plan*, qui conduit de l'état initial à un état qui satisfait au but. Or, dans le cadre de ces systèmes, des algorithmes *d'apprentissage analytique* ont été développés avec pour objectif l'amélioration du processus de construction des solutions [Mitchell & al. 1986, Laird & al. 1986, Minton 1989]. Le propre de l'apprentissage est de détecter les inefficacités contenues dans un plan, et d'engendrer des connaissances permettant au générateur de plans de les éviter par la suite.

Au cours des dernières années, les travaux de recherche relatifs à la génération de plans ont su s'écarter de problèmes auxquels l'on reprochait d'être de pure école pour aborder des problèmes plus en prise avec la réalité [Zweben & Fox 1994]. De la même manière, des techniques d'apprentissage se sont depuis quelques temps attachées à des problèmes de taille de plus en plus importante, voir de complexité industrielle [Zerr & Ganascia 1990, Veloso 1994]. Toutefois rares sont, aujourd'hui encore, les techniques d'apprentissage analytique relatives à des algorithmes qui résolvent des problèmes d'optimisation.

Parmi les techniques développées en intelligence artificielle, une d'entre elles semble se prêter particulièrement à une utilisation à l'intérieur de la méthode tabou, c'est celle du *raisonnement à partir de cas*. De manière simple, on peut dire que le raisonnement à partir de cas résout un problème en réutilisant la solution d'un ancien problème similaire [Kolodner 1993]. Ces couples de problèmes et solutions sont appelés *cas* et sont stockés dans une mémoire de cas. Si la solution à réutiliser ne correspond pas tout à fait au problème posé, elle est adaptée. On suppose d'une part que cette adaptation est moins difficile que la construction d'une nouvelle solution, et que d'autre part, la qualité originale de la solution survit à l'adaptation.

Grâce à ce simple mécanisme de réutilisation et d'adaptation d'anciennes solutions, le raisonnement à partir de cas peut s'appliquer à des problèmes où les seules connaissances, utiles à la résolution, sont de nature opérationnelles. Elles peuvent en outre représenter une expérience particulière qui serait difficile à formaliser. Or, tel est le cas pour le problème de la sélection de l'état successeur dans la méthode tabou, puisqu'il n'existe a priori pas de théorie formelle relative à l'identification des régions de l'espace de solutions “contenant des solutions de bonne qualité”. L'expérience, de nature opérationnelle, de la recherche antérieure pourra par contre fournir des indices sur ce sujet. En effet, lors des nombreuses itérations qu'effectue la méthode tabou, au cours d'une même optimisation, elle sélectionne maintes fois un état successeur. Cependant, quelques uns de ces états successeurs vont, au cours de l'optimisation, s'avérer plus intéressants que d'autres, par exemple en menant vers des solutions d'une qualité excellente.

De ce constat empirique découle une technique de raisonnement à partir de cas simple et originale. Nous attribuons une récompense à chaque sélection d'un état successeur. Cette récompense est associée à la description de l'état où la sélection a eu lieu. Ce couple constitue un cas dans notre système. Les cas sont alors réutilisés pour améliorer, dans la suite de la même recherche, les futures sélection des états successeurs. Si par exemple une récompense négative a été associée auparavant à un état successeur très similaire à un état successeur potentiel, ce dernier doit en conséquence voir sa chance d'être sélectionné amoindrie. La technique tente ainsi d'éviter de commettre les mêmes erreurs plusieurs fois.

L'algorithme proposé est indépendant de tout problème d'optimisation pour trois raisons. D'abord, au niveau conceptuel, la sélection d'états successeurs est partie intégrale de toute méthode tabou. Ensuite, au niveau technique, les méthodes retenues pour l'attribution de la récompense et la définition des cas ne sont pas liées à un problème d'optimisation particulier, mais sont formalisées de manière générique. Enfin, au niveau opérationnel, les problèmes d'optimisation sont décrits en un langage de représentation basé sur la logique des prédicats, ce qui, peu commun en méthode tabou, permet l'application de la technique à diverses familles de problèmes d'optimisation, sans pour autant nécessiter une réimplantation de l'algorithme de raisonnement à partir de cas.

L'évaluation expérimentale de notre méthode est conduite sur les problèmes d'optimisation classiques que sont le problème de placement et d'allocation à charge fixe, le problème d'ordonnancement simple à une machine ainsi que le problème de la coloration minimale de graphes.

Le second point majeur évoqué dans cette thèse porte sur le raisonnement à partir de cas en tant que tel. L'application que nous en faisons nous amène à étudier de manière approfondie deux composants majeurs de tout système de raisonnement à partir de cas : l'organisation de la *mémoire de cas* et l'évaluation de la *similarité* entre deux cas. La mémoire de cas est responsable du “stockage” et de la mise à disposition de l'expérience que détient le système sous forme de cas. Afin d'identifier le problème “stocké” qui est le plus proche du problème que le système doit résoudre, la mémoire de cas dispose d'une technique d'évaluation de la similarité.

Ces deux notions, mémoire et similarité, sont étroitement liées, puisque pour des raisons d'implantation des systèmes de raisonnement à partir de cas, le choix de l'une conditionne le choix de l'autre [Porter 1989]. On distingue deux types de mémoires, chacune employant une autre technique d'évaluation de la similarité : la mémoire dite *des plus proches voisins* et la mémoire basée sur *l'indexation selon attributs communs*. Les mémoires du type plus proches voisins organisent les cas en une liste séquentielle [Aha & al. 1991]. Lors de la phase de récupération du cas le plus proche, le problème "actuel" est comparé à l'ensemble des problèmes se trouvant en mémoire. Si cette manière de procéder a l'avantage de laisser un libre choix quant à la technique d'évaluation de la similarité, qui peut donc être adaptée au domaine du système, elle présente néanmoins l'inconvénient majeur de s'accompagner d'une dégradation de la vitesse d'interaction (acquisition de cas et récupération) lorsque le nombre de cas en mémoire devient important.

Dans les mémoires de cas qui sont organisées par une indexation des cas selon attributs communs l'évaluation de la similarité fait partie intégrale de la représentation des cas [Schank 1982]. En effet, on suppose, dans ce type de mémoires, que deux problèmes sont similaires s'ils comportent dans leur description les mêmes attributs. Cette hypothèse justifie d'organiser les cas en mémoire par rapport aux attributs qu'ils partagent. Ainsi, lors de la récupération, il est possible d'accéder directement aux cas qui possèdent les mêmes attributs que le problème actuel, sans devoir comparer ce dernier à l'ensemble des cas en mémoire. Toutefois, la restriction que pose la représentation des problèmes sur la mesure de similarité peut être non-négligeable [Cunningham & al. 1993].

Aucune des deux organisations de la mémoire de cas ne répond aux besoins qui sont ceux de notre système. En effet, l'intégration du raisonnement à partir de cas dans la méthode tabou impose des contraintes quant à l'efficacité de l'acquisition et de la récupération des cas. Ceci est d'autant plus vrai lors que la méthode tabou nécessite un nombre important d'itération au cours d'une optimisation, et qu'en conséquence, le nombre de cas en mémoire devient important. L'organisation des cas en une liste séquentielle ne parait donc pas réellement envisageable. Par ailleurs, la représentation des problèmes d'optimisation en un langage basé sur la logique des prédicats requiert un appariement préalable des éléments des descriptions lors de l'évaluation de la mesure de similarité [Ganascia 1987].

Nous proposons donc dans cette thèse une organisation de la mémoire de cas qui est basée sur un *arbre de prototypes dynamique*. Basée sur les arbres de prototypes classiques qui sont issues d'origine de la reconnaissance de formes [Ramapriyan 1976], les arbres de prototypes dynamiques cumulent les avantages des mémoires de plus proches voisins et ceux de celles basées sur l'indexation selon attributs communs. En effet, ils permettent de recourir librement à l'évaluation de la similarité, tout en maintenant la facilité d'accès aux cas alors même que ceux-ci sont nombreux.

Cette thèse a donc pour objectif l'étude de l'intégration d'une technique d'apprentissage analytique dans la technique d'optimisation qu'est la méthode tabou. Ceci nous conduit à évoquer deux disciplines qui bien que proches par leurs méthodes n'en diffèrent pas moins de par leurs objectifs et leurs vocabulaires : la recherche opérationnelle et l'intelligence artificielle. Une première tâche, à laquelle nous consacrerons nos deux premiers chapitres, consistera à présenter les parties clés de ces deux disciplines et à les mettre en parallèle, de telle sorte que leurs points communs et leurs divergences puissent clairement apparaître. Après avoir brièvement étudié les problématiques et méthodes à la fois de l'optimisation en recherche opérationnelle et de la résolution de problèmes en intelligence artificielle, il nous sera possible de préciser les différences concernant les problèmes dont elles traitent. Leur examen nous permettra d'établir l'intérêt, ou l'inintérêt, qu'il y a à employer telle quelle l'une des techniques d'apprentissage analytique visées ci-dessus.

Dès lors se pose le problème du choix de la technique d'apprentissage analytique qui pourrait s'avérer la plus adaptée à une intégration dans la méthode tabou. En anticipant sur ce choix, et surtout sur sa justification, nous présenterons à la fin du second chapitre, le raisonnement à partir de cas. Ce dernier, grâce à sa simplicité conceptuelle, peut être appliqué à un très grand nombre de problèmes, et notamment à des problèmes d'optimisation.

Dans un troisième chapitre, nous présenterons les raisons précises pour lesquelles parmi les technique d'apprentissage nous avons retenu le raisonnement à partir de cas pour l'intégrer dans la méthode tabou. Nous décrirons aussi l'architecture de *Alois* (“*Analogical Learning for Optimisation to Improve operator Selection*”), le système qui implante notre approche.

Les chapitres quatre et cinq traitent, quant à eux, de la conception de la technique de raisonnement à partir de cas que nous employons. Nous considérerons, au quatrième chapitre, l'organisation de la mémoire de cas, et nous introduirons la mémoire de cas basée sur un arbre de prototypes dynamique qui nous permet de concilier les besoins, en termes d'efficacité et de mesures de similarité, de notre système. Le cinquième chapitre, quant à lui, sera l'occasion d'évoquer le lien étroit qu'il y a entre d'une part, l'évaluation d'une mesure de similarité portant sur des descriptions représentées en un langage basé sur la logique des prédicats, et d'autre part, le problème d'appariement des éléments de ces structures.

Le sixième et dernier chapitre de cette thèse présente la validation expérimentale de l'approche proposée. Cette validation est réalisée sur les trois problèmes, traditionnels en recherche opérationnelle, précédemment visés : celui du placement et de l'allocation d'entités, celui de l'ordonnancement simple à une machine et enfin celui de la coloration minimale de graphes. L'évaluation est effectuée soit sur des instances benchmark, soit sur des méthodes tabou confirmées, qui sont réimplantées. Les mêmes expérimentations permettent également de vérifier le comportement de la mémoire de cas que nous proposons. Mais examinerons dans un premier temps l'optimisation combinatoire et la méthode tabou.

Chapitre 1.

Optimisation combinatoire et recherche tabou

Tant en intelligence artificielle qu'en recherche opérationnelle il n'existe pas, pour de nombreux problèmes, de méthode de résolution efficace. On doit alors se résoudre à l'emploi de techniques heuristiques, rapides certes, mais présentant l'inconvénient de produire des solutions qui ne sont pas toujours de la qualité désirée. Il est en conséquence concevable, et souhaitable, de tenter d'améliorer ces heuristiques en y intégrant des méthodes qui apprennent. Ces méthodes peuvent engendrer un accroissement permanent des compétences de l'heuristique, et une amélioration continue de son efficacité.

Dans cette thèse, nous envisagerons une coopération étroite entre deux techniques, l'une originaire de la recherche opérationnelle et l'autre issue de l'intelligence artificielle. Ces deux disciplines sont très proches dans leurs objectifs comme dans leurs méthodes, plus proches que ne permet de le soupçonner un simple regard externe. Dans les deux premiers chapitres, nous présenterons ces deux disciplines dans le double but avoué d'une part, loin de toute prétention à l'exhaustivité, de placer les deux techniques que nous emploierons, la *méthode tabou* et le *raisonnement à partir de cas*, dans le contexte qui est le leur, à l'intérieur de leur domaine respectif ; d'autre part, de mettre en évidence les points communs aux deux disciplines, et leurs différences. Pour ce faire, nous nous permettrons de recourir à un formalisme qui n'est sans soute pas des plus usité, surtout en recherche opérationnelle.

Ce premier chapitre traite donc de l'optimisation combinatoire en général. Nous évoquerons d'abord la nature des problèmes étudiés par cette discipline, avant de considérer les principales techniques de résolution, en distinguant la résolution exacte de la résolution heuristique. Dans cette dernière catégorie, nous présenterons les techniques qui, basées sur la recherche locale, sont aujourd'hui les plus répandues. Il s'agit du recuit simulé, des algorithmes génétiques, et de la méthode tabou, que nous présenterons, quant à elle, dans la deuxième section de ce chapitre.

1.1. L'optimisation combinatoire

Cette section tend à être une introduction aux principaux objectifs et méthodes de *l'optimisation*. Nous proposerons donc d'abord une formulation générique des problèmes d'optimisation, avant d'évoquer différentes techniques utiles à leur résolution. Ces techniques sont de deux types. On trouve d'une part les méthodes exactes et d'autre part les méthodes heuristiques. Nous les présenterons dans deux paragraphes différents. Parmi les méthodes heuristiques, la recherche locale nous intéresse plus particulièrement, puisqu'elle est à la base de diverses techniques d'optimisation modernes, telles que par exemple le recuit simulé, les algorithmes génétiques ou encore la méthode tabou. Dans un dernier paragraphe nous examinerons des techniques d'apprentissage proposées en recherche locale. Notons que la méthode tabou, compte tenu de la place centrale qu'elle occupe dans nos travaux, fera l'objet d'une présentation distincte dans la prochaine section.

1.1.1. Formulation des problèmes d'optimisation

De manière générale, un problème d'optimisation peut être défini par un *espace du problème* $\mathcal{E}$, qui contient un *sous-espace de solutions* $\mathcal{S}$ (aussi appelé *région de faisabilité*), $\mathcal{S} \subseteq \mathcal{E}$, ainsi qu'une *fonction objectif* $f(\cdot)$, $f : \mathcal{S} \rightarrow \mathcal{R}$ [Minoux 1987]. L'espace du problème décrit en quelque sorte le monde, tandis que l'espace des solutions contient l'ensemble des *solutions* s, $s \in \mathcal{S}$, admissibles à la résolution du problème. Le rôle de la fonction objectif est de produire un critère d'évaluation permettant de comparer les solutions admissibles entre elles, et donc de déterminer, sur une échelle globale, monotone et unique, la qualité d'une solution.

Dans un problème d'optimisation ainsi défini, le but est de trouver une solution s, dorénavant appelée *solution optimale* s^*, qui *minimise*, ou *maximise*, la fonction objectif sur l'espace des solutions. Le problème peut donc s'écrire de la manière suivante :

$$(\text{OPT}) \qquad \text{minimise } f(s) \tag{1}$$

$$\text{avec:} \quad s \in \mathcal{S} \subseteq \mathcal{E} \tag{2}$$

Sans restriction de généralité, il est possible de limiter l'exposé au problème de minimisation. En effet, tout problème de maximisation peut être formulé comme un problème de minimisation avec :

$$\max f(s) = - \min \{- f(s)\} \tag{3}$$

Notons qu'en général, il n'existe pas de critère explicite et efficace permettant d'affirmer qu'une solution s quelconque soit la solution optimale [Minoux

1987]. Néanmoins, en comparant les solutions entre elles, toute solution optimale s^* doit satisfaire à la condition suivante :

$$\forall s \in \mathcal{S} \qquad f(s^*) \leq f(s) \tag{4}$$

La condition (4) autorise l'existence de plusieurs solutions optimales distinctes. Elles sont de qualité égale, puisque leurs évaluations par $f(\cdot)$ sont identiques par définition. En général, un problème d'optimisation est considéré comme étant résolu lorsque le caractère optimal *d'une* solution a été démontré. Celui-ci est démontré lorsque l'algorithme de résolution permet d'établir la preuve que (4) est satisfait. Différentes conditions nécessaires doivent être satisfaites pour que l'existence d'une solution optimale soit établie : a) $\mathcal{S}$ doit être non-vide, et b) $f(\cdot)$ doit avoir une borne inférieure sur $\mathcal{S}$. Pour les problèmes dont nous traitons dans cette thèse, nous supposons que ces deux conditions sont toujours vérifiées.

En général on distingue entre un *problème* et une *instance d'un problème*. La description d'un problème formalise ses structures génériques. Elle contient des paramètres. La description d'une instance de problème affecte des valeurs, souvent numériques, à ces paramètres.

1.1.2. La complexité des problèmes d'optimisation

Lorsque l'on aborde le sujet des problèmes d'optimisation surgit inévitablement la question de leur complexité. Rappelons brièvement qu'en théorie de la complexité, on distingue entre les *problèmes de décision* et les *problèmes d'optimisation* [Garey & Johnson 1979]. Un problème de décision peut se formuler par une question du type “Existe-t-il une solution ayant telle caractéristique ?”, et les réponses sont du type “Oui/Non”. Un problème d'optimisation, quant à lui, donne lieu à une question telle que “Quelle est la solution maximale (minimale) ?”. Une réponse du type “Oui/Non” s'avère alors insuffisante, puisque, a priori, l'existence d'une solution maximale (minimale) n'est pas mise en cause. On cherche plutôt à connaître la solution optimale elle-même.

Il est néanmoins possible de formuler un problème d'optimisation par une suite de problèmes de décision. Ainsi, on peut se demander “Existe-t-il une solution de taille de au moins (au plus) x ?”. En augmentant (diminuant) x après chaque réponse positive, on se rapproche pas à pas de la solution optimale. Toutefois, la résolution de cette suite de problèmes de décision est au moins aussi complexe que la résolution du problème d'optimisation initial.

Notons que la théorie de la complexité traite des problèmes de décision, mais *pas* des problèmes d'optimisation. Ainsi, les classes de complexités *P*, *NP* et

NP-complet se rapportent aux problèmes de décision uniquement. Cependant, il est convenu d'appeler *NP-difficile*[1] tout problème d'optimisation qui se formule par une suite de problèmes de décision qui eux sont NP-complets [Reeves 1993]. Un problème NP-difficile est en conséquence aussi difficile à résoudre qu'un problème NP-complet.

1.1.3. Les méthodes exactes

Sous la formulation générique de (1) - (2) sont englobés divers problèmes d'optimisation ayant des caractéristiques très variées. Des *algorithmes exacts* efficaces permettent de résoudre certains d'entre eux. Pour les autres, seules des techniques de résolution approximatives, ou heuristiques, permettent une résolution au niveau d'efficacité requis. Le domaine scientifique consacré aux algorithmes exacts est celui de la *programmation mathématique*. Ce domaine est vaste, tant en termes de problèmes traités que de méthodes proposées. Selon les caractéristiques de la fonction objectif et de l'espace de problème, on discerne différentes classes de problèmes d'optimisation, chacune d'elles ayant donné lieu à différentes approches spécifiques utiles à leurs résolutions [Minoux 1987]. Typiquement, les problèmes se distinguent les uns des autres en raison du caractère continu ou discret, linéaire ou non-linéaire de la fonction objectif. De même, l'espace du problème, comme l'espace des solutions, peut être continu, convexe et/ou compact.

Traiter de l'ensemble de ces différentes classes de problèmes est hors du cadre posé. Aussi, nous présenterons seulement deux classes de problèmes : la programmation linéaire, qui est le grand classique de la programmation mathématique, et l'optimisation combinatoire, classe de problèmes à laquelle s'applique la méthode tabou. Le lecteur intéressé trouvera toute précision désirée concernant les autres éléments de la programmation mathématique dans des références classiques [Minoux 1987, Beale 1988, Nemhauser & al. 1989].

a) La programmation linéaire

Historiquement, la programmation mathématique est née avec l'algorithme du Simplexe, mis au point dans les années 50 par Danzig (1963). Cet algorithme était le premier à résoudre les problèmes appartenant à la *programmation linéaire*. Un programme linéaire (PL) s'écrit de façon suivante :

---

[1] “NP-hard” en anglais.

$$\text{(PL)}\qquad \text{minimise } f(s) = \mathrm{c}s \tag{5}$$

$$\text{avec : }\quad \mathrm{A}s \leq \mathrm{b} \tag{6}$$

$$s = (s_1\ s_2\ \ldots\ s_n) \tag{7}$$

$$s_i \in \mathcal{R}^+,\ i = 1\ \ldots\ \mathrm{n} \tag{8}$$

L'espace de problème $\mathcal{E}$ d'un programme linéaire est représenté par un vecteur de *n variables* réelles s_i, $s = (s_1\ s_2\ \ldots\ s_n)$ (7). Chaque s_i a comme domaine de définition[2] $\mathcal{R}^+$, $s_i \in \mathcal{R}^+$ (8). En conséquence, on a $\mathcal{E} = \mathcal{R}^+$. La région de faisabilité $\mathcal{S}$ est alors un sous-espace de $\mathcal{R}^+$, $\mathcal{S} \subseteq \mathcal{R}^+$. Elle est décrite par des contraintes qui sont des combinaisons linéaires entre variables. Ces contraintes sont représentées par la matrice A et le vecteur membre droit b (6). Avec p contraintes décrivant le problème, la matrice A est de dimension $p \times n$ et le vecteur membre droit b est de dimension p. La fonction objectif (5) est, elle aussi, une combinaison linéaire de variables. Plus précisément, elle correspond au produit scalaire du vecteur de coûts c avec le vecteur des variables s. Une solution d'un programme linéaire est une instanciation des variables s_i de sorte qu'il soit satisfait aux contraintes (6). La solution optimale est alors une instanciation des variables s_i qui minimise la fonction objectif.

Revenons à la différence entre un problème et une instance de problème (cf. p. 19). Ici la description d'un problème linéaire définit essentiellement la structure de la matrice A. La description d'une instance de problème, elle, précise les valeurs numériques de A, b et c.

L'expression “programmation linéaire”, datant des années 50, peut paraître à première vue un peu étrange. Néanmoins, elle est tout à fait appropriée. En effet, le mot “programmation” tient compte des moyens employés pour la résolution des problèmes, c'est à dire la programmation sur des calculateurs électroniques, un outil nouveau pour l'époque ; et “linéaire” indique simplement que les contraintes et la fonction objectif s'expriment sous forme de combinaisons linéaires des variables.

La programmation linéaire connaît d'importantes applications industrielles, car l'optimisation de programmes linéaires est considérée comme facile, aujourd'hui, pour deux raisons. D'une part, il existe des variantes de l'algorithme du Simplexe qui sont remarquablement efficaces. Etant de complexité exponentielle, l'algorithme du Simplexe présente en pratique un comportement semblable à un algorithme de complexité polynomiale [Papadimitriou & Steiglitz 1982]. D'autre

---

[2] Sans perte de généralité, les variables s_i (8) peuvent être astreintes à la non-négativité [Minoux 1987], au lieu de $s_i \in \mathcal{R}$.

part, les méthodes du type points intérieurs sont effectivement de complexité polynomiale [Williams 1993b]. Un exemple d'un programme linéaire sera donné plus loin lorsque nous traiterons du calcul de la similarité entre deux cas (chapitre 5, cf. p. 156).

b) La programmation linéaire-mixte

La *programmation linéaire-mixte* est un autre champs de la programmation mathématique qui donne lieu à d'importantes applications industrielles. Elle se distingue de la programmation linéaire par le seul fait qu'un programme linéaire-mixte contient des variables discrètes. L'espace du problème $\mathcal{E}$ n'est en conséquence plus. Un programme linéaire-mixte s'écrit de la façon suivante :

(PLM)

$$\text{minimise}\, f(s) = \mathrm{c}s \tag{9}$$

$$\text{avec :} \quad \mathrm{A}s \leq \mathrm{b} \tag{10}$$

$$s = (x_1 \ldots x_j \ldots x_n \, y_1 \ldots y_j \ldots y_m) \tag{11}$$

$$x_j \in \mathcal{R}^+, j = 1 \ldots n \tag{12}$$

$$y_i \in \mathcal{Z}^+, i = 1 \ldots m \tag{13}$$

Comme pour la programmation linéaire, la fonction objectif (9) et les contraintes (10) sont des fonctions linéaires. Le vecteur s (11) contient deux parties : l'une est celle des variables réelles, les x_j (12), l'autre celle des variables discrètes, les y_i (13). Les conditions de (13) sont aussi appelées *conditions d'intégralité*. Un cas spécial de la programmation linéaire-mixte est celui de la *programmation entière*, où les problèmes ne contiennent pas de variables réelles. On a donc $n = 0$ dans (12).

Par l'introduction de variables discrètes, des problèmes plus complexes qu'en programmation linéaire peuvent être modélisés. Habituellement, ces variables désignent soit des composants homogènes (tels que le nombre de voitures produit sur une chaîne de production, la production d'une fraction d'une voiture n'ayant pas de sens), soit des expressions logiques [Williams 1993a]. Avec la seule programmation linéaire, il n'est pas possible de décrire ce type de contraintes. Au chapitre 6, nous donnerons l'exemple d'un modèle de la programmation linéaire-mixte (cf. p. 166).

L'inconvénient de la programmation linéaire-mixte se situe au niveau de l'efficacité de l'optimisation. En effet, les problèmes à variables discrètes sont loin d'être aussi faciles à résoudre que les programmes linéaires. Il se peut ainsi que des instances contenant seulement quelques dizaines de variables discrètes soient difficiles à résoudre. Ceci pour deux raisons. D'une part, la programmation linéaire-mixte se prête tout à fait à la modélisation de problèmes NP-difficiles. Pour ce type de problèmes, la croissance exponentielle des besoins en ressources pour la résolution est indépendante de l'algorithme d'optimisation employé [Garey & Johnson 1979]. D'autre part, en programmation linéaire-mixte il n'existe pas de technique qui soit à la fois générique *et* efficace. Il y a certainement des techniques générales pour l'optimisation de problèmes à variables discrètes, mais elles ne sont pas aussi efficaces que l'est l'algorithme du Simplexe pour les programmes linéaires. A titre indicatif, entrent dans cette catégorie a) en premier lieu, les techniques classiques de *séparation et évaluation* [Lawler & Wood 1966] ("*branch & bound*") qui, en relâchant les conditions d'intégralité, utilisent souvent l'algorithme du Simplexe pour déterminer des bornes inférieures au problème relâché ; b) les techniques basées sur la dualité lagrangienne [Fischer 1981] ; et c) la programmation dynamique [Bertsekas 1995].

Cela ne veut pas dire qu'il n'existe pas, pour certains problèmes de la programmation linéaire-mixte, des techniques efficaces de résolution. Seulement, ces techniques ne sont pas nécessairement généralisables à d'autres problèmes. Un cas exemplaire en sont les problèmes de la programmation linéaire-mixte avec des matrices A qui sont *unimodales* (10). Ces problèmes sont résolus directement par les techniques de la programmation linéaire. En effet, toutes les solutions produites par des algorithmes de la programmation linéaire sont automatiquement entières [Williams 1993a, 1993b]. Le problème de l'appariement bipartite est un exemple d'un problème unimodal (cf. p. 156).

c) L'optimisation combinatoire

Les problèmes *d'optimisation combinatoire* constituent un autre secteur important de la programmation mathématique. Sa définition formelle exacte ne fait cependant pas l'unanimité [Papadimitriou & Steiglitz 1982, Nemhauser & Wolsey 1988]. Nous n'adoptons ni l'une ni l'autre des définition données, mais employons ce terme afin de désigner les problèmes d'optimisation qui comportent des variables discrètes et qui sont NP-difficiles. Les problèmes de l'optimisation constituent alors un sous-ensemble des problèmes de la programmation linéaire-mixte (puisque ces derniers ne sont pas nécessairement NP-difficiles). Selon cette caractérisation, le problème de placement et d'allocation d'entités est un problème d'optimisation combinatoire (cf. p. 166). En général, la formulation des problèmes d'optimisation combinatoire sous la forme d'un programme linéaire-mixte, c'est à dire en contraintes linéaires et en variables continues et discrètes, est assez directe. Toutefois, certains problèmes ne se prêtent pas facilement à cela. Tel est le cas du problème, peut-être le plus connu en optimisation combinatoire, du voyageur de commerce [Lawler & al. 1985, Reinelt 1994].

En ce qui concerne les algorithmes exacts de résolution de ces problèmes, ce sont les mêmes que ceux employés pour la résolution des problèmes de la programmation linéaire-mixte. Ils se heurtent donc aux mêmes limites pour ce qui est de leur efficacité. Le recours à une technique d'optimisation heuristique s'impose donc souvent comme la seule alternative.

1.1.4. Les méthodes heuristiques

Une technique *d'optimisation heuristique* a pour but de construire, en un temps de calcul qui reste acceptable, des solutions de *bonne qualité*, c'est à dire des solutions qui sont proches de l'optimum sans pour autant satisfaire au critère d'optimalité (4). Ce type de technique est une alternative valable aux méthodes d'optimisation exactes pour les problèmes où celles-ci ne sont pas assez efficaces. Ces dernières années, avec l'apparition de techniques heuristiques très performantes telles que le recuit simulé, les algorithmes génétiques ou encore la méthode tabou, ces méthodes se sont fortement répandues et ont connu de nombreuses applications [Reeves 1993].

Il existe un nombre important de classes d'heuristiques. Il serait cependant hors de propos d'en faire ici une analyse ou une classification détaillée. Le lecteur peut trouver ces précisions ailleurs [Zanakakis & Evans 1989]. Nous limitons notre présentation aux heuristiques qui sont basées sur la recherche locale, dont fait aussi partie la méthode tabou.

a) La recherche locale

Les techniques d'*exploration de voisinage*, aussi appelées *recherche locale*, figurent parmi les techniques d'optimisation heuristiques les plus anciennes et les plus répandues. L'une des premières références à une telle technique concerne le problème du voyageur de commerce [Lin 1965]. Leur succès est certainement dû, au moins en partie, à leur simplicité d'emploi, qui les rend applicables à un grand nombre de problèmes aux caractéristiques très variées.

En recherche locale, l'espace de problème $\mathcal{E}$ est un espace d'états. Ainsi, l'espace de solutions $\mathcal{S}$, $\mathcal{S} \subseteq \mathcal{E}$ contient[3] l'ensemble des *états solutions* s. De plus, il existe un ensemble $\mathcal{M}$ de *transitions* m, $m : \mathcal{S} \rightarrow \mathcal{S}$, qui permettent de passer d'un état solution à un autre. Pour chaque état solution s, son *voisinage* $N(s)$

---

[3] Notons que la distinction explicite entre espace de problème et espace des solutions est peu commune en recherche locale.

(“*neighbourhood*”) est composé par l'ensemble des états solutions que l'on peut atteindre par l'ensemble des transitions applicables à s (figure 1). On a ainsi comme définition de $N(\cdot)$:

$$\forall s \in \mathcal{S}, \qquad N(s) = \{s' \mid \exists m \in \mathrm{M},\ s' = m(s)\} \tag{14}$$

Les méthodes de recherche locale explorent l'espace des solutions en appliquant de manière répétitive des transitions m, menant d'un état solution s à un état solution successeur s', $s' = m(s)$, membre de $N(s)$. Le but de cette exploration est d'identifier des solutions qui soient optimales (ou proches de l'optimum) par rapport à la fonction objectif $f(s)$. En général, l'espace de recherche ainsi défini est fortement connexe.

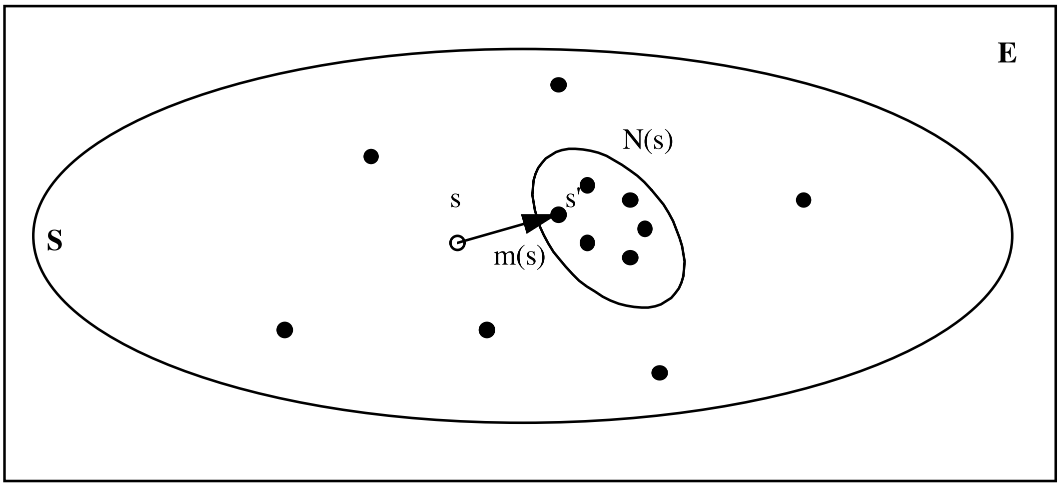


Figure 1. Voisinage $N(s)$ d'un état solution s

La sélection de l'état successeur parmi l'ensemble des états membres du voisinage se fait de manière gloutonne par rapport à l'intérêt attribué à chacun de ces états. En général, l'intérêt qu'il y a à appliquer une transition m à un état s, et ainsi à sélectionner $s' = m(s)$ comme état successeur, est évalué au travers d'une *fonction de prévision* p, $p : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{R}$. Afin d'améliorer au plus la fonction objectif à chaque itération, l'état successeur est choisi comme étant celui qui maximise $p(\cdot)$ sur $N(s)$. Traditionnellement, $p(\cdot)$ reflète l'évolution de la fonction objectif avant et après l'application de la transition $m(s)$. Il en découle dans ce cas la définition suivante pour $p(\cdot)$:

$$p(s, s') = f(s) - f(s') = f(s) - f(m(s)) \tag{15}$$

Cette exploration de l'espace de recherche n'est ni complète ni systématique. Compte tenu de la taille possible de l'espace des solutions d'une instance d'un problème NP-difficile, une exploration complète n'est pas envisageable. Elle n'enregistre pas non plus les état visités au cours de la recherche. Il est ainsi possible qu'un état solution soit visité plusieurs fois lors de la même recherche. Ce type de méthodes basées sur l'amélioration itérative locale souffre en général du problème des minima locaux (figure 2). Ainsi, au cours de l'exploration de l'espace de recherche, un état solution est identifié dont le voisinage ne comporte pas de successeurs qui améliorent la fonction objectif. Cet état est alors optimal par rapport à son voisinage, mais pas nécessairement par rapport à $\mathcal{S}$. On appelle un tel état solution *minimum local* s^{LM}. Il peut être défini ainsi :

$$\forall s' \in N(s^{LM}), f(s^{LM}) \leq f(s') \tag{16}$$

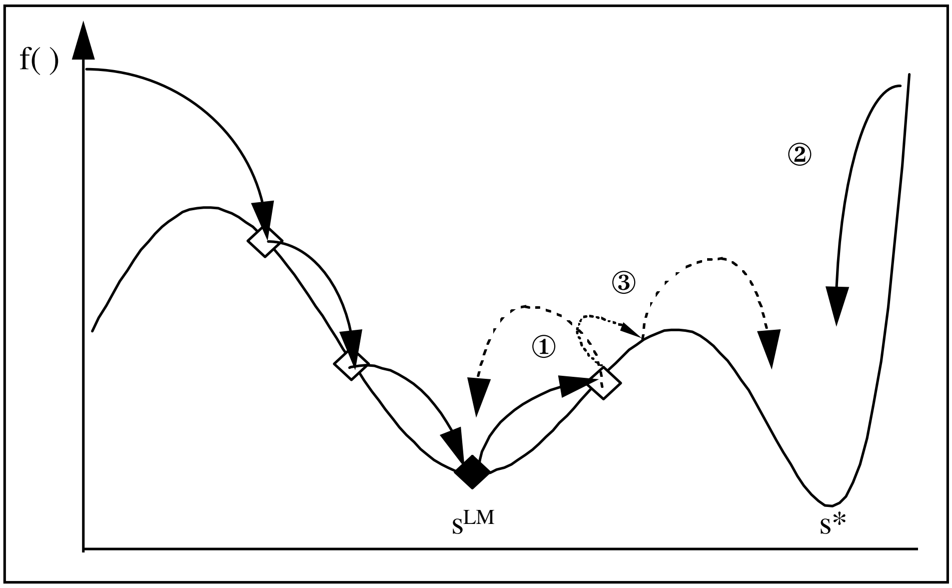


Figure 2. Minimum local s^{LM} versus solution optimale s^*

En général, il n'existe pas de moyen permettant d'identifier un minimum local comme étant une solution optimale selon le critère (4). Le phénomène des minima locaux est un inconvénient majeur des techniques de recherche locale. Souvent, les minima locaux sont, en effet, d'une qualité bien inférieure à la solution optimale. Différentes alternatives sont envisageables afin de pallier au problème des minima locaux :

a) La recherche continue à partir du minimum local (figure 2 ①). Dans ce cas, il est fort probable qu'elle ne va pas pouvoir quitter le puits d'optimalité locale. Par conséquence, elle y retournera et l'exploration bouclera. Puisque cela empêcherait l'exploration d'autres régions de l'espace de recherche, ce phénomène est indésirable.

b) La recherche continue dans un état nouvellement construit (figure 2 ②). Cet état peut alors être produit soit de manière aléatoire, soit en suivant une procédure systématique. Néanmoins, il n'est pas toujours évident d'élaborer une technique qui, de manière robuste, assure que l'exploration trouve des minima locaux différents pour des instances de problèmes différentes.

c) La recherche continue à partir du minimum local en imposant l'application de plusieurs transitions qui dégradent la fonction objectif (figure 2 ③). Le but est de sortir ainsi du puits d'optimalité locale et donc d'éviter le bouclage. Le problème est alors de veiller à ce que l'on ne s'éloigne pas trop du minimum local afin d'éviter de trop altérer la qualité du nouveau point de départ. Cette approche présente les mêmes problèmes de robustesse que la précédente.

A elle toute seule, la recherche locale réalise donc souvent des performances limitées, produisant des solutions de qualité médiocre. Cependant, elle a donné lieu à de multiples extensions et raffinements. Récemment, trois familles de techniques améliorées se sont imposées comme étant les meilleures méthodes d'optimisation heuristiques [Pirlot 1992, Reeves 1993] : le recuit simulé, les algorithmes génétiques et la méthode tabou. Chacune apporte sa propre solution au problème des minima locaux. Nous présenterons ces trois approches ci-dessous.

b) Le recuit simulé

Le *recuit simulé* [Kirkpatrick & al. 1983, Van Laarhoven & Aarts 1988] est une variante du schéma classique de recherche locale qui exploite une analogie avec le processus thermodynamique du refroidissement de matériaux. Il recherche la convergence vers la solution optimale au bout d'une longue recherche dans l'espace des solutions. La particularité de cette recherche se situe au niveau de la sélection des transitions. Ainsi, et à l'opposé de ce qui existe en recherche locale, ce n'est pas la transition qui améliore le plus la fonction objectif qui est sélectionnée. En réalité, la première transition améliorante identifiée est appliquée. Les transitions sont choisies de manière aléatoire parmi l'ensemble des transitions applicables. Par ailleurs, l'application de transitions qui dégradent la fonction objectif est autorisée. Toutefois, la sélection de transitions dégradantes est contrôlée par un paramètre appelé température. Plus la température est élevée, plus la probabilité de sélectionner une transition dégradante est forte. Cette température est réduite au fur et à mesure que la recherche progresse, et la sélection de transitions dégradantes devient alors moins probable.

C'est donc grâce à la sélection aléatoire des transitions ainsi qu'à l'acceptation de transitions dégradantes que le recuit simulé échappe aux minima locaux. En effet, il est démontré, en modélisant l'exploration de l'espace de recherche par une chaîne de Markov, que la recherche va converger en la solution optimale après, au plus tard, un nombre infini d'itérations. En pratique, il n'est heureusement pas nécessaire d'effectuer ce nombre infini d'itérations avant d'obtenir des solutions de bonne qualité. En effet, le recuit simulé a démontré son efficacité et son caractère performant sur un bon nombre de problèmes [Johnson & al. 1989, 1991].

c) Les algorithmes génétiques

Les *algorithmes génétiques* [Holland 1975, Goldberg 1989, Davis 1991] se fondent sur une analogie avec la reproduction de chromosomes. Leur mécanisme original consiste en la construction d'une nouvelle solution non pas par l'application d'une simple transition, mais par une recombinaison de parties d'anciennes solutions. L'exploration de l'espace de recherche se base ici sur un ensemble de solutions, appelé *population*, et non pas, comme c'est le cas en recherche locale, sur une seule. Il existe deux mécanismes pour assembler une nouvelle solution *fille* à partir des solutions *parents*, membre de la population. La *mutation* qui modifie de manière aléatoire une caractéristique élémentaire d'une solution parent, et, souvent plus important, le *croisement* qui assemble une nouvelle solution en scindant deux parents en deux et en concatenant un bout de chacun[4]. A chaque pas de la recherche, il y a d'abord sélection de l'opération à appliquer, puis sélection des membres de la population sur lesquels l'opération s'applique, et en dernier lieu la décision de retenir ou non les nouvelles solutions engendrées comme membres de la population, en remplaçant des solutions plus anciennes.

Les algorithmes génétiques pallient au problème des minima locaux surtout grâce à l'opération de croisement qui a la possibilité de construire des solutions fortement originales et peut ainsi mener la recherche dans des nouvelles régions de l'espace de recherche, et donc loin des minima locaux déjà rencontrés. En effet, dans le cadre des algorithmes génétiques, il peut être débattu du problème lié aux minima locaux, puisque ces techniques se basent à chaque état de la recherche sur tout un ensemble de solutions, et non pas sur une seule. Toutefois, il convient d'ajouter que l'existence d'une homogénéité aiguë de la population peut provoquer le phénomène de convergence prématurée. Dans ce cas, le croisement de deux solutions presque identiques rend les nouvelles solutions trop similaires à leurs parents.

---

[4] Le croisement est une opération qui ne produit pas de solution qui soit une modification locale d'une solution précédente. On peut, en effet, considérer que les algorithmes génétiques n'appartiennent pas à la famille de la recherche locale, mais constituent plutôt une technique d'optimisation globale.

d) La méthode tabou

La *méthode tabou* [Glover 1989, 1990] gère de manière active le problème lié aux puits d'optimalité locale. Elle emploie une technique de contrôle tactique visant à permettre à la recherche de franchir le seuil d'optimalité (figure 2 ③). A cette fin, toute transition appliquée est inscrite dans une *liste tabou*[5]. Ensuite, dans le but d'éviter le bouclage, toute transition, qui inverse une transition membre de la liste tabou, est déclarée tabou et, en conséquence, interdite d'application. Nous donnerons une présentation plus détaillée de la méthode tabou dans la prochaine section de ce chapitre (cf. p. 35).

1.1.5. Des limites opérationnelles de la recherche locale

Il convient encore de faire quelques remarques d'ordre pratique sur la mise en oeuvre de techniques basées sur la recherche locale. En effet, bien que la recherche locale soit en principe assez simple, elle peut rencontrer des problèmes de *complexité* algorithmique.

a) La recherche locale polynomiale

Les problèmes concernent d'une part la construction des états solutions, et d'autre part l'évaluation de la fonction objectif. Pour ce qui est de la construction des états solutions, notons que l'exploration de l'espace de recherche commence dans une solution initiale. Elle se poursuit en passant à travers d'autres états solutions. Ce processus de recherche peut se décomposer en 3 phases : 1) la construction de l'état initial, 2) la construction du voisinage de l'état actuel de la recherche, et 3) la sélection de l'état successeur parmi les membres du voisinage. Chacune de ces trois étapes est importante en termes d'efficacité, même si sur un plan pratique, les deux dernières le sont plus particulièrement dans la mesure où elles sont appliquées de manière répétitive. Plus elles sont rapides, plus grande est la partie de l'espace de recherche, en termes d'états visités, que l'exploration parcours en une durée donnée, et en conséquence, plus la solution s'approche de l'optimum, car plus grande est la probabilité de trouver des solutions de bonne qualité.

Si la complexité de chacune de ces trois phases est polynomiale, polynomiale par rapport à la taille de l'instance de problème, on dit que la méthode fait partie de la classe de complexité *PLS* (pour “*Polynomial Local Search*”) [Johnson

---

[5] On peut attribuer à la liste tabou un but plus général que celui d'échapper au puits d'optimalité locale, celui de pousser la recherche continuellement en avant.

& al. 1988]. Les réalisations pratiques font généralement partie de *PLS*. En ce qui concerne la phase 1, une solution triviale, et donc facile à construire, est souvent choisie comme état initial. La réalisation de la phase 2 requiert deux éléments qui doivent être de complexité polynomiale, par rapport à la taille de l'instance de problème. D'une part, il s'agit de s'assurer que le nombre de transitions, et ainsi la taille du voisinage, est borné par un polynôme. D'autre part, la détermination de l'applicabilité d'une transition, et en conséquence l'établissement de la décision si un état solution fait partie du voisinage, doit aussi être polynomial. La construction d'un voisinage étant une tâche très répétitive, divers auteurs préconisent des transitions menant à de petites complexités, souvent quadratiques, de cette phase 2 [Lin 1965, Taillard 1991]. Cependant, Lin (1965) indique que l'emploie de transitions plus complexes augmente la qualité des solutions identifiées. Il juge par contre ces transitions utiles seulement dans le cas où le temps de calcul supplémentaire est disponible.

Il se peut que même un voisinage de taille polynomiale soit trop grand. On définit alors la notion *d'ensemble candidats* $V(s)$ comme sous-ensemble du voisinage original, $V(s) \subseteq N(s)$. Souvent de taille fixe, $V(s)$ est, en général, construit en sélectionnant, de manière aléatoire, des membres de $N(s)$. La taille de $V(s)$ peut être aussi petite que $|V(s)| = 1$.

En ce qui concerne la phase 3, elle fait intervenir l'évaluation de la fonction de prévision pour chaque état membre du voisinage. Cette évaluation doit donc être de complexité polynomiale. Au cas où l'évaluation de $p(\cdot)$ fait intervenir la fonction objectif $f(\cdot)$, telle que dans (15), l'évaluation de $f(\cdot)$ doit aussi être de complexité polynomiale. Il existe des problèmes où l'évaluation de $f(\cdot)$ est complexe. Dans ce cas, il est possible de baser $p(\cdot)$ non sur $f(\cdot)$, mais sur une approximation de $f(\cdot)$ [Crainic & al. 1993b]. Lors de la sélection des transitions, il n'est alors plus certain de retenir celle qui améliore le plus $f(\cdot)$.

A partir de là, on peut constater qu'il y a un compromis à trouver, lors de la modélisation d'un problème d'optimisation, entre la complexité de la technique de recherche locale mise en oeuvre et la qualité des solutions produites. Plus une technique de recherche locale est complexe, meilleures sont les solutions qu'elle engendre, toutefois plus elle est complexe, plus elle requiert de temps de calcul.

b) Décomposition des transitions

Afin d'utiliser pleinement les facultés de modélisation résultant de l'introduction d'un langage formel de modélisation dans *Alois* (cf. p. 88), nous allons redéfinir la notion de transition. En effet, selon la technique de recherche locale introduite ci-dessus, les transitions conduisent toujours d'un état solution à un autre état solution. Nous allons donc relâcher cette contrainte en admettant le passage par des états qui ne sont pas des solutions au problème d'optimisation.

Prenons à titre d'exemple le problème d'ordonnancement simple à une seule machine (cf. p. 181). La modélisation d'origine utilise une transition `insère` qui enlève une tâche de l'ordonnancement pour la replacer ensuite dans une autre position. La taille du voisinage de `insère` est de $O(n^2)$, n étant le nombre de tâches de l'instance de problème. En effet, il y a n tâches que l'on peut enlever et pour chaque tâche n places possibles [Barnes & Laguna 1993]. Cependant, cette transition comporte deux actions, une action `enlève` qui correspond à l'enlèvement de la tâche et une action `place` qui revient à placer à nouveau la tâche sur la machine. Chacune de ces deux actions est de complexité linéaire $O(n)$. En séparant ces actions et en les appliquant de manière séquentielle (et sans retour en arrière), la taille totale de leurs voisinages n'est que de complexité $O(2 \cdot n)$, donc toujours linéaire (figure 3).

Pour ce qui est de la terminologie, nous utilisons *transition* pour désigner une opération qui conduit directement d'un état solution vers un autre état solution ; tandis qu'un *opérateur* passe par un état non-solution. Nous appelons par ailleurs *transformation* une séquence d'opérateurs menant d'une état solution vers un autre état solution en ne passant que par zéro ou par plusieurs états non-solutions. Ainsi, `insère` est une transition, `enlève` et `place` sont des opérateurs et la séquence (`enlève`, `place`) est une transformation. Notons en outre que nous adoptons pour la représentation des opérateurs la forme canonique des règles de production, introduites dans le deuxième chapitre (cf. p. 46).

Le gain au niveau de la taille des voisinages s'accompagne cependant de deux inconvénients. D'une part, lors de la sélection d'un opérateur qui ne conduit pas à un état solution, il n'est pas possible d'utiliser la fonction de prévision définie par (15). En effet, la fonction objectif n'est pas définie pour un état non-solution. Il faut donc employer une autre définition pour la fonction de prévision. D'autre part, un contrôle de l'enchaînement des opérateurs est nécessaire. Il doit assurer qu'un état solution soit vraiment atteint au bout d'une transformation. S'agissant de décomposer des transitions, il est possible de modéliser le contrôle par un simple diagramme de transitions d'état. Le diagramme d'exemple de la figure 4 indique qu'à partir de l'état 0 un opérateur `enlève` est appliqué. Dès lors, dans l'état 1, il est possible d'appliquer soit un opérateur `place` pour regagner immédiatement un état solution (état 0), procédé similaire à une transition `insère`, soit un deuxième opérateur `enlève` (état 2). Dans ce dernier cas, l'application de deux opérateurs `place` mène à nouveau à un état solution similaire à l'exécution d'une transition `échange`.

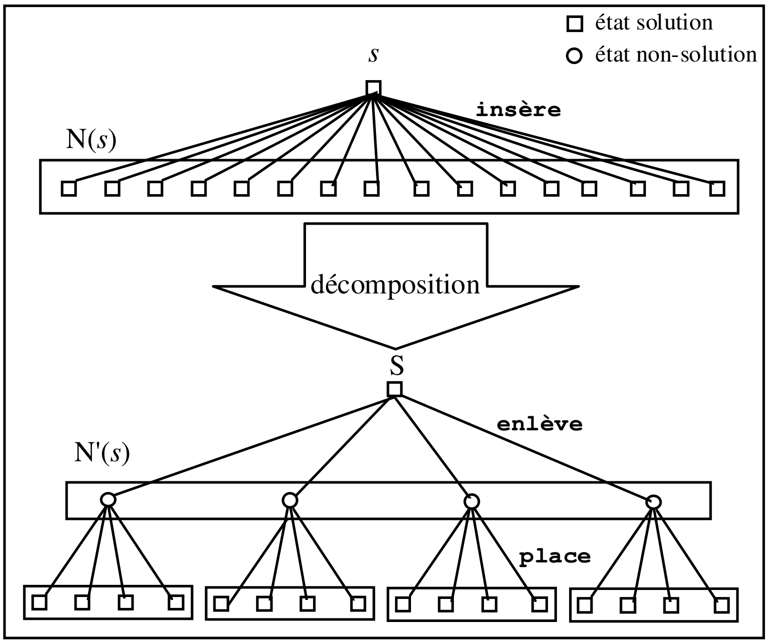


Figure 3. Décomposition des transitions pour le problème d'ordonnancement simple à une machine

Cette décomposition des transitions, même si elle est simple, nous permettra par la suite de recourir à une modélisation de plus petit grain des problèmes d'optimisation. La technique d'apprentissage, que nous introduirons au chapitre 3, s'appuiera notamment sur ce type d'opérations. Nous utiliserons la modélisation par décomposition des transitions non seulement pour le problème d'ordonnancement, mais aussi pour le problème de placement et d'allocation (cf. p. 166).

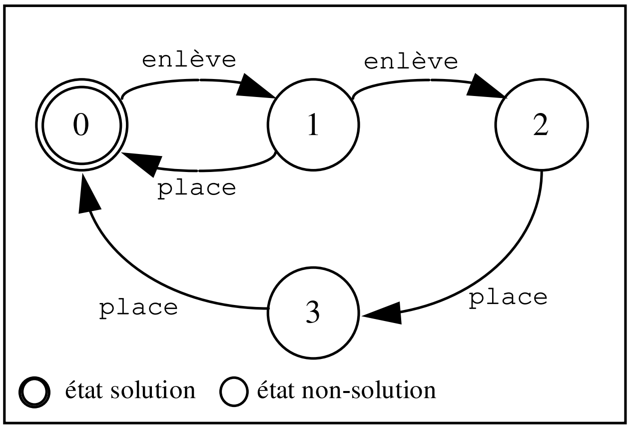


Figure 4. Exemple du contrôle pour l'enchaînement des opérateurs dans le problème d'ordonnancement à une machine

1.1.6. L'apprentissage en recherche locale

Les imperfections de la recherche locale ont, dès son origine, motivées certaines propositions sur l'incorporation d'algorithmes d'apprentissage. En effet, les concepteurs de ces techniques se trouvaient confrontés à une méthode simple qui, de manière très répétitive, explore l'espace de recherche de manière aveugle. Dès lors, l'objectif des techniques d'apprentissage a été d'améliorer cette démarche par trop limitée.

Une première approche consiste à réduire la complexité de l'instance de problème que l'on optimise au fur et à mesure de l'optimisation. Définie dans le cadre du problème du voyageur de commerce, cette approche s'appuie sur la collecte de données statistiques relatives à la fréquence des apparitions d'un même groupe de villes, reliées entre elles selon un ordre identique, dans plusieurs solutions de bonne qualité. Deux manières d'exploiter ces informations ont été proposées. Lin (1965) suppose que le chemin reliant ces villes doit être de très bonne qualité, puisqu'on le retrouve fréquemment dans des bonnes solutions. Il enlève tout le groupe de villes de la description de l'instance de problème et remplace le chemin par un seul arc liant directement ces villes initiales et finales. La taille de l'instance de problème et, en conséquence, sa complexité s'en trouvent réduites.

La deuxième approche repose sur une hypothèse qui est tout à fait à l'opposé de celle de [Lin 1965]. De la présence d'un même sous-chemin dans plusieurs solutions de bonne qualité Steiglitz & Weiner (1968) induisent que la technique d'optimisation se trouve par trop attirée par ce sous-chemin, et qu'en conséquence étant quelque peu prisonnière de celui-ci, elle propose des solutions de qualité médiocre. Ils en concluent qu'il faut interdire la présence de ce sous-chemin dans les nouvelles solutions, et donc l'ôter de l'instance de problème. Notons que ces deux approches font figure de précurseurs des techniques d'intensification et de diversification employées dans la méthode tabou, que nous verrons dans ce qui suit.

Une autre approche est proposé par Robert & Flores (1966). Travaillant également sur le problème du voyageur de commerce, ils ont remarqué que quelques sous-chemins figurent dans plusieurs bonnes tournées. Ils tentent alors de combiner ces morceaux de solutions afin de produire de nouvelles tournées de meilleure qualité.

Les différentes techniques d'apprentissage que nous venons d'évoquer n'ont pas eu le succès attendu par leurs auteurs, ceci certainement en raison de l'absence d'une méthodologie les réunissant. Ce n'est qu'avec l'apparition de la méthode tabou que l'apprentissage a recouvré un intérêt dans les techniques heuristiques de résolution de problèmes d'optimisation.

1.2. La recherche tabou

La *recherche tabou* ("*tabu search*") [Glover 1989, 1990] a pour objectif d'améliorer des techniques d'optimisation du type recherche locale. Souvent qualifiée de *méta-heuristique*, au même titre que le recuit simulé et les algorithmes génétiques, elle est indépendante de toute heuristique d'optimisation. En intelligence artificielle, elle serait qualifiée de "indépendante du domaine". La recherche tabou élargit la recherche locale classique grâce à un ensemble de techniques servant au contrôle de l'exploration de l'espace de recherche. Ces techniques englobent des processus de mémorisation et d'apprentissage.

Introduite à l'origine par Glover (1986), la méthode tabou connaît aujourd'hui un succès incontestable en termes de construction de bonnes, et très bonnes, solutions pour des problèmes d'optimisation combinatoire de complexité industrielle. Un article récent inventorie plus de soixante-dix différents domaines d'application de la méthode tabou [Glover 1995]. Les présenter tous ici n'est pas envisageable. En revanche, nous présenterons les principaux composants de la recherche tabou, ce qui nous permettra de passer en revue différentes techniques d'apprentissage qui ont été proposées dans son cadre. Notre tout dernier chapitre sera, par ailleurs, consacré à la présentation de trois exemples de méthodes tabou (cf. p. 166, p. 181 et p. 187).

1.2.1. Le principe

La méthode tabou étant une technique de recherche locale, tout ce que nous avons pu constaté, au cours de la dernière section, concernant cette dernière reste vrai pour elle (cf. p. 25). La méthode tabou étend la recherche locale par un contrôle actif du processus de recherche afin d'accélérer l'identification de solutions de bonne qualité. Pour ce faire, elle emploie explicitement des techniques de mémorisation et d'apprentissage, dont la justification est dérivée de l'hypothèse fondamentale suivante :

> *Hypothèse de la recherche tabou* : Lors de la résolution d'une instance d'un problème d'optimisation, la recherche itérative de solutions de bonne qualité met en évidence certaines caractéristiques de l'espace de recherche qui sont propres à l'instance. La réutilisation de ces informa-

tions permet d'orienter l'exploration de l'espace de recherche de sorte qu'elle conduise à des solutions de meilleure qualité.

Les techniques de mémorisation et d'apprentissage sont responsables de l'identification et de la réutilisation des informations caractéristiques d'une instance de problème. La définition générale de la méthode tabou [Glover 1989, 1990] présente plusieurs mécanismes que l'on peut employer à cette fin. Leur mise en oeuvre pratique est, quant à elle, fonction du problème d'optimisation. Cependant, il est loin d'être facile de déterminer la pertinence d'un mécanisme pour la résolution d'un problème d'optimisation sans recours à des expériences.

De manière générale, le contrôle mis en oeuvre par une méthode tabou se situe à deux niveaux différents. D'une part, un niveau de contrôle tactique fournit à la recherche un contrôle réactif, essentiellement basé sur la mémorisation, et d'autre part, un niveau de contrôle stratégique met en oeuvre des fonctions d'apprentissage et de réutilisation afin d'analyser plus profondément la recherche antérieure. Ces deux niveaux de contrôle sont complémentaires, et tout deux contribuent en général de manière équivalente au succès d'une implantation de la méthode tabou.

a) Le contrôle tactique de la recherche

Lors de la première section de ce chapitre, nous avons remarqué que la recherche locale parcours un espace de solutions S en appliquant des transitions m afin de passer d'un état solution s à un autre. Le voisinage d'un état solution $N(s)$ est défini dans (14) (cf. p. 25). Il est composé par l'ensemble des états solutions que l'on peut atteindre au moyen de l'ensemble des transitions applicables à s. Ce type d'exploration est sujet au problème des minima locaux où l'exploration est susceptible d'être emprisonnée, en raison du phénomène de bouclage (cf. p. 27).

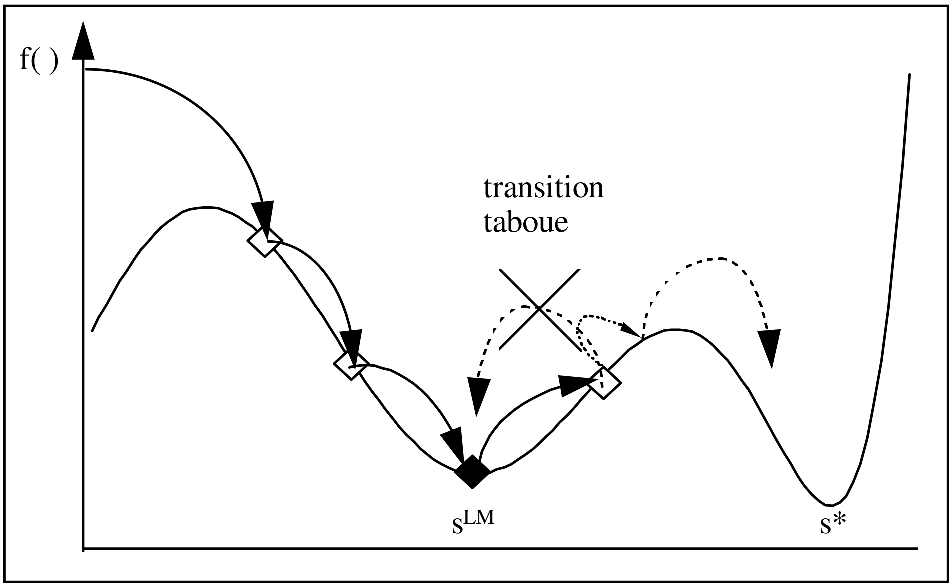


Figure 5. Echapper à un minimum local s^{LM}

Le premier objectif du contrôle tactique de la recherche tabou est de proposer un outil permettant de quitter les minima locaux (figure 5). Cet outil est la *liste tabou L.* Chaque transition appliquée (ou, le plus souvent la transition qui inverse une transition appliquée) est inscrite dans la liste tabou. Cette liste est du type *file*, et donc gérée d'après la règle du premier-entré-premier-sorti. Ainsi, une transition nouvellement inscrite remplace dans la liste tabou la plus ancienne. Tout transition $m \in L$ est interdite d'application et en conséquence déclarée *tabou*, d'où le nom de la méthode. En conséquence la définition des voisinages d'un état s se trouve modifiée. Appelons $N^T(s)$ ce nouveau voisinage. Il se définit ainsi :

$$\forall s \in \mathrm{S}, \quad N^T(s) = \{s' \mid \exists m \in M - L,\ s' = m(s)\} \tag{17}$$

On peut attribuer à la liste tabou un objectif plus général que celui d'échapper au puits d'optimalité locale, tel que celui de pousser la recherche continuellement en avant.

Avec ce mécanisme simple, l'exploration de l'espace de recherche est continuellement poussée à aller de l'avant. Ceci devrait permettre de franchir les puits d'optimalité des minima locaux. Une méthode tabou peut contenir une ou plusieurs listes tabou, selon les besoins. Une liste tabou peut être de taille fixe ou, le plus souvent préférable, de taille variable [Taillard 1991]. La taille de la liste tabou, c'est à dire sa longueur, est aussi importante. Si la liste est trop courte, la recherche n'arrive pas à dépasser la région attractrice de certains minima locaux. Cependant, si la liste est trop longue, la qualité des solutions s'en trouve affectée, car les voisinages des états sont trop clairsemés [Skorin-Kapov 1990]. Au début de la méthode tabou, des chercheurs [Glover 1986, Hertz & De Werra 1987] préconisaient des listes de taille 7, nombre magique [Miller 1956].

Toutefois, même si la taille de la liste tabou est choisie convenablement, il peut y avoir un élagage du voisinage N^T conduisant à la suppression de transitions menant à des états successeurs intéressants. Le *critère d'aspiration* $A(m, s)$ est conçu pour remédier partiellement à ce problème [Glover 1989]. Il permet l'acceptation de transitions qui seraient a priori tabou mais dont l'intérêt de l'état successeur justifie le maintient dans N^T. On retient alors que toute transition taboue m qui satisfait $f(m(s)) < A(m, s)$ voit sa qualité de “tabou” annihilée.

Il existe différentes manières de définir $A(m, s)$. Celle le plus souvent employée recourt au vecteur $Best(q)$, avec q défini sur l'ensemble des valeurs que peut prendre la fonction objectif. Pour $q = f(s)$, ce vecteur contient la meilleure valeur $f(m'(s'))$ atteinte, au cours de la même optimisation, par une transition m' appliquée à un autre état s', antérieur à s, qui était de qualité q, $f(s') = q$. En conséquence, une transition m satisfait au critère d'aspiration si elle conduit à un état successeur $m(s)$ qui est d'une meilleure qualité $f(m(s))$ que tout autre état successeur jusque là atteint à partir d'un état de même qualité que s [Glover 1989].

Un exemple clarifie ceci. Prenons un état s_1 avec $f(s_1) = 10$ et appliquons une transition m_1 menant de s_1 vers un état de qualité $f(m_1(s_1)) = 9$. Ceci conduit à fixer $Best(10) := 9$. Supposons que plus tard, la recherche se trouve dans un état s_2 avec à nouveau $f(s_2) = 10$ et une transition applicable m_2 menant vers un état de qualité $f(m_2(s_2)) = 8$. Cette transition m_2 satisfait alors au critère d'aspiration, puisque $f(m_2(s_2)) < Best(f(s_2))$. Même si m_2 était taboue, elle ferait à titre exceptionnel partie du voisinage de s_2. Au cas où m_2 est appliquée, le critère d'aspiration est actualisé en mettant $Best(10) := 8$.

Si cette définition du critère d'aspiration parait à première vue un peu laborieuse, elle est néanmoins astucieuse. Défini de cette manière, le critère d'aspiration n'accepte que des transitions dont il est établi qu'elles ne font pas partie d'une boucle de la recherche. La raison d'être, principale, de la liste tabou étant d'éviter que l'exploration boucle, il n'y a aucun intérêt à maintenir tabou une transitions satisfaisant à ce critère d'aspiration.

b) Le contrôle stratégique

Outre ces mécanismes portant sur le contrôle tactique, et immédiat, qui tendent à remédier au problème des minima locaux, la méthode tabou définit deux stratégies de contrôle de niveau plus élevé, *l'intensification* et la *diversification*. Ces deux techniques relèvent d'une volonté de véritablement contrôler l'exploration de l'espace recherche. L'intensification vise à focaliser la recherche sur des zones prometteuses où des solutions de très bonne qualité pourraient être localisées. La diversification, quant à elle, a pour but de conduire l'exploration vers des zones qui n'ont pas encore été explorées. De nature complémentaire, ces deux techniques peuvent considérablement contribuer à la construction d'excellentes solutions. Tant l'intensification que la diversification ont été influencées par les premières techniques d'apprentissage proposées pour la recherche locale (cf. p. 34). La méthode tabou est allée plus loin en réunissant diverses techniques d'apprentissage dans un même cadre.

Schématiquement, l'intensification et la diversification consistent à fixer quelques variables du problème d'optimisation et à en interdire le changement pendant un certain nombre d'itérations. Considérons quelques exemples typiques de ces deux techniques. L'intensification consiste souvent à fixer la valeur de variables qui ont, au préalable, été identifiées comme étant caractéristiques d'une structure présente dans des bonnes solutions (exemple : dans le problème de placement et d'allocation, le nombre d'entités qui sont placées [Crainic & al. 1993b]). Arrêter ainsi les valeurs des variables focalise l'exploration sur un sous-ensemble de l'espace de recherche. En outre, les voisinages se trouvent élagués des transitions modifiant ces variables, ce qui accélère la recherche.

La diversification, quant à elle, conduit souvent à fixer des variables à des valeurs que celles-ci n'avaient pas pris depuis longtemps, ou encore à appliquer en priorité des transitions qui n'ont pas été appliquées depuis longtemps. En outre, il est possible de recourir, de temps à autre, à des transitions spéciales qui modifient profondément la structure de l'état solution actuel, mais qui sont trop complexes pour être évaluées à chaque itération. La sélection aléatoire de transitions constitue également un moyen de produire ce type d'effet [Faigle & Kern 1992, Rochat & Taillard 1995]. Prenons comme exemple le problème de l'affectation quadratique, où une première réalisation conduit à préconiser les transitions peu souvent utilisées [Skorin-Kapov 1990], et une seconde recommande de favoriser les transitions menant vers des solutions nouvelles [Taillard 1991].

Le contrôle stratégique est un composant important des méthodes tabou puisqu'il les assiste dans la production de solutions de bonne qualité [Kassou 1992], et les techniques proposées figurent clairement au nombre des techniques d'apprentissage [Glover 1989, 1990], puisque leur but est d'acquérir et de réutiliser l'expérience relative à l'optimisation d'une instance de problème particulière.

Il faut toutefois admettre que, si les deux stratégies de contrôle se justifient essentiellement par des raisons intuitives et par des résultats concrets [Glover 1989, 1990], leur conception et leur mise en oeuvre, propres à chaque problème d'optimisation, nécessitent une phase d'expérimentation et d'analyse non négligeable, qui ne constituent pas des tâches des plus simples. En outre, le succès obtenu lors de l'application d'une technique de contrôle stratégique, sur un problème d'optimisation particulier, ne garantit en rien que cette même technique donne des résultats aussi intéressants sur un autre problème d'optimisation. Le contrôle stratégique efficace doit être conçu pour chaque nouveau problème d'optimisation.

Si les méthodes esquissées ci-dessus sont, bien qu'efficaces, très simples et spécifiques à chaque problème, des approches plus récentes tendent à rendre ces techniques d'acquisition et de réutilisation d'expérience plus universelles. Nous les considérerons dans le prochain paragraphe.

1.2.2. L'apprentissage en recherche tabou

L'intensification et la diversification peuvent être qualifiées de techniques d'apprentissage, puisqu'elles acquièrent et réutilisent des expériences qui sont relatives à une instance de problème. La conception de ce type de techniques qui sont à la fois génériques, c'est à dire qui s'appliquent à divers problèmes d'optimisation, et performantes, dans le sens où elles améliorent considérablement la qualité des solutions, est un problème important mais qui reste ouvert [Glover 1995]. Plusieurs approches visant ce problème ont été proposées. Nous présenterons les trois approches majeures dans les développements qui suivent : la recherche tabou réactive, l'analyse de la cible et le chunking.

a) La recherche tabou réactive

La *recherche tabou réactive* (“*reactive tabu search*”) propose une approche de l'intensification et de la diversification par le biais d'une gestion évoluée de la taille de la liste tabou [Battiti & Tecchiolli 1994]. Elle est à la recherche permanente d'un compromis dynamique entre une liste tabou assez courte, qui n'élague pas trop les voisinages, et une liste suffisamment longue qui évite le bouclage de l'exploration.

La recherche tabou réactive observe l'exploration de l'espace de recherche. Dès qu'elle détecte une répétition excessive d'états identiques elle augmente de manière drastique la longueur de la liste tabou afin d'entreprendre une phase de diversification. Par la suite, et dans le but d'éviter les inconvénients liés à cette liste par trop longue, cette dernière se trouve lentement réduite au fur et à mesure que la recherche progresse. S'il est en outre constaté que l'exploration reste prisonnière d'une région de l'espace de recherche, la recherche tabou réactive prévoit le recours à une seconde technique de diversification consistant simplement en la sélection aléatoire d'un certain nombre de transitions.

Brièvement, on peut constater que la recherche tabou réactive tend à combiner les avantages des listes tabou courtes et longues. Toutefois, elle requiert une mémoire contenant l'ensemble des transitions appliquées au cours d'une optimisation. Ainsi, à chaque itération, cette mémoire est appelée afin de vérifier s'il y a ou non bouclage. L'utilisation de techniques de *hashing* rend la charge, résultant de l'interrogation de la mémoire, acceptable.

b) L'analyse de la cible

La méthode *d'analyse de la cible* (“*target analysis*”) est à notre connaissance la seule technique d'apprentissage appliquée en méthode tabou qui vise à réutiliser les connaissances apprises au travers de plusieurs instances de problèmes appartenant à la même classe de problèmes d'optimisation [Glover & Greenberg 1989, Laguna & Glover 1993]. Pour cela, elle commence par optimiser des instances représentatives de la classe de problème cible. La trace de ces premières optimisations est ensuite analysée afin de mettre en évidence les erreurs commises dans la sélection des transitions. Ces erreurs peuvent par exemple concerner la sélection tardive d'une transition qui, plus tard, devait mener à une solution d'excellente qualité. Les résultats de ces analyses sont ensuite utilisés afin d'élaborer une stratégie de sélection de transitions qui ne commette plus les anciennes erreurs. En utilisant cette stratégie sur des instances appartenant à la même classe de problèmes on devrait donc atteindre plus rapidement des solutions de bonne qualité.

Cette méthode a deux inconvénients majeurs. D'une part, les définitions des notions telles que “classe de problème” et “instance représentative” ne sont pas précises. Cela rend difficile leur généralisation à d'autre problèmes d'optimisation en dehors du domaine d'application original qui est l'ordonnancement. D'autre part, la mise en oeuvre de la méthode d'analyse de la cible est assez lourde, puisqu'elle nécessite une optimisation préliminaire (et extensive) de plusieurs instances du problème. De même, la phase d'analyse des traces de solutions peut être coûteuse, lorsqu'elle peut se faire de manière automatique. Sans quoi elle implique fortement l'utilisateur [Glover & Laguna 1991].

c) Le chunking

Une technique basée sur le *chunking*[1] a été proposée pour la méthode tabou [Woodruff 1994]. Cette méthode repose sur l'identification de traits communs aux

---

[1] Cette technique est indépendante de la technique d'apprentissage à partir d'explications du même nom employée par *Soar* dans le cadre de la génération de plans [Laird & al. 1986].

bonnes solutions. Ainsi, un *chunk* porte sur un sous-ensemble des variables d'une instance de problème. Il est instancié dans tout état solution par les valeurs que prennent ses variables. Ensuite, une fonction d'évaluation du chunk permet d'établir une valeur qui est caractéristique de l'instanciation du chunk dans cet état.

A partir de ces valeurs caractéristiques des chunks instanciés, la méthode propose le recours à un contrôle tactique et/ou stratégique de la recherche. A cette fin, elle préconise les mêmes techniques déjà présentes dans la méthode tabou classique. Ainsi, on peut par exemple déclarer taboue toute transition qui annule une modification de la valeur caractéristique d'un chunk. De même, des techniques d'intensification et de diversification, basées sur la fréquence de modification des caractéristiques des états solutions sont directement transposables dans les chunks.

Une question importante concerne l'identification des chunks intéressants et la définition de leurs fonctions caractéristiques. Une première approche, envisageable lorsque l'on traite un problème réel, est d'interroger les utilisateurs sur des bouts de solutions qui leurs sont familiers. Une autre démarche conduit à identifier ces chunks par une comparaison des bonnes solutions obtenues lors de la résolution d'instances test.

Au travers de ces trois méthodes nous avons pu donner une présentation qualitative des techniques d'apprentissage actuellement utilisées en recherche tabou. On peut remarquer que ces méthodes ne sont que de loin liées aux techniques d'apprentissage analytique telles qu'elles sont connues en intelligence artificielle et lesquelles nous présenterons dans le prochain chapitre.

Chapitre 2.

Résolution de problèmes, apprentissage et raisonnement à partir de cas

Au cours du premier chapitre, nous avons pu constater qu'il y a un intérêt à intégrer des techniques d'apprentissage dans des méthodes d'optimisation du type recherche locale. Or, en intelligence artificielle existe une branche de recherche florissante portant sur des algorithmes d'apprentissage, celle de *l'apprentissage analytique*. Les techniques qui y sont développées tendent précisément à améliorer la compétence et la performance d'algorithmes. Cependant, par vocation, l'apprentissage analytique s'applique de manière classique aux algorithmes de *génération de plans*, tels qu'employés en intelligence artificielle, depuis quasiment l'origine de la discipline, pour la résolution de problèmes.

Dans la première section, il convient donc de présenter ce que l'on entend par *résolution de problèmes* en intelligence artificielle, avant d'évoquer brièvement les algorithmes de génération de plans. A la fin de cette section, nous serons à même de désigner ce qui nous parait être la différence principale entre les problèmes traités par ces deux disciplines.

Nous présenterons ensuite, dans la deuxième section, l'apprentissage analytique. Les algorithmes d'apprentissage analytique, tels que développés dans le domaine de la génération de plans en intelligence artificielle, n'étant malgré tout pas adaptés à cette tâche, il se pose la question de la nature de la technique d'apprentissage à employer. Le recours au raisonnement à partir de cas constitue selon nous la réponse appropriée, nous justifierons notre position, mais seulement dans le chapitre 3.

Le raisonnement à partir de cas, quant à lui, a donné aux systèmes à base de connaissances la capacité de représenter et d'employer de manière efficace des connaissances de nature opérationnelle. Dans la troisième et dernière section de ce chapitre, nous présenterons ce type de techniques et examinerons son aptitude à améliorer la résolution de problèmes issus de la recherche opérationnelle.

2.1. Résolution de problèmes

Dans cette section, nous présenterons l'une des principales techniques de l'intelligence artificielle, la *résolution de problèmes* (“*problem solving*”). Elle est intéressante à deux titres. D'une part, elle est d'une ampleur telle qu'il parait inévitable d'établir une relation entre elle et les problèmes d'optimisation. D'autre part, de nombreux algorithmes d'apprentissage ont été développés dans ce cadre. Présenter la résolution de problèmes nous permettra donc, dans une section ultérieure, d'analyser ces algorithmes d'apprentissage, d'en retenir les buts et les moyens, et de les considérer en vue d'une intégration dans une méthode d'optimisation.

2.1.1. Les systèmes symboliques

Un *système symbolique physique* (“*physical symbol systems*”) [Newell & Simon 1976] est composé d'un ensemble de *symboles* qui sont des représentations d'entités physiques et qui sont liés entre eux dans des structures ; d'un ensemble de *méthodes* qui savent manipuler ces structures de symboles ; et d'un mécanisme (ou générateur) qui, en utilisant les méthodes, transforme les symboles ainsi que leur structure. Un problème, quant à lui, comporte d'un *test* qui permet de déterminer pour toute structure de symboles, si elle est, ou si elle n'est pas, une solution au problème. McCarthy (1956) qualifie de *bien-défini* (“*well-defined*”) un problème pour lequel il existe un test qui est calculable[1]. Un système symbolique résout des problèmes en transformant progressivement ses structures de symboles. Il s'arrête lorsqu'il en a trouvé une qui satisfait au test. Cette technique s'appelle *résolution de problèmes* (“*problem solving*”) [Newell & Simon 1963].

Tout système, y compris symbolique, ne dispose *en pratique* que d'un nombre limité de ressources, qu'il s'agisse de la capacité de calcul, de stockage, ou de tout autre élément. Il est donc, en général, impossible au générateur d'explorer l'espace des structures de symboles dans sa globalité. Il doit en conséquence être sélectif dans le choix de ses actions. Cette sélection des actions s'appuie sur des *heuristiques* permettant de distinguer entre actions avantageuses et actions infructueuses. Ce processus d'exploration de l'espace des solutions est appelé *recherche heuristique* [Newell & Simon 1976]. Il est complet et systématique.

A cet endroit, il nous semble important de remarquer que la notion *d'heuristique* employée en optimisation n'est pas la même que celle employée en intelligence arti-

---

[1] Simon (1973) reprend cette notion de “bien-défini” comme point clé, absent, des problèmes *mal-structurés* (“*ill-structured*”).

ficielle [Minski 1963]. En optimisation, les solutions qui sont construites par des heuristiques ne sont pas nécessairement optimales, mais seulement *de bonne qualité*, tandis qu'en recherche heuristique, toute solution engendrée est *parfaite* dans le sens où elle satisfait au test. Ici, le terme “heuristique” concerne par contre le contrôle de l'exploration de l'espace de recherche, donc le processus qui engendre les solutions. Le but est alors d'employer un contrôle “intelligent” de la recherche afin d'éviter le parcours complet de l'espace de recherche. Suivant les termes employés en optimisation, la recherche heuristique en IA est une méthode exacte.

2.1.2. Deux techniques de résolution de problèmes

Le cadre des systèmes symboliques et de la résolution de problèmes a motivé une partie importante de la recherche en intelligence artificielle et a donné lieu au développement de nombreuses méthodes. Dans ce qui suit, nous en considérerons deux, prépondérantes : la recherche heuristique du type $A*$ et les systèmes de productions.

a) La recherche heuristique du type $A*$

L'*algorithme* $A*$ [Nilsson 1980, Pearl 1984] traite du problème du plus court chemin dans un espace d'états. Cet espace est donné de manière implicite par un état initial et un ensemble de transitions. Un état solution est un état qui satisfait au *but*, et l'état final recherché est l'état solution le plus proche de l'état initial. Pour tout état membre de l'espace de recherche, l'opération qui consiste à vérifier si le but est atteint est facile. La fonction d'évaluation $h*(E)$ indique pour chaque état E une estimation de la longueur du chemin de E vers l'état solution qui lui est le plus proche. A chaque itération, l'algorithme sélectionne le noeud pour lequel la somme de sa distance de l'état initial et de $h*(E)$[2] est minimale. L'exploration de l'espace de recherche est complète et systématique. L'algorithme $A*$ est un cas spécial des méthodes de séparation et d'évaluation [Nau 1984] et, en conséquence, une technique exacte, et non pas une heuristique, dans le sens où l'entend la recherche opérationnelle.

Dans ce bref exposé, on reconnaît que $A*$ contient l'ensemble des caractéristiques de la résolution de problèmes bien-définis présentés ci-dessus. Les descriptions des états correspondent aux structures de symboles. Elles sont manipulées par les transitions que le générateur applique. Le “but” correspond au “test”, et le rôle de la fonction d'évaluation $h*$ est alors de fournir l'heuristique nécessaire pour guider de manière efficace la recherche. La ressemblance entre l'algorithme $A*$ et la recherche locale apparaît

---

[2] Puisqu'en général $h*$ n'est pas connue, la fonction h, de préférence minorante de $h*$, est employée.

aussi. Tout les deux parcourent un espace d'états, guidés par une fonction qui évalue la qualité des états rencontrés. La différence majeure se situe toutefois au niveau du test de solution, c'est à dire du but. Si celui-ci est efficace pour ce qui est de A^*, tel n'est pas le cas en recherche locale où le but est d'identifier l'état qui minimise la fonction objectif.

b) Les systèmes de productions

Les *systèmes de productions* constituent l'un des formalismes prépondérants parmi les systèmes à base de règles [Hayes-Roth & al. 1978, Giarrantano & Riley 1994]. Ils sont composés d'un *moteur d'inférence*, d'un ensemble de *règles de productions* et d'un ensemble de données, appelées *faits* se trouvant dans la mémoire de travail, qui, elle, s'appelle *base de faits*. Une règle de production est un couple (conditions - actions). Les actions modifient la base de faits en ajoutant, effaçant ou transformant des faits. Les conditions décrivent les caractéristiques auxquelles doit satisfaire la base de faits pour qu'une règle de production devienne applicable. Une règle de production applicable est appelée *instanciation*[3]. Si, pour une même base de faits, il existe plusieurs manières de satisfaire aux conditions d'une même règle de production, chacune donne alors lieu à une instanciation. C'est le moteur d'inférence qui manipule les règles de production. Il détermine les instanciations par le processus du *filtrage*, et les inscrit dans son *agenda*.

La tâche d'un système de productions est de construire, à partir de la base de faits initiale et à l'aide des règles de productions, une base de faits finale qui satisfasse au *but*. Le but est un ensemble de faits. Il y est satisfait lorsque ses faits sont contenus dans la base de faits. Pour ce faire, le moteur d'inférence enchaîne des instanciations. Lors de la phase de *résolution de conflits*, une instanciation appartenant à l'agenda est sélectionnée. Ensuite sa partie action est appliquée, et l'agenda est mis à jour afin de prendre en compte les modifications apportées à la base de faits. Le pouvoir d'un système de productions d'engendrer un état satisfaisant au but dépend fortement de sa stratégie de résolution des conflits [McDermott & Forgy 1978]. Les heuristiques employées à cette fin, les *connaissances de contrôle*, constituent un composant essentiel des systèmes de productions.

Le lien entre les systèmes de productions et la résolution de problèmes est apparent. La base de faits correspond à une structure de symboles. L'espace de recherche est exploré par le moteur d'inférence qui applique les règles de production, et qui est guidé par les connaissances de contrôle. Le “test” est mis en oeuvre au travers du “but”.

---

[3] On emploie souvent la terminologie plus générale de Strips, où la production s'appelle *schéma d'opérateur* et l'instanciation s'appelle *opérateur* [Fikes & Nilsson 1971]. Nous reprendrons cette convention.

Il est efficace puisqu'il consiste simplement à vérifier la présence d'un nombre fini de faits dans la base de faits.

Le mode de fonctionnement d'un système de productions présenté ci-dessus s'appelle *chaînage avant*, le système parcourant l'espace de recherche de l'état initial vers l'avant afin d'atteindre un état but. Il existe une deuxième manière d'explorer l'espace de recherche, appelée *chaînage arrière*. Dans ce cas, la recherche a pour point de départ le but et s'oriente par référence à la différence qu'il y a entre celui-ci et la base de faits (“*means end analysis*”) [Newell & Simon 1963].

On appelle *plan* l'enchaînement des instanciations qui mène de l'état initial à l'état final. La notion de plan est liée à la démonstration de théorèmes : un plan correspond à la démonstration d'un théorème. Les systèmes de productions sont alors souvent appelés *générateurs de plans de première génération* [Tate & al. 1990]. D'autres approches de génération de plans existent, telle que la planification non-linéaire [Minton & al. 1994] qui se ramène souvent à une planification dans un espace de plans [Sacerdoti 1977]. Nous limitons notre exposé aux systèmes de productions car c'est essentiellement ceux-ci qui ont fait l'objet des algorithmes d'apprentissage. Pour prendre connaissance de l'état de l'art en matière de génération de plans, on se reportera à [Georgeff 1987, Tate & al. 1990]. Dans les développements qui suivent, nous emploierons comme des synonyme les expressions “générateur de plans”, “système de productions” ou “générateur de plans de première génération”.

2.1.3. Résolution de problèmes versus optimisation

Nous avons présenté les préoccupations et les méthodes qui gouvernent la *résolution de problèmes* et *l'optimisation*. Bien que ces deux domaines se ressemblent beaucoup, leurs différences, à la mise en évidence desquelles nous consacrerons ce paragraphe, influenceront fortement notre travail. Nous attirerons l'attention sur la différence qui existe concernant la notion de but. Celle-ci est importante puisque les algorithmes d'apprentissage, dont nous traiterons plus loin, y ont recours.

a) Deux classes de problèmes différents

Les problèmes d'optimisation sont d'une nature différente de celle des problèmes de planification. Nous avons évoqué plus haut la différence entre les problèmes de décision et les problèmes d'optimisation (cf. p. 19), nous la rencontrons à nouveau ici. On peut observer que la résolution de problèmes apporte des solutions à des problèmes de décision. Un système de productions, ou un générateur de plans, en construisant un plan, vérifie de manière implicite si une solution existe. S'il n'existe aucun état solution, le générateur de plans ne peut bien évidemment pas en trouver. En revanche, s'il en existe, ne serait ce qu'un seul, il le trouve[4], et en outre, il propose un plan permettant de l'atteindre à partir de l'état initial. Ce sont, en effet, les plans qui sont souvent considérés comme solutions, et non pas l'état final lui-même.

Le problème de la génération de plans peut être formulé comme étant un problème d'optimisation. Il s'agit de définir une fonction objectif telle que “construit un plan qui soit de longueur minimale”, et de donner au système la capacité de générer divers plans et de les comparer entre eux. Cependant, ceci est peu commun en génération de plans basée sur les systèmes de productions. Un générateur de plans ne tente pas de générer des plans optimaux [Georgeff 1987, Fox 1994].

b) Satisfaire des buts de nature différente

Après ces considérations sur les classes des problèmes traités, nous abordons à présent les notions de “but” et de “satisfaction aux buts”. En termes de système de productions, le but est l'implantation du test défini dans la résolution de problèmes. Il est aisément vérifiable, donc bien-défini ; un simple examen de l'état actuel de la recherche suffit pour constater s'il satisfait au but ou pas. De ceci découle que la satisfaction aux buts est définie pour chaque état de manière locale.

Pour les problèmes d'optimisation, il n'en va pas ainsi. En l'occurrence, il s'agit de minimiser la fonction objectif sur l'ensemble de l'espace de recherche. Le but est donc d'identifier l'état solution pour lequel cette fonction objectif est minimale. Un état solution est optimal s'il satisfait au *critère d'optimalité* (équation 4, cf. p. 19). En effet, l'état final optimal doit satisfaire à ce critère global, défini par rapport à chaque état solution de l'espace de recherche. Ce critère n'est en général pas facile à vérifier [Minoux 1987].

Il serait aussi possible d'aborder la résolution de problèmes d'optimisation en termes de satisfaction aux buts à deux niveaux [Ruby & Kibler, 1992]. Il s'agirait d'abord de construire des états solutions au problème, puis de choisir, parmi l'ensemble des états solutions, celui qui minimise le critère d'optimalité. Le premier point correspond à la résolution de problèmes. Il est, en ce qui concerne la recherche locale, simplement abandonné, puisque l'un des principes de base de ce domaine veut qu'il soit facile de construire des états solutions. Le deuxième point n'est pas considéré par la résolution de problèmes. De manière plus succincte, on peut dire que la résolution de problème construit des solutions qui sont faisables, alors que l'optimisation cherche la meilleure des solutions faisables.

---

[4] A conditions que a) la solution fasse partie de son espace de recherche, ce qui n'est pas nécessairement le cas, et que b) l'espace de recherche soit exploité de manière systématique et complète.

2.2. L'apprentissage analytique en résolution de problèmes

Des techniques *d'apprentissage analytique* ont été proposées pour permettre aux générateurs de plans d'identifier leurs dysfonctionnements et leurs inefficacités, et de les supprimer. En analysant la trace du générateur de plans, ces techniques construisent des connaissances qui devront permettre au générateur de ne pas répéter ses erreurs et donc d'améliorer ses performances. L'objectif de ces techniques est identique à celui que nous poursuivons dans cette thèse, à la différence près que nous considérons la résolution de problèmes d'optimisation.

Nous allons présenter dans un premier paragraphe, l'une des principales techniques d'apprentissage analytique : l'apprentissage à partir d'explications. Le problème majeur rencontré par toutes les techniques d'apprentissage analytique se situe au niveau de l'utilité des connaissances qu'elles apprennent. Appelé *problème d'utilité*, il sera aussi exposé ci dessous. Si à l'origine, toutes les techniques d'apprentissage analytique s'appliquaient au problème de la génération de plans, certaines d'entre elles ont timidement abordées une sorte de problème d'optimisation à travers le “masking effect”. C'est à ces techniques que nous consacrerons le deuxième paragraphe. Au terme de celui, nous serons en mesure de qualifier l'intérêt qu'il peut y avoir à intégrer telle quelle une technique d'apprentissage analytique dans une technique d'optimisation basée sur la recherche locale.

2.2.1. L'apprentissage à partir d'explications

a) Le principe

L'idée principale, sous-jacente aux approches d'apprentissage analytique, consiste en l'observation du générateur de plans, lorsqu'il résout un problème, et en l'analyse de la trace ainsi obtenue pour identifier ses inefficacités ou ses “raccourcis”. A partir de cette analyse, le système d'apprentissage construit de nouvelles connaissances, qui seront introduites dans le générateur de plans. La performance d'un générateur de plans dépend des connaissances de contrôle dont il dispose pour orienter la recherche. Depuis *Lex* [Mitchell 1983], l'un des premiers systèmes, les techniques d'apprentissage analytique construisent le plus souvent des connaissances de contrôle. Certaines approches cependant engendrent des schémas d'opérateur qui sont des connaissance du domaine.

Unifiant diverses techniques d'apprentissage analytique, Mitchell & al. (1986) et DeJong & Mooney (1986) proposent la technique dite *d'apprentissage à partir d'expli-*

cations (*EBL*), décrite au tableau 1. Cette méthode suppose une démarche en deux temps. En premier lieu, on utilise un générateur de plans, fonctionnant en général en chaînage arrière, afin de construire un plan. Ce plan est appelé *explication* ou, par référence à la démonstration de théorèmes, *arbre de preuve*. Il mène de la description initiale de l'exemple d'apprentissage à un état qui satisfait au but. Sa construction ne requiert en général aucune intervention de l'utilisateur.

Dans la deuxième phase, ce plan est généralisé, souvent par la technique de régression [Nilsson 1980], afin de couvrir un plus grand nombre d'états initiaux. De cette généralisation résulte un schéma d'opérateur qui résume ce plan. Sa partie “condition” contient l'union des conditions des opérateurs employés dans le plan sous-jacent, de la même manière que la partie action contient l'union des actions de ces opérateurs. De cette manière, le plan peut être reconstruit à l'identique, dans tout autre état où la partie “condition” du schéma d'opérateur appris est instanciée, et ceci sans aucun effort de planification. Il en résulte une confiance élevée dans la validité des connaissance apprises.

La technique particulière de Mitchell & al. (1986) ne construit pas des connaissances de contrôle mais des connaissances du domaine ; elle équivaut à *l'évaluation partielle* en démonstration de théorèmes [Van Harmelen & Bundy 1988]. Des approches à peu près similaires ont été proposées pour l'apprentissage de connaissances de contrôle dans *Soar* [Laird & al. 1986], *Prodigy/EBL* [Minton 1989] ou *Static* [Etzioni 1993].

Donné :

- le *but* : le concept à apprendre,
- l'*exemple d'apprentissage* : un exemple du concept but,
- la *théorie du domaine* : un ensemble de schémas d'opérateur et de faits utilisés pour expliquer comment l'exemple atteint le concept but, et
- le *critère d'opérationnalité* : spécification de la forme dans laquelle la règle doit être exprimée.

Résultat :

- Une généralisation de l'exemple qui assure une spécialisation minimale du concept but et qui satisfait au critère d'opérationnalité.

Tableau 1. L'apprentissage par explication, traduit de [Mitchell & al. 1986]

Le rôle du critère d'opérationnalité dans ces méthodes a été longuement débattu. Il recouvre en effet deux caractéristiques distinctes : *l'utilisabilité* et *l'utilité* des connaissances apprises [Keller 1988]. L'utilisabilité assure qu'un nouveau schéma d'opérateur peut être utilisé de manière efficace par le système, tandis que l'utilité rend compte de ses répercussions sur la performance du système entier. L'utilisabilité est typiquement prise en compte par le système par le biais de la rétention des conditions les plus faibles sous lesquelles le plan reste valide (“*weakest preconditions*”) [Minton 1989]. Par contre, l'utilité d'un nouveau schéma d'opérateur n'est pas si facile à affirmer. Nous y consacrerons un paragraphe entier ci-dessous.

Exemple d'apprentissage

F1 `Volume(Obj1,1)`

F2 `Density(Obj1,1)`

F3 `IsA(Obj2,Endtable)`

F4 `Less(1,5)`

Théorie du domaine

O1 `Lighter(x,y)` ⇒ `Safe-To-Stack(x,y)`

O2 `Volume(p1,v1)` ∧ `Density(p1,d1)` ⇒ `Weight(p1,v1*d1)`

O3 `Weight(p1,w1)` ∧ `Weight(p2,w2)` ∧ `Less(w1,w2)` ⇒ `Lighter(p1,p2)`

O4 `IsA(p1,Endtable)` ⇒ `Weight(p1,5)`

Tableau 2. L'exemple `Safe-To-Stack` simplifié de [Mitchell & al. 1986]

Plutôt que de présenter l'un ou l'autre des algorithmes d'apprentissage par explication, il nous parait plus intéressant d'apporter un exemple concret des résultats produits par une telle technique. Considérons l'exemple classique du concept `Safe-To-Stack(x ,y)` [Mitchell & al. 1986], qui indique qu'il est prudent d'empiler un objet `x` sur un autre objet `y`. Le tableau 2 présente un exemple d'apprentissage composé de quatre faits F1 - F4 et d'une théorie du domaine avec quatre schémas d'opérateur O1 - O4. Supposons que le système ait pour but `Safe-To-Stack(Obj1,Obj2)`. Il doit alors expliquer ce but en construisant un plan qui permette d'engendrer le but à partir de la situation décrite dans l'exemple d'apprentissage. Pour ceci, il commence par appliquer O2 instancié sur F1 et F2, ce qui permet d'obtenir `Weight(Obj1,1)`. De même, O4, instancié avec F3, donne lieu à `Weight(Obj2,5)`. Avec ces deux nouveaux faits et F4, l'application de O3 ajoute le fait `Lighter(Obj1,Obj2)`, ce qui à son tour permet d'appliquer O1 et d'engendrer le but.

Ayant construit ce plan, le système démarre la deuxième phase de l'algorithme. L'apprentissage généralise le plan et construit un nouveau schéma d'opérateur qui a comme partie condition l'ensemble des faits de l'exemple d'apprentissage qui étaient employés par le plan. La partie action concerne simplement l'ajout du but. Dans l'exemple, le système crée le schéma d'opérateur :

`Volume(x,v1)` ∧ `Density(x,d1)` ∧ `IsA(y,Endtable)` ∧ `Less(v1*d1,5)`
⇒ `Safe-To-Stack(x,y)` (18)

Considérons maintenant un exemple d'apprentissage de connaissances de contrôle de *Prodigy/EBL* [Minton 1989]. Supposons qu'il y ait trois blocs `A`, `B` et `C`, tous les trois se trouvant par terre (figure 6). Le but est de poser `A` sur `B` et `B` sur `C`. Il se décrit par `Goal(On(A,B))` $\wedge$ `Goal(On(B,C))`. Le générateur de plans possède les schémas d'opérateur nécessaires au déplacement des blocs, tels que `empiler` et `désempiler`. A l'occasion, il commence par résoudre `Goal(On(A,B))` et empile `A` sur `B`. Cependant, en considérant le deuxième but, il ne tarde pas à se rendre compte qu'il doit désempiler `A` de `B` afin de pouvoir empiler `B` sur `C`. Le système d'apprentissage remarque ensuite que le générateur de plans avait mal ordonné les deux buts. Afin d'y remédier, il engendre le schéma d'opérateur de contrôle suivant :

$$\texttt{Goal(On(x,y))} \wedge \texttt{Goal(On(y,z))} \Rightarrow \texttt{Prefer(Goal(On(y,z)),Goal(On(x,y)))} \quad (19)$$

Dans les deux exemples, le schéma d'opérateur construit est ajouté à la théorie du domaine. Ainsi, le générateur de plans peut l'utiliser dès qu'il rencontre une situation qui satisfait à la partie condition. L'avantage est qu'il n'est plus nécessaire, dans le premier exemple, de dériver à nouveau le plan dans son entier. Quant au deuxième exemple, l'inefficacité due au mauvais tri des buts est évitée. Le processus de planification se trouve simplifié dans les deux cas.

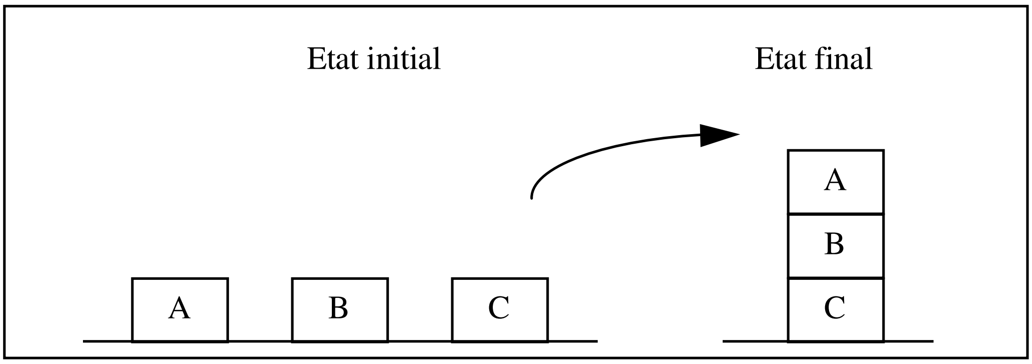


Figure 6. Un exemple du monde des blocs de *Prodigy/EBL*

b) Le problème d'utilité

Les techniques d'apprentissage analytique améliorent les performances d'un générateur de plans en augmentant les connaissances dont celui-ci dispose. Cependant, l'utilisation de ces connaissances entraîne un coût opérationnel. Ce coût supplémentaire peut être plus grand que l'économie due à l'augmentation des compétences du système ; il peut même ralentir son exécution. Ce phénomène est connu sous le nom de *problème d'utilité* [Minton 1990]. Il n'est pas propre aux techniques d'apprentissage par explications ; il apparaît aussi en apprentissage inductif [Holder 1992] et en raisonnement à partir de cas

[Francis & Ram 1993][5]. Il met sévèrement en cause l'intégration de techniques d'apprentissage analytique dans des générateurs de plans.

Deux facteurs sont la cause du problème d'utilité dans le domaine de l'apprentissage par explications [Minton 1990]. D'une part, les connaissances apprises augmentent l'espace de recherche. Lors de la génération de plans, le système doit considérer un nombre plus important d'alternatives. D'autre part, la phase de filtrage devient plus importante, puisque d'une part il y a davantage de schémas d'opérateur à considérer et d'autre part les schémas d'opérateur appris ont souvent des parties conditions complexes. En effet, "*EBL transforme un problème d'exploration d'espace de recherche (…) en un problème de filtrage*" [Minton 1990, p. 369], problème de filtrage qui est NP-complet avec le nombre d'éléments des conditions à filtrer.

Différentes approches permettant de pallier au problème d'utilité ont été conduites selon deux voies orthogonales. D'une part, le système procède à un apprentissage sélectif en ne retenant que les schémas d'opérateur qui sont vraiment intéressants. Dans *Prodigy/EBL*, l'intérêt d'un schéma d'opérateur est évalué de manière expérimentale une fois ce schéma mis à disposition du générateur de plans [Minton 1989], tandis que [Subramanian & Hunter 1992] utilisent un modèle d'estimation des coûts afin d'éviter cette expérimentation coûteuse. D'autre part, le coût de filtrage d'un schéma d'opérateur est fonction de la complexité de sa partie condition. En limitant cette complexité, le filtrage des connaissances apprises devient moins coûteux, et, en conséquence, la dégradation des performances se trouve réduite. Ainsi dans *Soar* l'expressivité du langage de représentation des schémas d'opérateur appris est restreinte [Tambe & al. 1990], tandis que dans *Ariane* [Zerr 1992] les partie conditions des schémas d'opérateur sont spécialisés après leur construction. En outre, une étude théorique sur la complexité de l'apprentissage par explication indique qu'en aucun cas il n'est intéressant d'apprendre des schémas d'opérateur à partir de plans récursifs[6] [Etzioni 1993].

2.2.2. Le "masking effect"

Les connaissances qu'une technique d'apprentissage peut mettre à la disposition d'un générateur de plans peuvent servir trois intérêts orthogonaux : a) elles peuvent permettre

---

[5] En raisonnement à partir de cas, le problème d'utilité est un problème d'importance moindre et essentiellement dû à une dégradation de la vitesse d'interaction avec la mémoire de cas. Nous le traiterons donc dans le chapitre relatif à la conception de la mémoire (cf. p. 97).

[6] La définition d'un *plan récursif* est issue de la démonstration de théorèmes. Ainsi, une démonstration d'un théorème est récursive si elle dérive un littéral à partir d'un autre littéral auquel le premier s'unifie. Par exemple l'utilisation de la clause de Horn `audessus(X,Y)` :– `sur(X,Z), audessus(Z,Y)` dans une démonstration rend celle-ci récursive [Etzioni 1993].

au générateur de plans de construire plus rapidement des plans, c'est à dire d'accélérer le processus de la génération de plans ; b) les plans peuvent être de meilleure qualité, et ceci indépendamment de l'effort réalisé pour le construire ; et c) les plans peuvent conduire à des états finaux qui sont eux de meilleure qualité. Indépendant des deux premiers points, ce troisième élément n'est pas classique en génération de plans.

a) Accélération de la construction de plans

La figure 7 illustre ces trois cas. La situation concerne un agent *A* qui doit générer un plan permettant d'atteindre une destination finale (non-indiquée). Cependant, il ne doit pas être rattrapé par l'agent *B*. Dès qu'il aperçoit l'agent *B*, l'agent *A* cherche à se cacher derrière un mur. Pour se faire, il a besoin de construire un nouveau plan. La figure 7.a) présente une première situation où *A* aperçoit *B*. *A* construit donc un plan qui le mène en un lieu sûr P_1, derrière un mur indiqué en horizontal. Ce plan étant un succès, la technique d'apprentissage dont dispose *A* construit les connaissances nécessaires pour pouvoir le reproduire.

Survient alors la situation représentée dans la figure 7.b) et qui ressemble beaucoup à la situation précédente. En apercevant *B*, l'agent *A* doit construire un nouveau plan pour se cacher. Grâce à l'apprentissage précédent, il peut reconstruire le plan utilisé dans la figure 7.a) et se cache à P_2. Ce plan étant correct ainsi qu'immédiatement disponible[7], l'apprentissage a rendu la génération de plan très efficace.

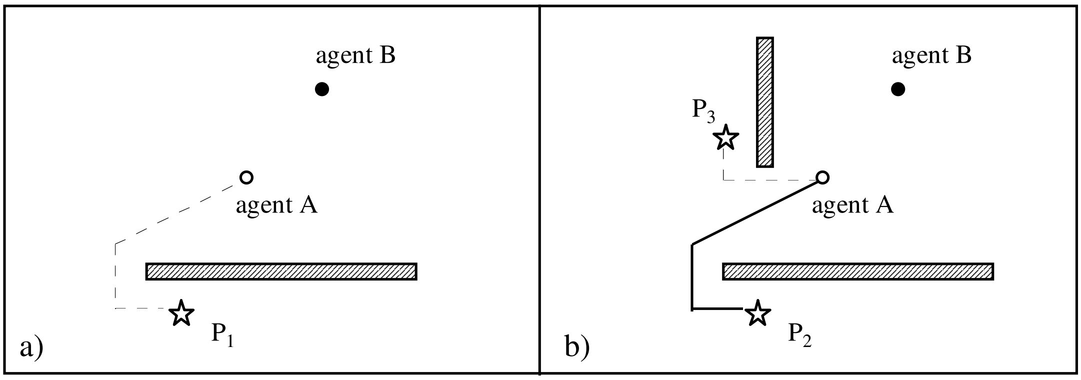


Figure 7. Le “masking effect”, d'après [Tambe & Rosenbloom 1993]

b) Amélioration des plans

Malgré tout, le plan menant à P_2 est d'une qualité médiocre, puisqu'il ne permets pas d'identifier le meilleur abri. Celui-ci se trouve en effet à P_3 puisqu'il est plus proche de *A*

---

[7] Nous ignorons pour l'instant le problème d'utilité.

que P_2. Le problème est que les connaissances apprises, en permettant la construction efficace d'un plan, sont réutilisées de préférence. Elles occultent donc des plans qui pourraient s'avérer meilleurs que les plans appris puisqu'elles empêchent le générateur de plans d'envisager d'autres alternatives de plans. Ce phénomène, très général, est appelé "*masking effect*" [Clark & Holte 1992].

Considérons l'exemple `Safe-To-Stack`. Le schéma d'opérateur appris (18) est utilisé de préférence aux autres schémas de la théorie du domaine. Dans une situation où il s'agit de démontrer `Safe-To-Stack(x1,y1)`, et où la base de faits contient déjà le littéral `Lighter(x1,y1)`, le générateur de plans utilise ce schéma d'opérateur appris, et ne remarque pas que le schéma d'opérateur O1 permettrait directement d'engendrer le concept recherché. Or, O1 est a priori plus simple que le schéma appris, puisqu'il nécessite l'évaluation de moins de conditions.

L'ensemble des systèmes d'apprentissage analytique cités ci-dessus souffrent de ce phénomène. Ils se limitent tous à l'acquisition de connaissances correctes dans le seul but d'accélérer le processus de génération de plans lui-même. Ils ne connaissent pas la notion de qualité d'un plan. Les techniques prenant en compte la qualité de plans sont plus récentes [Clark & Holte 1992, Tambe & Rosenbloom 1993, Pérez 1996], et seule Pérez (1996) l'aborde effectivement sous la forme d'un problème d'optimisation.

c) Amélioration de la qualité de l'état final des plans

Cependant, même ces dernières méthodes [Clark & Holte 1992, Tambe & Rosenbloom 1993, Pérez 1996] rencontrent des difficultés lorsque l'état final du plan présente un intérêt en soi. Dans notre exemple, l'agent *A* a pour objectif non seulement d'échapper à l'agent *B*, mais également d'atteindre au plus vite sa destination finale. Il est alors important que, lorsqu'il doit se cacher, ce lieux sûr soit le plus proche possible de cette destination finale. Il y a de la sorte une notion de qualité qui est directement attachée à l'abri et qui est relative à la distance qu'il y a entre l'abri et la destination finale de l'agent.

Ainsi, il ne suffit pas de produire, au plus vite, un plan permettant à *A* de se cacher, ni de construire un plan menant vers l'abri le plus proche, mais il faut de préférence élaborer un plan menant vers un abri qui est le plus proche possible de la destination finale de *A*. Le générateur de plans doit en conséquence être capable de choisir son plan selon ces trois critères : efficacité de génération, qualité du plan *et* qualité de l'état final du plan. En conséquence, la technique d'apprentissage devrait assister le générateur de plans sur ces trois points, pour vraiment être intéressante. A notre connaissance, il n'existe pas de telle technique d'apprentissage analytique.

2.2.3. L'apprentissage analytique en optimisation

A l'issue du premier chapitre ainsi que de cette première section du deuxième chapitre, les similitudes et les différences entre d'une part les techniques d'optimisation basées sur la recherche locale et d'autre part la génération de plans sont établies. Il s'agit à présent de considérer l'intérêt de l'intégration d'une technique d'apprentissage analytique dans une méthode de recherche locale. L'objectif d'un générateur de plans est de créer un plan qui mène vers un état satisfaisant au but. En recherche locale, la problématique se présente de manière différente. La satisfaction du but en tant que telle n'est pas aisément vérifiable. En ce sens, on peut dire que les problèmes abordés par la recherche locale ne sont pas *bien-définis*.

Les techniques d'apprentissage par explication ont pour objectif d'accélérer le processus même de la génération de plans. Pour ce faire, elles analysent et généralisent des plans exemples. En recherche locale la construction de plans exemples elle-même est mise en cause, puisque la vérification de la satisfaction au but est extrêmement difficile. Si l'on ne sait pas si un plan mène à un état satisfaisant au but, existe-t-il alors une raison d'apprendre ce plan ? Nous en doutons.

En outre, une autre raison permet de douter de l'intérêt que pourrait revêtir l'apprentissage de plans en recherche locale. En effet, c'est de la qualité de l'état final, mesurée par la fonction objectif, que se préoccupe une technique de recherche locale ; or cette qualité n'est absolument pas prise en compte par les techniques qui apprennent des plans. Mais si l'on ne connaît pas la qualité de l'état final d'un plan, le motif même justifiant la réutilisation d'un plan n'est plus établi.

De ce qui précède, nous concluons qu'il serait peu astucieux d'intégrer directement une technique d'apprentissage analytique, conçue pour la génération de plans, dans une méthode de recherche locale. En effet, les conditions pour rendre un telle méthode intéressante ne seraient pas réunies. Ceci étant, nous allons à présent nous consacrer à une technique d'apprentissage qui, comme nous expliquerons, nous parait être intéressant pour l'apprentissage des connaissances de contrôle pour le guidage de la méthode tabou : le raisonnement à partir de cas.

2.3. Raisonnement à partir de cas

Le raisonnement à partir de cas est une méthode de résolution de problèmes qui emploie l'expérience comme principale source de connaissances dans le processus de construction de solutions. Ainsi, un système de raisonnement à partir de cas résout un nouveau problème en réutilisant la solution d'un problème similaire antérieur. Il se distingue donc des systèmes à base de règles par cette incorporation explicite de l'expérience dans le processus de résolution de problèmes. Parmi les domaines d'application des systèmes de raisonnement à partir de cas figurent la génération de plans, la catégorisation, le diagnostic et la conception. Toutefois, son application à la résolution de problèmes d'optimisation est encore assez rare.

Le but de cette section est de donner un aperçu de cette technique d'intelligence artificielle. Le nombre de systèmes qui emploient des cas est fort élevé, une présentation assez complète se trouve dans [Kolodner 1993, Marir & Watson 1994]. Pour notre part, après une brève introduction, nous considérons les domaines d'application majeurs à travers quelques systèmes importants. Enfin, nous présenterons différents systèmes qui se proposent de résoudre des problèmes d'optimisation.

2.3.1. Le principe du raisonnement à partir de cas

Historiquement, le *raisonnement à partir de cas* a été développé à la fin des années 70 par Schank et Kolodner. Dès l'origine, il se désignait tout autant comme modèle cognitif que comme principe de résolution de problèmes [Schank 1982, Ganascia 1990]. Dans cette thèse nous n'allons nous préoccuper que de très loin de l'aspect cognitif. L'emploi que nous en faisons dans le système *Alois* nous paraît expliquer cette distance.

a) Le cycle opérationnel

Le principe de base du raisonnement à partir de cas est que des problèmes similaires ont des solutions similaires [Leake 1996]. Le comportement opérationnel d'un système générique basé sur le raisonnement à partir de cas peut être schématisé par un cycle composé de quatre étapes (tableau 3) [Aamodt & Plaza 1994]. Confronté à un nouveau problème[1], appelé *problème actuel* ou *problème cible*, le système cherche dans sa mé-

---

[1] Par définition, le problème à résoudre fait partie du même domaine que les problèmes qui se trouvent dans la mémoire de cas. S'il n'en va pas ainsi, on parle de raisonnement par analogie.

moire le cas qui contient le problème qui est le plus ressemblant au problème cible. Ce problème ainsi récupéré est appelé le *problème source*. Celui-ci est utilisé pour résoudre le problème cible, par exemple en adaptant la solution du problème source de façon à ce qu'elle résolve le problème cible. Il se peut toutefois que cette solution adaptée n'ait pas la qualité souhaitée. Afin de détecter d'hypothétiques défaillances et d'y remédier, elle peut être révisée. Enfin, si la nouvelle solution trouvée est originale, elle peut être utilisée pour enrichir la mémoire de cas, en y étant incorporée. Ainsi le système accroît son expérience.

1. RECUPERE le cas le plus proche dans la mémoire de cas,
2. REUTILISE ce cas pour essayer de résoudre le problème,
3. REVISE la solution proposée,
4. RETIENS la nouvelle solution comme composant d'un cas.

Tableau 3. Le cycle opérationnel du raisonnement à partir de cas, traduit de [Aamodt & Plaza 1994]

L'intérêt majeur du raisonnement à partir de cas réside donc dans sa capacité à résoudre des problèmes à partir d'anciennes solutions, sans devoir recommencer le processus de construction de solutions dès le début. En fait, il peut s'avérer plus facile de transformer des solutions anciennes que d'en construire de nouvelles[2] [Leake 1996]. Deux cadres se prêtent à l'utilisation du raisonnement à partir de cas :

1. Le développement de systèmes à base de connaissances requiert l'identification préalable d'un modèle [Wielinga & al. 1992]. Toutefois, il y a des domaines où, s'il existe, ce modèle est très difficile à déterminer. Dans ces domaines, le raisonnement suivi par les experts peut lui-même être fortement casuel, accordant une place primordiale à l'expérience. Constituent des exemples de ce type de domaine : le raisonnement juridique dans les pays de Common Law, où les jugements sont pris sur la base d'anciens jugements concernant des affaires similaires [Ashley 1988] ; le chargement d'un autoclave [Barletta & Hennessy 1989] ; la modélisation d'objets pour faciliter la résolution d'équations différentielles partielles par la méthode d'éléments finis [Hurley 1995] ; ou encore la prévision de l'évolution temporelle et spatiale de feux de forêts [Rougegrez-Loriette 1994].

---

[2] En ce qui concerne la génération de plans, Hanks & Weld (1995) donnent une estimation de la ressemblance minimale entre le problème original et le problème à résoudre afin que la réutilisation soit intéressante.

2. La construction de systèmes à base de connaissances requiert au préalable l'élicitation des connaissances d'un ou de plusieurs experts. Cependant, ce processus est connu pour être difficile. En outre, même une fois l'acquisition des connaissances achevée et le système expert construit, la maintenance de la base des connaissances pose elle-même un problème non négligeable. La représentation des connaissances expert sous forme de cas peut alléger ces deux tâches [Aamodt 1995]. Dans le cas de figure de l'emploi d'un système de raisonnement à partir de cas, l'élicitation de connaissances revient à collectionner des anciennes solutions fournies par l'expert, tandis que la maintenance du système consiste en la validation des cas se trouvant dans la mémoire, en procédant à un ajout ou à une suppression de cas si besoin est [Watson & Marir 1994].

Il convient de regarder ces deux applications d'un oeil critique. Elles proposent, en effet, le raisonnement à partir de cas comme solution simple et élégante à deux problèmes jusqu'alors considérés comme étant fort difficiles. Toutefois, dans la pratique les choses sont plus complexes. Ces deux applications reposent sur une représentation de l'expérience sous forme de connaissances casuelles. Elles négligent cependant les connaissances auxiliaires dont les systèmes ont besoin pour la manipulation des cas. Souvent, ces connaissances sont tout aussi difficiles à acquérir et à maintenir.

Pour illustrer l'importance des connaissances non-casuelles considérons un *problème de conception* tel qu'on en rencontre en architecture. Le raisonnement à partir de cas est considéré comme bien adapté aux problèmes de conception, puisqu'il n'existe en général pas de théorie formelle [Hindrichs 1988]. Dans le domaine de l'architecture, l'hypothèse sous-jacente est que les architectes se réfèrent largement à des constructions antérieures, les leurs ou celles de leurs confrères, lors de la conception de nouveaux projets [Pearce & al. 1992]. Une étude de cas, portant sur le type de connaissances réellement utilisé par des architectes [Strube & al. 1995], montre que les cas ont un rôle prépondérant dans l'orientation du raisonnement des experts, ici des architectes. Toutefois, les architectes n'emploient que très peu de cas. En effet, pas plus de 10% des connaissances employées n'étaient de nature casuelle, pour le reste il s'agissait essentiellement de règles et de normes. Strube & al. (1995) concluent que "… *cases do not have the significance which CBR theory [e.g. Kolodner 1993] suggests*" (p. 551)[3]. Ces résultats confirment tout à fait les assertions plus intuitives de Ganascia (1990).

Il est bien évidemment difficile de projeter les résultats d'une étude cognitive sur des systèmes opérationnels si ces systèmes ne sont pas directement des implantations

---

[3] Traduction : "… les cas n'ont pas l'importance que la théorie du raisonnement à partir de cas leur attribue."

d'un modèle cognitif. En raisonnement à partir de cas cependant, la frontière entre l'implantation d'un modèle cognitif et une technique de résolution de problème est souvent floue. En effet, un bon nombre de systèmes de raisonnement à partir de cas sont eux-mêmes des implantations de modèles cognitifs [Kolodner 1993]. L'affirmation de [Strube & al. 1995] ne peut donc pas non plus être simplement ignorée.

Le joueur de basse de Ramalho (1996) nous parait révélateur de l'importance que peuvent prendre ces connaissances non-casuelles dans un système qui, explicitement, n'est pas l'implantation d'un modèle cognitif. Ce système n'emploie pas moins de cinq différents systèmes à base de règles, alliés à une technique de raisonnement à partir de cas, la mémoire musicale [Ramalho & Ganascia 1994], afin de simuler les lignes de basse produites par un joueur de jazz qui accompagne un soliste. Cependant, ce sont les cas qui sont pour une bonne partie responsables de la performance intéressante du système.

Considérons maintenant les composants des systèmes de raisonnement à partir de cas. Ceci nous permettra de mettre en évidence différents endroits où, de manière implicite ou explicite, des connaissances auxiliaires, non casuelles, peuvent être introduites.

b) Les cas

L'originalité du raisonnement à partir de cas se situe, entre autre, au niveau de l'utilisation de *connaissances casuelles* pour la résolution de problèmes. Ce sont les *cas* qui représentent ces connaissances par extension. Ces connaissances sont en général de nature opérationnelle, décrivant des actions en étroite liaison avec leurs contextes [Kolodner 1993]. Ceci les oppose aux règles, telles qu'employées dans des systèmes à base de règles.

Un cas peut être composé de divers éléments, selon les besoins du système. Il existe trois composants majeurs (tableau 4). Le cas contient une *description du problème* auquel il s'applique, et la *description de sa solution*. Avec ces deux éléments de base, il est possible de résoudre des problèmes par adaptation et réparation. En ce qui concerne la description du problème, il est important qu'elle englobe le plus possible d'éléments caractéristiques du problème à décrire. Extraire de l'état du système cette description est appelé *mise en forme* [Py 1994] ("*situation assessment*"). Ce point important, où interviennent typiquement des connaissances extérieures au système, n'a reçu jusqu'ici que très peu d'attention de la recherche sur le raisonnement à partir de cas.

Il est possible que la description du problème ne suffise pas en tant que telle à justifier le recours à la solution contenue dans le cas. C'est, par exemple, possible quand la description du problème est partielle ou contient des inconnues. La réutilisation de tels cas pour résoudre le problème cible peut alors mener à des solutions qui sont de qualité différente de celle de la solution inscrite dans le cas. Le troisième élément des cas, la *ré-*

tribution, est alors utilisé afin d'éviter que des récupérations non appropriées se répètent, et donc qu'une réutilisation peu intéressante se renouvelle.

- Description du problème ou de la situation,
- Solution ou action,
- Rétribution

Tableau 4. Structure générique des cas

Souvent, les cas comportent, outre ces trois éléments, des *index*. Les index servent à décrire les attributs caractéristiques et discriminatoires des cas. Leur but est d'indiquer les conditions dans lesquelles un cas peut être utile. Lors de la récupération, les index sont employés de manière à *prédire* l'utilité d'un cas dans la résolution du problème cible. L'attribution d'index est en général automatique, et se réalise par rapport à des dimensions qui elles, soit sont sélectionnées par le concepteur du système, tel que dans *Chef* [Hammond 1986], soit sont sélectionnées par le système en utilisant les techniques d'apprentissage par explication (cf. p. 50), tel que dans *Caber* [Barletta & Mark 1988]. Les index soit sont simplement extraits de la description du cas, et sont donc des attributs de surface, soit sont des connaissances du domaine employées afin de les construire à partir de cette description.

c) La mémoire de cas

La mémoire de cas remplit des fonctions importantes à l'intérieur d'un système de raisonnement à partir de cas. C'est elle qui met à disposition du processus d'adaptation la matière première, les cas. Ainsi, lors de la *récupération*, elle cherche, parmi l'ensemble des cas dont elle dispose, celui, ou ceux, qui sont le plus similaires au problème cible. De même, s'il y a lieu de *mémoriser* de nouveaux cas, c'est elle qui les retient de façon à faciliter les récupérations ultérieures.

Deux formes *d'organisation de la mémoire de cas* ont été développées, l'une basée sur les techniques des plus proches voisins, l'autre sur l'indexation selon les attributs communs [Porter 1989]. Nous reviendrons sur la mémoire de cas ainsi que sur son lien avec les mesures de similarité dans le chapitre 4, où nous introduirons aussi notre organisation de la mémoire qui emploie des arbres de prototypes dynamiques (cf. p. 101).

d) Adaptation

La récupération identifie le (ou les) cas le plus approprié pour la résolution du problème. *L'adaptation* est le processus qui ensuite transforme la solution contenue dans le cas de manière à ce qu'elle résolve le problème cible (tableau 5).

Entrée :

- La description du problème
- Une solution pas-tout-à-fait-correcte
- La description du problème que cette solution résout (facultatif)

Sortie :

- Une solution qui résout le problème

Méthode :

- Adapte la solution pas-tout-à-fait-correcte afin qu'elle corresponde à la description du problème

Tableau 5. Adaptation, traduit de [Kolodner 1993]

En fonction du contenu des cas, il existe deux types de procédure d'adaptation. Soit les cas contiennent des descriptions de solutions, et l'adaptation consiste alors à réutiliser directement ces solutions, soit les cas contiennent des traces du processus de construction de solutions. Dans ce dernier cas, l'adaptation reconstruit une nouvelle solution depuis le début, mais elle se laisse guider par la trace fournie par le cas [Veloso 1994].

Il existe un large éventail de méthodes relatives à l'adaptation. Elles se divisent en trois grandes classes [Kolodner 1993]. Tout d'abord, les méthodes de substitution qui remplacent des sous-parties des cas par d'autres descriptions. Viennent ensuite les méthode d'adaptation qui emploient des techniques de transformation de solutions, plus élaborées, telles que la résolution de problèmes ou même le raisonnement à partir de cas [Leake & al. 1995]. En dernier lieu, il y a les méthodes directement conçues pour un domaine d'application et qui sont généralement propre au système. La mise en oeuvre d'une méthode d'adaptation requiert souvent des connaissances du domaine. En outre, le système doit par lui-même, ou guidé par l'utilisateur, décider de la partie du cas qui doit être adaptée, ainsi que de la méthode qui y est la plus appropriée. Les techniques relatives à ce point sont variées et se basent typiquement sur des heuristiques propres au domaine d'application du système.

2.3.2. Les domaines d'application

Dans ce paragraphe, nous ne traiterons que des domaines d'application principaux du raisonnement à partir de cas. Il s'agit de la résolution de problèmes, de la catégorisation, du diagnostic et de la conception. Nous évoquerons également la nouvelle approche qui se propose d'intégrer des bases de données industrielles, en tant que sources fertiles de cas, donnant lieu à la création de systèmes à base de connaissances à l'échelle de toute une entreprise. Dans le paragraphe suivant, nous présenterons les systèmes de raisonnement à partir de cas qui s'appliquent à des problèmes d'optimisation.

a) La génération de plans

Un problème de *génération de plans*, ou de *résolution de problèmes*, est défini par un ou plusieurs buts, un état initial et un ensemble d'opérateurs. La tâche consiste à construire un enchaînement d'opérateurs, appelé *plan*, qui mène d'un état initial à un état final qui satisfait aux buts (cf. p. 44). L'accélération de ce processus de construction de plans à fait l'objet de l'apprentissage par explication (cf. p. 50) et du raisonnement à partir de cas. Les deux approches ont pour but de guider le processus de construction de plans de manière à ce qu'il rencontre le moins d'échecs possibles.

Un des premiers systèmes de raisonnement à partir de cas, dédié à la génération de plans est *Chef* [Hammond 1986]. Il a pour domaine l'élaboration de recettes de cuisine. Les buts décrivent les caractéristiques auxquelles un repas doit satisfaire, et ses opérateurs sont des actions de cuisson. Les cas dans *Chef* décrivent des recettes, c'est à dire des plans complets. Pour construire une nouvelle recette, *Chef* cherche dans sa mémoire de cas une recette ayant des caractéristiques similaires, c'est à dire des plans avec les mêmes buts. Si les caractéristiques de la recette trouvée ne sont pas identiques à celles de la recette à l'étude, il modifie la recette trouvée afin de combler ces différences. *Chef* emploie des connaissances du domaine, appelé *critiques*, afin de modifier la recette.

Une fois ces différences éliminées et la recette construite, *Chef* simule son exécution. Si elle s'avère correcte, elle est retenue comme nouveau cas. Sinon, elle est *réparée*. Pour cette réparation, *Chef* reconnaît différents types d'échecs ainsi que des stratégies de réparations. Ces recettes “avec échec” sont elles aussi ajoutées à la mémoire de cas, ainsi que leurs plans de réparation. Lors de la construction ultérieure de plans, *Chef* peut anticiper ses erreurs et ainsi éviter de les répéter.

Le générateur de plans *Plexus* [Alterman 1986] tend aussi à réutiliser d'anciens plans dans de nouvelles situations. Il se distingue de *Chef* en ce qu'il s'intéresse plus aux processus d'adaptation de plans, et moins à leur réparation. Lors de l'adaptation, *Plexus* utilise des réseaux sémantiques afin d'abstraire et de spécialiser des actions du plan source jusqu'à ce que celui-ci devienne correct pour la situation cible.

Le système *Spa* [Hanks & Weld 1995] élargit les approches de *Chef* et de *Plexus* car il considère les plans comme faisant partie d'un espace de plans. Dans cet espace de plans, des transformations permettent de passer d'un plan à un autre, et le problème de génération de plans devient alors un problème de recherche dans un espace d'états [Sacerdoti 1977]. Les cas fournissent alors des points de départ pour accélérer cette recherche. Dans cette optique, il y a une forte ressemblance entre la construction de plans et l'adaptation de plans ; la construction correspond simplement à l'adaptation d'un plan vide.

Le système *Prodigy/Analogy* emploie une stratégie bien différente des trois approches précédentes [Veloso 1994]. Ici, les cas ne contiennent pas des plans, mais des traces annotées d'anciens épisodes de génération de plans. Chaque cas contient des informations sur les différentes décisions qui ont été prises, en les qualifiant de bonnes ou de mauvaises. Les cas sont indexés par les buts auxquelles les plans s'adressent [Muñoz-Avila & Huellen 1995], et la réutilisation a pour objectif de contrôler les futures phases de construction de plans par les annotations contenues dans les traces de résolution. Les cas apportent ainsi des connaissances de contrôle.

b) La catégorisation

La *catégorisation* traite de l'affectation de classes préétablies à des exemples[4]. Domaine classique de l'apprentissage symbolique, ce type de problèmes a aussi fait l'objet de développements en raisonnement à partir de cas. L'approche étudiée consiste à identifier, parmi un ensemble de cas, celui qui est le plus similaire de l'objet à classer [Kolodner 1993]. Il existe deux sortes de techniques. La première procède à une simple recherche dans une liste de cas. Appelée *apprentissage à partir d'instances* (“*instance-based learning*”), elle appartient à la catégorie plus générale des algorithmes des plus proches voisins. Nous en traiterons au chapitre 4.1 (cf. p. 109).

La deuxième approche consiste en une utilisation plus intense des cas. Appelée *catégorisation à partir d'exemplaires*, son représentant principal est *Protos* [Porter & al. 1990]. *Protos* exécute la catégorisation en deux phases. Tout d'abord, à partir de la description de l'objet à classer, *Protos* avance une première hypothèse sur la classe de l'objet. Il choisit alors dans sa mémoire un cas exemplaire de cette catégorie et établit la similarité de celui-ci avec l'objet à l'étude. Si cette similarité s'avère suffisamment élevée, l'hypothèse est acceptée. Dans le cas contraire, *Protos* examine, durant la deuxième phase, plus profondément les différences qui existent entre les représentations des deux objets. S'il découvre un autre cas, lié au cas sélectionné, qui présente des différences comparables avec celles constatées, *Protos* sélectionne ce cas comme nouvelle hypothèse, et ré-établit la similarité avec l'objet à classer.

D'autres techniques de catégorisation à partir d'exemplaires utilisent une méthode qui ressemble à celle de *Protos*. Procédant en deux étapes, ils utilisent d'abord une

---

[4] Nous utilisons le terme *catégorisation* afin de désigner l'opération qui consiste à attribuer une classe pré-définie à des exemples. Aussi appelé *apprentissage supervisée* ou *discrimination*, ce terme correspond au terme anglais de “*classification*”.

technique de catégorisation classique. Lorsque celle-ci produit un résultat auquel il n'est pas fait confiance, une deuxième phase est enchaînée. Cette deuxième phase emploie des cas comme connaissances opérationnelles de catégorisation. D'autres techniques de catégorisation entreprises lors de la première étape sont par exemple C4.5 [Surma & Vanhoof 1995] ou les réseaux de neurones [Malek 1995].

c) La conception

Un *problème de conception* est souvent décrit par un ensemble de contraintes que toute solution doit satisfaire. Cependant, il y a une différence importante entre les problèmes de conception et les *problèmes de satisfaction de contraintes* [Van Hentenryck 1989]. Si les deux problèmes rattachent la faisabilité d'une solution à sa conformité avec un ensemble de contraintes, dans les problèmes de conception, les contraintes ne spécifient pas comment la solution peut être construite. Ainsi, la contrainte imposant à un pont de supporter une certaine charge n'indique pas le moyen qui pourrait faciliter la construction de ce pont.

Les contraintes dans les problèmes de conception peuvent être de deux types. Pour le *problème sous-contraint* le nombre de solutions est vaste, et il s'agit en conséquence d'en trouver une bonne[5]. S'agissant d'un *problème sur-contraint* il n'existe de solutions qu'une fois quelques contraintes relâchées. Le système doit alors décider quelles contraintes il veut satisfaire, et lesquelles il peut abandonner [Kolodner 1993]. La plupart des problèmes de conception présentent ces deux caractéristiques de manière simultanées.

La manipulation et la sélection des contraintes constituent en général un problème délicat puisque d'elles, et de leur interaction, dépend en grande partie la solution. Les systèmes de raisonnement à partir de cas utilisent donc des cas qui indiquent comment différentes contraintes ont été manipulées conjointement. Cependant, les solutions aux problèmes de conception sont en général de trop grande taille pour qu'un seul cas suffise à fournir une solution de départ valable. Ainsi, les systèmes doivent combiner plusieurs cas entre eux. En fonction du type de combinaison, on distingue deux sortes de systèmes. Les uns, tels que *Clavier* [Barletta & Hennessy 1989] ou *Mediator* [Kolodner & Simpson 1989], utilisent un cas comme fil rouge pour la résolution du problème de conception, et plusieurs autres cas afin de remplir les trous laissés par le premier. Les autres systèmes, tels que *Julia* [Hindrichs 1988] ou le *générateur d'itiné-*

---

[5] Toutefois, les problèmes de conception n'ont, en général, pas de fonction objectif permettant d'évaluer la qualité d'une solution. Il s'agit plutôt de préférences entre les différentes contraintes qui peuvent être satisfaites.

raires [Haigh & Veloso 1995], procèdent par décomposition du problème en sous-problèmes, sans passer par un cas “fil rouge”. Des cas sont récupérés pour résoudre les sous-problèmes, et leurs solutions sont assemblées pour résoudre le problème original.

d) Le diagnostic

Dans un *problème de diagnostic* il s'agit de construire une explication pour un phénomène qui est en général décrit par un ensemble de symptômes. Souvent, il existe un modèle causal du domaine qui par un raisonnement abductif permet de construire une explication du phénomène [Console & Torasso 1992]. Cependant, la construction d'une explication peut être un processus complexe, et il existe en général plusieurs explications possibles pour un seul phénomène, parmi lesquelles le système doit choisir la plus appropriée. Les systèmes de diagnostic utilisant le raisonnement à partir de cas intègrent souvent le modèle causal.

Ainsi, le système *Casey* [Koton 1988] explore la réutilisation d'anciens diagnostiques pour expliquer de nouveaux phénomènes. Il opère dans le domaine médical des problèmes cardiaques et utilise un modèle causal très complet pour générer une base de cas. Face à un nouveau patient, *Casey* identifie dans sa mémoire de cas le patient qui présente des symptômes similaires. Il réalise ensuite une première mise en correspondance des symptômes des deux patients, et procède à des transformations de cette correspondance jusqu'à ce que toutes les différences soient éliminées. Avec une perte de compétence très limitée, *Casey* est beaucoup plus rapide que le programme qui utilise directement le modèle causal.

e) Les bases de cas industrielles

Récemment, différents systèmes de raisonnement à partir de cas ont commencé à intégrer des sources de cas nouvelles, à savoir les *bases de données* des entreprises. En effet, un bon nombre d'entreprises possèdent aujourd'hui des bases données portant sur une large partie de leurs activités. Contenant des informations fortement opérationnelles, ces bases de données peuvent ainsi englober des connaissances sur “la vie” de l'entreprise que l'entreprise elle-même ignore. En effet, le domaine de la découverte de connaissances dans les bases de données est aujourd'hui très prometteur [Brachman & Anand 1994].

Intégrer, dans des systèmes de raisonnement à partir de cas, des bases de données industrielles, en tant que fournisseurs de cas, pose un certain nombre de problèmes. Sans parler de la modélisation des connaissances nécessaires pour les applications envisagées, la particularité d'une telle intégration se situe au niveau de l'interaction du système à base de cas avec les bases de données. En effet, ces dernières sont déjà opérationnelles, et en général extrêmement larges. En conséquences, elles ne seront que difficilement modifiées pour accueillir un module supplémentaire d'une importance secondaire. En outre, se pose le problème technique du caractère reparti et hétérogène des bases de données d'aujourd'hui.

La recherche portant sur une intégration du raisonnement à partir de cas dans les bases de données industrielles est encore jeune. Les problèmes clés sont la représentation invariante des cas au travers des divers systèmes de bases de données [Brown & al. 1995], la récupération de cas dans des bases de données qui ne permettent pas une évaluation de la similarité pendant la récupération, mais seulement en avant ou en aval [Shimazu & al. 1993], ou encore l'utilisation de requêtes SQL floues ou partielles [Daengdej & al. 1996].

2.3.3. Les systèmes d'optimisation à partir de cas

Différents systèmes de raisonnement à partir de cas s'attachent à la résolution de *problèmes d'optimisation*. Rappelons qu'un problème d'optimisation est décrit par une région de faisabilité et par une fonction objectif. La solution recherchée est celle qui, sur la région de faisabilité, maximise (ou minimise) la fonction objectif (cf. p. 18).

Ces systèmes ont suivi deux approches pour la résolution de problèmes d'optimisation. Dans la première, le problème d'optimisation n'est pas considéré en tant que problème d'optimisation. Typiquement, l'espace de faisabilité ou la fonction objectif ne sont pas donnés explicitement au système, mais sont représentés dans les cas. Ainsi, la résolution ne se focalise pas sur l'identification de la solution optimale, mais s'attache au processus de construction de solutions. L'hypothèse de base de ce type d'approche est que la bonne qualité des solutions, qui ont donné lieu à l'élaboration de cas, survit au processus d'adaptation.

L'exemple d'un tel système est *Clavier* [Barletta & Hennessy 1989]. Il résout un problème d'optimisation du type sac à dos à deux dimensions, qui consiste à placer un nombre maximal de pièces dans un récipient à capacité réduite. Cependant, les conditions d'incompatibilité entre les objets à placer ne sont pas aisément formalisées. L'expert qui charge la machine les acquiert par expérience. Les cas de *Clavier* représentent alors des chargements du récipient qu'un expert considère bons et valides. Ensuite, *Clavier* construit de nouveaux chargements à partir de ces cas.

Un autre exemple d'un tel système est le *générateur d'itinéraires* de [Haigh & Veloso 1995]. Le problème d'optimisation auquel il s'attache est celui du chemin minimal entre deux noeuds dans un graphe. Le système dispose d'une carte géographique et de règles permettant de construire des trajets à partir de la carte. Par ailleurs il connaît, sous forme de cas, un ensemble d'anciens itinéraires qui ont été exécutés et qui en conséquence sont annotés. Lors de la construction d'un parcours menant d'un point A vers un point B, le système réutilise le plus possible ses cas. En consultant les annotations des cas, le système peut alors éviter des endroits connus pour être peu fluides ou en travaux. La fonction objectif n'est pas donné au système, c'est seulement au travers des cas qu'il parvient à construire des solutions de “bonne” qualité.

La deuxième approche de la résolution de problèmes d'optimisation par le biais des systèmes de raisonnement à partir de cas s'intéresse directement au problème d'optimisation. L'ensemble de ces systèmes *d'optimisation à partir de cas* utilisent des techniques d'optimisation heuristiques basées sur la recherche locale (cf. p. 25). Nous présenterons trois de ces systèmes.

a) Le système Ease

Le système *Ease* [Ruby & Kibler 1992] a pour but d'optimiser *a posteriori* des solutions à des problèmes de conception. Présenté dans le domaine de la construction de circuits logiques, il fait appel à un système classique de construction de circuits pour l'élaboration d'une première solution qui remplit les spécifications du problème posé. *Ease* tente alors de transformer cette première solution afin de l'optimiser par rapport à un certain critère, ici le délai sur le chemin critique. Pour conduire cette optimisation, il dispose de cas représentant d'anciennes transformations avantageuses. Confronté à une première solution, *Ease* recherche dans la mémoire l'ensemble des cas qui contiennent des transformations possibles. Il applique ces transformations, l'une d'après l'autre, selon le gain attendu de chacune d'elles.

Ease acquiert sa base de cas pendant une phase d'entraînement préliminaire. Cet entraînement porte sur des instances de problèmes plus simples que celles qu'il devra résoudre par la suite. Ici se trouve le point délicat de la méthode. En effet, *Ease* construit de manière exhaustive l'ensemble des instances de problèmes simples (impliquant une ou deux transformations seulement) pour l'ensemble des transformations dont il dispose. Le nombre des instances ainsi créées devient rapidement important, et l'entraînement est en conséquence assez lourd. Toutefois celui-ci s'avère inévitable si l'on veut construire une base de cas contenant des éléments représentatifs. Le point positif de cette technique est que seulement très peu de cas sont effectivement retenus, peu compte tenu du nombre de problèmes d'entraînement construits, la plupart des problèmes étant inutiles. Les auteurs ne donnent cependant aucune indication permettant de réduire le nombre d'instances de problèmes d'entraînement.

b) Le système Cabins

Tout comme *Ease*, le système *Cabins* [Sycara & Miyashita 1994, Miyashita & Sycara 1995] tend à améliorer des solutions sous-optimales. *Cabins* s'applique au problème d'ordonnancement appelé “job shop” où un ensemble de tâches doit être ordonné sur plusieurs machines non interchangeables de sorte que chaque tâche soit travaillé, en séquence, par chaque machine. A un ordonnancement préliminaire construit par une technique d'optimisation heuristique telle que le recuit simulé, *Cabins* apporte des modifications locales afin de l'améliorer. Plusieurs fonctions objectifs sont en pratique employées dans ce type de problème. Typiquement, il s'agit d'améliorer un ordonnancement par rapport à sa durée totale, mais aussi de limiter le nombre de tâches qui sont simultanément en travail, ou encore de minimiser le retard des tâches sujettes à des impératifs de délais. Ces critères sont souvent contradictoires, et chaque utilisateur attribue à chacun d'eux une importance différente.

Le système *Cabins* prend en compte, de façon originale, les préférences des utilisateurs lorsqu'il révise un ordonnancement. En effet, *Cabins* n'emploie pas de fonction objectif explicite, mais utilise plutôt des cas comme représentations par extension de la fonction objectif propre à l'utilisateur. Un cas est composé d'une action de révision, de la description de la situation dans laquelle cette action fut entreprise et de rétributions relatives à l'application de l'action de révision dans cette situation. Les situations des cas comportent tout autant des caractéristiques locales de la tâche sujet de révision, que des caractéristiques globales de l'ordonnancement tel qu'il se présentait avant la révision. Les cas eux-mêmes sont acquis à partir de l'observation des actions de révision telles que réalisées par l'utilisateur. Cette acquisition de cas procède donc de la même logique que celle de *Clavier* [Barletta & Hennessy 1989]. Lorsque *Cabins* est face à une situation pour laquelle il dispose d'un cas, l'action de révision de ce dernier est réappliquée à l'ordonnancement.

c) Le problème du voyageur de commerce

Il existe une étude sur l'application de raisonnement à partir de cas au problème du *voyageur de commerce* [Cunningham & al. 1995]. Le but est d'éviter ici le recours systématique à l'optimisation dès le début de chaque instance de problème. Il s'agit donc de réutiliser d'anciennes tournées de bonne qualité, représentées dans les cas, en les modifiant légèrement pour prendre en compte la spécificité de la nouvelle instance. L'adaptation d'une tournée ancienne se fait en substituant les villes qui appartiennent à la nouvelle instance, et n'y figurent pas, aux villes qui n'en font plus partie. Lorsque les différences au niveau des villes sont éliminées, le système procède à une amélioration locale de la tournée en appliquant des transformations du type 2-opt [Lin 1965] à l'ensemble des villes voisines.

Les instances du problème contiennent 40 villes chacune. Ces villes sont tirées aléatoirement dans un ensemble invariant de 100 villes. Les cas proviennent aussi de tournées impliquant 40 villes parmi les mêmes 100. Ils sont au préalable construits par une technique simple de recuit simulé. La similarité entre un cas et une instance de problème est déterminée par le nombre de villes qu'elles ont en commun. Les auteurs constatent que plus cette similarité entre une instance et un cas est élevée, plus la qualité de la tournée solution est bonne. Ils en concluent que le nombre de cas nécessaires, pour couvrir l'espace des tournées de manière satisfaisante, est très élevé[6]. En ce qui concerne le temps de calcul nécessaire pour l'adaptation de la solution, il est bien moins important que celui que nécessite le recuit simulé pour la construction d'une solution de bonne qualité.

Ce chapitre nous a permis d'esquisser les principes de l'apprentissage analytique ainsi que du raisonnement à partir de cas. Nous avons présenté ce dernier de manière indépendante de la méthode tabou. Or, cette thèse à pour objectif de combiner ces deux techniques afin d'obtenir une nouvelle approche de la résolution de problèmes d'optimisation. La présentation de cette approche fera l'objet du prochain chapitre.

---

[6] Pour des tournées à 40 villes, choisies parmi 500, il faudra 10^5 cas afin que 12 villes soient, en moyenne, communes à une instance de problème et au cas qui lui est le plus proche.

Chapitre 3.

Le système *Alois*

Le système *Alois* introduit une technique de raisonnement à partir de cas dans la méthode d'optimisation qu'est la recherche tabou. L'objet de ce chapitre est de présenter et de justifier cette intégration. Cette technique acquiert des connaissances de contrôle relatives à l'exploration de l'espace de recherche dans le but de focaliser la recherche sur des solutions de bonne qualité. Dans la première section nous décrirons les choix conceptuels essentiels guidant l'intégration des deux techniques. La deuxième section sera, quant à elle, consacrée à l'algorithme d'optimisation fusionnant les deux techniques. Enfin, dans la troisième et dernière section nous évoquerons la modélisation des problèmes d'optimisation et, allant de pair avec celle-ci, la représentation des cas dans le système présenté.

3.1. L'apprentissage dans *Alois*

Dans les chapitres précédents, nous avons présenté la méthode tabou et le raisonnement à partir de cas. Il convient à présent de s'attarder davantage sur la mise en évidence des facettes communes à ces deux techniques. Nous considérerons d'abord la notion d'apprentissage en méthode tabou et en raisonnement à partir de cas, avant de traiter de la nature des connaissances que notre système apprendra et réutilisera.

3.1.1. La notion d'apprentissage en recherche tabou

La méthode tabou fait explicitement appel à des techniques d'apprentissage, et ceci plus particulièrement au niveau du contrôle stratégique de l'exploration de l'espace de recherche. Les stratégies d'intensification et de diversification étant donc concernées, il convient d'évoquer de manière plus détaillée les connaissances apprises par ces méthodes. Par la suite, nous verrons que la nature de ces connaissances est relativement proche de celle des connaissances employées par le raisonnement à partir de cas.

a) Les connaissances apprises en méthode tabou

Divers mécanismes d'intensification et de diversification sont employés dans le cadre de la méthode tabou (cf. p. 40). Ils reposent soit sur une analyse de la fréquence à laquelle intervient un certain événement, soit sur le délai écoulé depuis la survenance d'un événement [Woodruff 1994]. A titre indicatif, des éléments d'observation représentatifs peuvent être la fréquence de la modification d'une variable du problème d'optimisation, la présence d'une valeur particulière d'une ou de plusieurs variables dans une ou plusieurs solutions de bonne qualité, ou encore le délai écoulé depuis la dernière application d'une transition.

Ces éléments d'observation ne font pas partie de la modélisation initiale du problème d'optimisation. Il appartient plutôt à l'utilisateur aguerri de décider quels éléments doivent être analysés. Un élément est typiquement représenté par une variable, désignée par l'utilisateur. Le contrôle de la recherche est alors modifié selon la valeur que prend l'une ou l'autre de ces variables. En ce qui concerne les techniques de contrôle stratégique de base, c'est également à l'utilisateur de déterminer le comportement que doit adopter la recherche afin de répondre aux valeurs que prennent ces variables. Dans ce contexte, la technique d'apprentissage a alors essentiellement pour tâche de modifier ces variables, au fur et à mesure que l'exploration de l'espace de recherche progresse, et de mettre en oeuvre des stratégies de contrôle prédéfinies.

Considérons l'exemple d'un problème du type placement et allocation (cf. p. 166), et introduisons la variable W qui compte le nombre d'entités installées dans une solution. Une technique d'intensification peut alors déterminer W pour chaque solution rencontrée [Crainic & al. 1993b]. Si l'analyse de la distribution des valeurs de W indique qu'on a $W = k$ pour plusieurs bonnes solutions, une technique d'intensification pourrait alors, pendant quelques itérations, n'admettre que des transitions menant vers des solutions à k entités. Nous avons vu, dans la section relative à la méthode tabou, que le choix de ces éléments, ainsi que des actions qui en découlent, peut considérablement participer au succès d'une méthode tabou.

Ce qui est vrai pour ces techniques de contrôle stratégique de base, le reste également pour les techniques d'apprentissage plus élaborées telles que la *recherche tabou réactive*, *l'analyse de la cible* ou le *chunking*, introduites au premier chapitre (cf. p. 40). Le propos doit néanmoins être nuancé. En ce qui concerne le *chunking* [Woodruff 1994], qui est basé sur des chunks qui sont des bouts de solutions, ces derniers ne sont pas nécessairement donnés par l'utilisateur, mais peuvent être déterminés par une analyse préalable d'instances de problèmes typiques. Le contrôle de la recherche, effectué à partir de ces chunks, rentre cependant tout à fait dans le cadre dressé ci-dessus.

Pour ce qui est de la *recherche tabou réactive* [Battiti & Tecchiolli 1994], il s'agit simplement d'adapter, au cours de l'optimisation, des paramètres qui sont prédéfinis par la méthode. Ces paramètres sont relatifs au nombre de solutions qui peuvent être visitées de manière répétitive ou au nombre de fois où ces solutions peuvent être visitées. Une fois que ces paramètres ont franchi un certain seuil, la méthode réagit en modifiant le contrôle de la recherche. Concernant la méthode de *l'analyse de la cible* [Laguna & Glover 1993], cette dernière permet d'identifier certains paramètres d'une nouvelle fonction d'évaluation des transitions. Si, au bout d'une analyse approfondie portant sur plusieurs optimisations d'instances de problèmes typiques, l'analyse de la cible identifie les bonnes valeurs de ces paramètres, il appartient néanmoins à l'utilisateur de proposer lui-même la forme de la nouvelle fonction d'évaluation.

b) Une similitude avec le raisonnement à partir de cas

Au terme de cette analyse des techniques d'intensification et de diversification, on peut constater que les techniques d'apprentissage employées sont plutôt simples. Elles se limitent à affecter ces valeurs à des variables, et à coordonner en fonction de ces valeurs un contrôle assez figé de la recherche. Il convient toutefois de préciser que le choix de techniques d'apprentissage simples est tout à fait raisonnable compte tenu du contexte dans lequel ces techniques sont utilisées. Il s'agit, en effet, de limiter le coût opérationnel de ces techniques au minimum, tout en tentant d'obtenir un rendement maximal en termes de qualité de solutions.

Examinons maintenant la nature des connaissances produites par ces techniques de contrôle stratégique. D'une part, ces connaissances sont de nature *opérationnelle*. Elles reposent sur des analyses concrètes d'épisodes de la recherche antérieure sur une instance du problème. Aucune tentative de généralisation n'est entreprise (sauf dans l'analyse de la cible) qui pourrait rendre ces connaissances utilisables sur un plus vaste champ d'instances ou de problèmes.

D'autre part, ces connaissances sont indiscutablement de nature *expérimentale*. Elles apparaissent lorsque l'exploration de l'espace de recherche progresse, et elles constituent une expérience propre à l'instance de problème qui est optimisée. De même, il n'y a pas d'expert humain qui possède des connaissances concernant le contrôle stratégique de la recherche tabou[1], l'expérience étant la seule source de connaissances. En effet, même l'utilisateur averti ne réussit à identifier avec précision les processus de con-

---

[1] Il en va de même en ce qui concerne les connaissances de contrôle dans la génération de plans [Minton 1989].

trôle adaptés à son problème d'optimisation qu'au bout de maintes expérimentations [Hertz & De Werra 1987].

En conséquence, les techniques d'apprentissage en méthode tabou créent des connaissances qui s'avèrent proche du raisonnement à partir de cas pour deux raisons. D'une part, car ce dernier s'attache particulièrement à la mémorisation et à la réutilisation de connaissances opérationnelles, et ceci sans faire appel à des techniques d'induction ou de généralisation. D'autre part, parce que le raisonnement à partir de cas se révèle bien adapté à la manipulation de connaissances expérimentales (chapitre 2, cf. p. 58). En outre, l'efficacité du raisonnement à partir de cas a été reconnue à maintes reprises [Koton 1988, Barletta & Hennessy 1989, Sycara & Miyashita 1994, Cunningham & al. 1995].

c) Des disparités avec d'autres techniques d'apprentissage

Si l'on souhaite établir un parallèle, on peut dire que les autres techniques d'apprentissage symbolique sont aussi loin de l'apprentissage en méthode tabou que le raisonnement à partir de cas en est proche. Parmi ces autres techniques d'apprentissage figurent l'apprentissage analytique, et plus particulièrement *l'apprentissage à partir d'explications*. Or, cette technique ne peut pas être utilisée dans la recherche locale (chapitre 2, cf. p. 56), et donc pas davantage dans la méthode tabou.

Toutefois, le recours à une autre famille de techniques d'apprentissage, *l'apprentissage par renforcement* (“*reinforcement learning*”) [Sutton1988] pourrait être envisagé. Cette dernière est une version heuristique de la *programmation dynamique* [Bertsekas 1995], méthode d'optimisation exacte. Cependant, ces techniques apprennent, au bout de maintes itérations, à mener de mieux en mieux la recherche vers un état “but”. Une critique similaire à celle adressée à l'apprentissage analytique peut ici être formulée. En effet, les états “buts” dans un problème d'optimisation sont les états optimaux. A priori, on ne les connaît pas et ils ne sont pas non plus identifiables de manière efficace. Une technique d'apprentissage par renforcement devra donc s'en tenir à apprendre les chemins conduisant vers des minima locaux, ce qui ne présenterait pas nécessairement d'intérêt dans le cadre posé ici [Woodruff 1994].

3.1.2. La nature des connaissances apprises

L'existence de points communs entre les techniques de contrôle stratégique en méthode tabou, et les capacités du raisonnement à partir de cas étant établie, nous pourrons à présent esquisser les lignes directrices de la conception de *Alois*.

a) Le contrôle de la recherche

L'intensification focalise la recherche sur des régions de l'espace qui devrait contenir des solutions ressemblant à d'autres solutions de bonne qualité. Elle fait donc implicitement sien l'hypothèse de base du raisonnement à partir de cas qui est que “des problèmes similaires ont des solutions similaires” [Leake 1996]. On pourrait la reformuler, afin de l'adapter au cadre de l'intensification, de la manière suivante : “des régions de l'espace similaires contiennent des solutions de qualité similaire”. Il nous parait donc prudent d'aborder l'intégration du raisonnement à partir de cas dans la méthode tabou par le biais de l'intensification.

Par contre, il n'en va pas de même en ce qui concerne la *diversification*. Celle-ci tente de conduire la recherche vers des régions de l'espace qui ne sont pas encore explorées, et qui sont donc loin des (bonnes) solutions déjà rencontrées. La diversification relève d'un principe directeur qui est à l'opposé de celui de l'intensification, et en conséquence de celui du raisonnement à partir de cas. Nous écartons donc pour l'instant toute approche de l'apprentissage s'inspirant de la diversification. Toutefois, nous évoquerons plus loin, comme une perspective de développement de nos travaux, l'intégration de la diversification au moyen du même mécanisme de raisonnement à partir de cas (cf. p. 195).

L'intensification a pour objet de procéder à un contrôle de l'exploration de l'espace de recherche. Pour ce faire, diverses techniques ont été proposées. L'une d'elles analyse les transitions applicables dans un état, et en fonction de sa mémoire, les juge d'intéressantes ou de néfastes. La sélection de la transition à appliquer prend alors en considération cette appréciation des transitions par l'intensification. Outre ce type d'approche qui modifie l'évaluation des transitions, d'autres stratégies d'intensification peuvent consister à introduire de nouvelles transitions, ou à éliminer des ensembles de transitions [Glover 1995].

Cependant seule la première de ces trois approches nous parait la plus apte à la construction d'une technique d'intensification est indépendante du problème d'optimisation. En effet, l'ajout de nouvelles transitions, ou la suppression d'anciennes, demande au concepteur de la technique une bonne compréhension du problème d'optimisation afin qu'il puisse arrêter son choix sur ces transitions. Or, une approche qui se veut indépendante du problème d'optimisation ne devrait pas supposer cette bienveillance de la part de l'utilisateur.

b) L'hypothèse du contexte

En raisonnement à partir de cas, un cas contient une description explicite du problème pour lequel il est significatif. Cette description est importante puisque c'est à travers elle que l'on identifie un cas comme contenant une solution qui est réutilisable lors de la résolution d'un nouveau problème. En intensification “classique”, cette notion de description explicite de l'état de recherche n'existe pas, l'intensification consistant essentiellement en un mécanisme ad hoc d'ajustement de divers paramètres du processus d'optimisation.

Considérons une exploration, schématique, d'un espace de recherche par une technique de recherche locale (figure 8). Dans chaque état, représenté par un sommet, un certain nombre d'opérateurs sont évalués, et l'un d'entre eux est sélectionné et appliqué. Celui-ci mène vers un état successeur. L'évaluation des opérateurs fait intervenir, par l'intermédiaire d'une fonction de prévision telle que l'équation (15) (cf. p. 26), la qualité de l'état successeur.

Ceci est un critère insuffisant pour la sélection d'opérateurs, puisqu'ainsi la recherche est conduite, au bout de quelques itérations, à un minimum local. En effet, l'intérêt de l'application d'un opérateur ne dépend pas seulement de son état successeur direct, mais aussi des opérateurs qui seront appliqués par la suite. Une fonction de prévision devrait alors tenir compte de la qualité des états successeurs engendrés postérieurement à l'opérateur qu'elle évalue. Il lui faudrait considérer, par anticipation, les prochaines étapes de la recherche. Cependant, ceci entraînerait une augmentation de la complexité du calcul de cette fonction de prévision, qui est en général inacceptable[2].

Rappelons que nous considérons les opérateurs comme étant des couples (conditions, actions) tels qu'employés dans un système de productions (chapitre 1, cf. p. 46). Un opérateur devient applicable dans un état si la description de cet état satisfait à la partie condition de l'opérateur. La partie action provoque une modification locale de l'état, une fois l'opérateur appliqué. Une chaîne d'opérateurs, telle que celle présentée dans la figure 8, introduit alors une suite de modifications locales de l'état d'origine de la chaîne. Chaque opérateur de la chaîne n'est applicable que parce que la description de l'état d'origine, modifiée par les actions de ses opérateurs prédécesseurs, satisfait à sa partie condition.

---

[2] Le développement récent des chaînes d'éjection pourrait peut-être nuancer ce propos [Rego & Roucairol 1995].

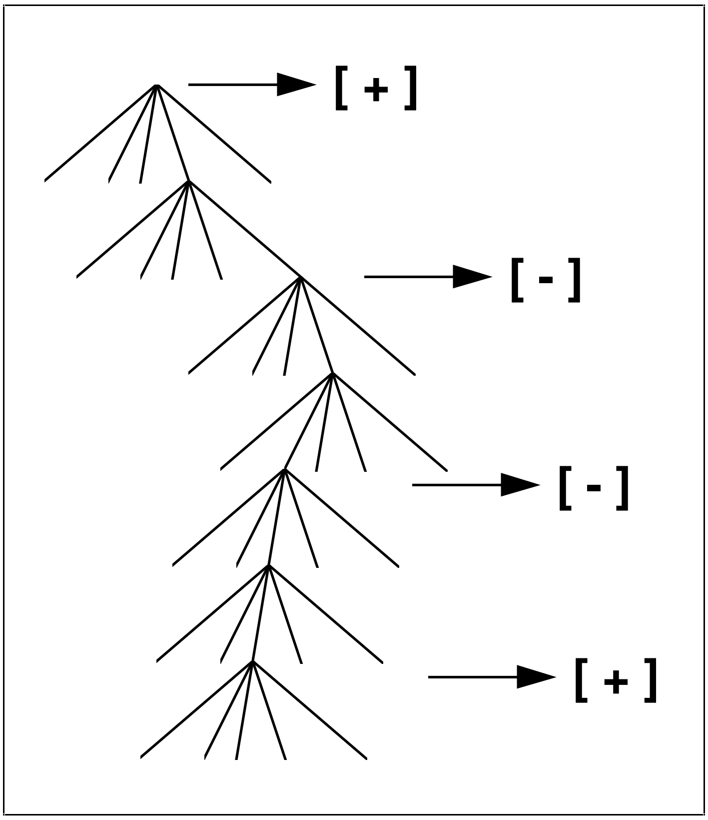


Figure 8. Exploration schématique d'un espace de recherche par une technique de recherche locale et attribution de récompense aux opérateurs

On peut en conclure que la description de l'état d'origine contient un sous-ensemble qui caractérise les états successeurs atteints au cours de l'application d'une chaîne d'opérateurs. Nous appelons pour l'instant *contexte* ce sous-ensemble de l'état d'origine de la chaîne. Le contexte englobe la description qui instancie la partie condition du premier opérateur de la chaîne, même s'il n'est pas nécessairement identique celle-ci.

De cette définition informelle du contexte découle une stratégie d'intensification originale et tout à fait simple. Nous observons des opérateurs qui conduisent à des états intéressants et nous mémorisons leurs contextes. Lors de l'évaluation d'un opérateur applicable, nous pouvons alors comparer son contexte à ceux mémorisés. En fonction de cette comparaison il est possible d'ajuster, c'est à dire d'augmenter ou de diminuer, l'évaluation de l'opérateur applicable.

Se pose alors le problème de la détermination de l'intérêt qu'il y a eu à appliquer un opérateur. Supposons pour l'instant que nous disposions d'un *oracle* qui décerne, a posteriori, une récompense indiquant si l'application de l'opérateur a été un succès, ou un échec. Dans la figure 8, cette récompense est notée `[+]` pour les succès et `[−]` en cas d'échec. Dans la suite de ce chapitre, nous décrirons de façon plus détaillée ce que nous entendons par “contexte” et par “récompense”. Mais regardons à présent l'algorithme d'optimisation de *Alois*.

3.2. L'algorithme d'optimisation

Cette section présentera le système *Alois*. les deux principaux composants en sont une technique de recherche locale simple et un mécanisme de raisonnement à partir de cas. Nous évoquerons d'abord l'architecture de *Alois* d'un point de vue abstrait, avant de présenter en détails l'interaction de ses deux éléments constitutifs.

3.2.1. L'architecture du système

Le tableau 6 présente le système *Alois*. En entrée, *Alois* requiert une description du problème, la description d'une instance de ce problème et une méthode tabou de base. Cette dernière doit disposer de la faculté de contrôle tactique proposé en recherche tabou, contrôle constitué d'une ou plusieurs listes tabou et, s'il y a lieu, du critère d'aspiration. La technique de raisonnement à partir de cas de *Alois* ne traite pas du contrôle tactique de la recherche, mais est relative au contrôle stratégique.

Les données :

- la **description d'un problème**, composée d'une théorie du domaine (un ensemble de schémas d'opérateur), d'une fonction d'évaluation pour chaque schéma d'opérateur $p(\cdot)$, et d'une fonction objectif $f(\cdot)$,
- la **description d'une instance** de problème et de sa solution initiale, et
- une **méthode tabou** de base comportant le contrôle tactique.

La procédure :

- **explore** de manière itérative l'espace de recherche en appliquant à chaque itération l'opérateur ayant la meilleure évaluation $p(\cdot)$,
- **adapte** l'évaluation des opérateurs $p(\cdot)$ en tenant compte de l'expérience de la recherche anrterieure. L'expérience est manipulée par une technique de raisonnement à partir de cas.

Le but :

- Identifier des solutions **proches de l'optimum** (ou optimales).

Tableau 6. Le système *Alois*

A la base, *Alois* explore l'espace de recherche à la manière d'une technique de recherche locale classique qui passe itérativement, par l'application d'opérateurs, d'un état à un autre. Une technique de raisonnement à partir de cas est ajoutée à ce mécanisme d'optimisation simple. Celle-ci a pour but d'améliorer le contrôle de la recherche afin de guider celle-ci vers des régions de l'espace qui contiennent des solutions de bonne qualité. Pour ce faire, la technique de raisonnement à partir de cas acquiert de l'expérience relative à l'application des opérateurs. Ces connaissances sont représentées sous forme de cas et retenues dans une mémoire. Un cas est relatif à l'application d'un opérateur. Il contient une description du contexte dans lequel l'opérateur s'appliquait, ainsi qu'une récompense attribuée à cette application.

Cette expérience est réutilisée afin de fournir des connaissances de contrôle stratégique, au cours de la même optimisation, dès lors que la mémoire dispose de cas relatifs à un opérateur applicable. La réutilisation porte sur tout opérateur applicable et s'effectue en deux étapes. D'abord, il s'agit de récupérer dans la mémoire, le cas dont le contexte est le plus similaire à l'opérateur applicable. Ensuite, la récompense contenue dans ce cas est utilisée afin de modifier la fonction d'évaluation de l'opérateur. Une fois la récupération de cas terminée pour tout opérateur applicable, le système sélectionne l'opérateur qui possède la meilleure évaluation, ce qui correspond au comportement classique en recherche locale.

Notons que nous utilisons un moteur d'inférence classique fonctionnant en *chaînage avant* pour la représentation de problèmes d'optimisation ainsi que pour la manipulation des opérateurs, c'est à dire la détermination des opérateurs applicables et pour l'application des opérateurs elle-même. Le système retenu, *Clips*, [Riley 1991, Giarrantano & Riley 1994], emploie un réseau de filtrage de type *Rete* [Forgy 1982] pour accélérer la phase de détermination des opérateurs applicables.

3.2.2. L'exploration de l'espace de recherche

Le tableau 7 donne l'algorithme d'optimisation de *Alois*. Un problème d'optimisation est défini par la théorie du domaine `domaine`, qui contient les schémas d'opérateur et le contrôle tactique de la méthode tabou, une instance du problème `instance`, la condition d'arrêt `arrêt` et la solution initiale `init`. Diverses conditions d'arrêt sont envisageables, telles que le nombre d'itérations, le temps d'exécution, le nombre d'itérations sans amélioration de la solution, ou encore des combinaisons de ces dernières.

On remarque que la méthode d'optimisation esquissée au tableau 7 correspond à la recherche locale classique. En effet, *Alois* itère d'un état solution à l'autre jusqu'à ce que la condition d'arrêt soit satisfaite. Rappelons que la recherche locale ne passe a priori que par des états solutions. Nous avons cependant proposé d'introduire, dans l'espace de recherche, des états qui ne sont pas des solutions (chapitre 1, cf. p. 31), et nous avons constaté qu'il est dès lors possible d'employer des opérateurs qui ont des voisinages moins complexes. Le système *Alois* peut utiliser cette faculté. La fonction *Prochaine_Solution* construit à partir d'un état solution son état solution successeur (tableau 8). Pour ce faire, elle peut passer par des états qui, eux, ne sont pas des solutions. En effet, elle construit ce que nous avons appelé auparavant une transformation, c'est à dire elle produit une chaîne d'opérateurs qui mène d'un état solution à un autre. *Prochaine_Solution* applique itérativement des opérateurs à l'état courant. Le contrôle, que l'utilisateur doit apporter, assure que cette boucle se termine en un nombre fini d'itérations. Dans chaque état, le système détermine l'ensemble des opérateurs applicables (tableau 8, ligne 2) et applique celui qui offre la meilleure évaluation de la fonction de prévision (ligne 7). En outre, le contrôle tactique de la méthode tabou est mise à jour (ligne 8).

```
fonction Optimise(domaine, instance, arrêt, init) : une solution
1  initialise le moteur d'inférence avec domaine et instance
2  Courant := init
3  Meilleur := init
4  tant que (arrêt non satisfait)
5      Prochain := Prochaine_Solution(Courant)
6      Courant := Prochain
7      si (Prochain est meilleur que Meilleur) alors
8          Meilleur := Prochain
9  fin tant que
10 renvoi Meilleur
```

Tableau 7. La fonction *Optimise()* de *Alois*

```
fonction Prochaine_Solution(Courant) : un état solution
1  répète
2      Agenda := ensemble des opérateurs non-tabou applicables à Courant
3      pour (chaque opérateur $o_i$ de Agenda) fait
4          $p_0$ := évalution de $o_i$, $p(o_i)$
5      fin pour
6      o := sélectionne dans Agenda l'opérateur avec la prévision maximale
7      Courant := applique o à Courant
8      mise à jour du contrôle tactique
9  jusqu'à (Courant est un état solution)
10 renvoi Courant
```

Tableau 8. Construction d'une transformation dans *Alois*

3.2.3. Le raisonnement à partir de cas

Le système *Alois* élargit la technique de recherche locale présentée ci-dessus par un composant raisonnement à partir de cas. Examinons ce dernier plus en détails.

a) La construction et l'acquisition des cas

Nous avons déjà laissé entendre qu'un cas dans *Alois* est un couple (`contexte`, `récompense`) où la `récompense` décrit la qualification d'une expérience portant sur l'application d'un opérateur dans un `contexte` particulier. Lors de la réutilisation, cette récompense modifiera l'évaluation des opérateurs qui s'appliquent dans des contextes semblables.

Une question importante concerne le moment où l'acquisition d'expérience peut avoir lieu. En effet, *Alois* entame toute nouvelle optimisation avec une mémoire de cas *vide*. Comme nous l'avons vu auparavant, si le transfert de connaissances entre instances de problèmes est une caractéristique principale des techniques d'apprentissage analytique, il n'en va pas ainsi du contrôle stratégique de la méthode tabou. Le système acquiert donc l'ensemble de son expérience au cours de la même optimisation durant laquelle il la réutilisera.

S'il n'est guère judicieux de procéder à une acquisition à l'intérieure d'une transformation, la fonction objectif n'étant pas définie pour des états qui ne sont pas des solutions, il n'en va pas de même une fois la transformation achevée. Ainsi, au terme de chaque transformation, les opérateurs appliqués sont transmis au mécanisme du raisonnement à partir de cas pour la construction et la mémorisation de cas. Ceci se reflète au niveau de la fonction *Optimise* (tableau 7), à laquelle les lignes 5.a - 5.e (tableau 9) seront ajoutées. Ainsi, pour chaque opérateur appliqué pendant la transformation menant de l'état courant, `Courant`, de la recherche à son état successeur `Prochain`, un cas est construit et inséré dans la mémoire de cas. Dans le chapitre 4, nous préciserons l'insertion d'un cas dans la mémoire (cf. p. 101).

Extension de la fonction *Optimise*

```
5.a  O := les opérateurs appliqués pour passer de Courant à Prochain
5.b  pour (chaque opérateur $o_i$ de O) fait
5.c      $cas_i$ := Construit_Cas($o_i$, Courant, Prochain, Meilleur)
5.d      Insère($cas_i$)
5.e  fin pour
```

Tableau 9. Acquisition des cas, extension au tableau 7

Le tableau 10 présente la construction des cas. En premier lieu, une récompense est attribuée par l'oracle. Ensuite, le contexte dans lequel l'opérateur s'est appliqué est construit. Appelé *contexte d'instanciation*, il est établi par un algorithme qui est indépendant du problème d'optimisation considéré, mais qui se base directement sur sa modélisation. Cet algorithme est présenté plus loin (cf. p. 95). Pour faciliter la manipulation ultérieure de son contenu, le contexte du cas est paramétré (cf. p. 99).

```
fonction Construit_Cas(opérateur, Courant, Prochain, Meilleur) : cas
    récompense := Construit_Récompense(opérateur, Courant, Prochain,
        Meilleur)
    context := Context_Instanciation(opérateur)
    Paramétre(context)
    renvoi cas := {context, récompense}
```

Tableau 10. Construction d'un cas

b) L'oracle et la récompense

L'*oracle* a pour tâche d'attribuer une récompense à l'application d'un opérateur dans un contexte particulier. Intuitivement, cette récompense devrait dépendre du degré de rapprochement de la recherche de la solution optimale après l'application de l'opérateur. Or, la solution optimale n'est bien évidemment pas connue.

Se pose donc la question de l'attribution de la récompense qui, à notre avis, ne peut être qu'heuristique. Dans le cadre des techniques *d'apprentissage par renforcement* divers algorithmes relatifs à l'attribution de récompenses ont été proposés. Cependant, nous n'avons pas envisagé de recourir à l'un de ces algorithmes pour les même raisons qui nous ont conduit à juger ces techniques d'apprentissage inadaptées à l'apprentissage de connaissances de contrôle dans la méthode tabou (cf. p. 76). Il se pourrait toutefois qu'une technique d'attribution de récompense de ce type puisse être retenue dans le cadre de *Alois*.

Nous avons donc opté pour une troisième approche de l'attribution de la récompense (tableau 11), dont le choix ne repose que sur des considérations de nature intuitive. L'opérateur appliqué fait partie de la transformation qui conduit de l'état `Courant` vers l'état `Prochain`. Toutefois, l'opérateur n'a ni nécessairement été appliqué à `Courant`, ni lui-même engendré `Prochain`. Si cette transformation a amélioré la fonction objectif en passant de `Courant` à `Prochain`, la récompense attribuée à l'opérateur est positive (*bonus*), sans quoi elle est négative (*malus*). Cette récompense est de nature qualitative, elle ne fait pas intervenir l'écart de la fonction objectif entre les deux états solutions.

```
fonction Construit_Récompense(opérateur, Courant, Prochain, Meilleur) : la récompense
    si (Prochain est meilleur que Courant) alors récompense := bonus
        sinon récompense := malus
    renvoi récompense
```

Tableau 11. Construction de la récompense

Il est clair que cette attribution de récompense est rudimentaire. Si elle a un sens dans des domaines où les transformations sont effectivement des enchaînements d'opérateurs, il n'en va pas exactement de même pour des modélisations où les opérateurs mènent directement d'un état solution vers un autre. La coloration de graphe en est un exemple (cf. p. 187). Cependant, comme l'évaluation expérimentale le révèle, même dans ce domaine cette récompense apporte un contrôle amélioré de la recherche.

c) La réutilisation des cas

L'expérience représentée par les cas concerne l'intérêt des opérateurs applicables. Si, pour un opérateur applicable, un cas contient une récompense positive, l'intérêt qu'il peut y avoir à appliquer l'opérateur se trouve renforcé, tandis que si la récompense est négative, l'intérêt est diminué. En outre, nous supposons que plus le contexte contenu dans un cas ressemble au contexte d'un opérateur applicable, plus l'expérience que représente le cas concerne l'opérateur.

Cette réutilisation est mise en oeuvre de la manière suivante. D'abord, il s'agit de procéder à une analyse approfondie de chaque opérateur applicable. Pour ce faire, il faut apporter les trois modifications présentées au tableau 12 à la fonction *Prochaine_Solution* (cf. p. 83). Ainsi, pour chaque opérateur applicable, le cas le plus proche est récupéré dans la mémoire. La réutilisation de ce cas donne une nouvelle valeur à la fonction d'évaluation de l'opérateur. La sélection de l'opérateur à appliquer se fera alors sur la base de cette nouvelle évaluation.

```
Extensions de la fonction Prochaine_Solution
4.a    cas := Récupère($o_i$)
4.b    $p_1$ := Réutilise($o_i$, cas, $p_0$)
4.c    affecte $p_1$ comme nouvelle prévision à $o_i$
```

Tableau 12. Réutilisation des cas, extensions au tableau 8

Ensuite, on procède à la réutilisation proprement dite. Le tableau 13 montre ce processus d'adaptation. D'abord, la similarité entre le cas récupéré et l'opérateur applicable est déterminée (tableau 13, ligne 3). Au chapitre 5 nous évoquerons la description de la mesure de similarité employée (cf. p. 127). La similarité est normalisée dans l'intervalle [0…1], avec `sim` = 1 pour deux contextes identiques, et `sim` = 0 pour deux contextes totalement différents.

```
fonction Réutilise(opérateur, cas, intérêt) : la nouvelle évaluation
1   context := Context_Instanciation(operator)
2   Cas_context := Context_Instanciation(cas)
3   similarité := sim(Cas_context, context)
4   récompense := la récompense du cas
5   $\alpha$ := 1 + similarité
6   si (récompense = bonus) alors p1 := $\alpha \cdot$ intérêt
7       sinon p1 := intérêt / $\alpha$
8   renvoi p1
```

Tableau 13. Réutilisation d'un cas

La modification de l'évaluation de l'opérateur se fait par rapport à, d'une part, cette similarité et, d'autre part, la récompense du cas. En ce qui concerne la formule précise de calcul de la nouvelle évaluation de l'opérateur, nous avons choisi une première approche simple (lignes 5 à 7). Prenons pour exemple une similarité de 0.75. Pour une récompense *bonus*, l'ancienne prévision se trouve majorée de 75%, tandis qu'en cas de récompense *malus*, elle est divisée par 1.75. Plus précisément, la nouvelle prévision prend des valeurs dans l'intervalle :

$$\left[\frac{\texttt{intérêt}}{2} \quad \ldots \quad 2\cdot\texttt{intérêt}\right] \tag{20}$$

Le choix de cet intervalle est arbitraire. Une extension de la méthode pourrait, en effet, conduire à adapter cet intervalle en fonction des *rétributions* obtenues pour chaque réutilisation d'un cas. De même, il serait peut-être possible de se reporter aux techniques de gestion de récompenses telles qu'employées en apprentissage par renforcement. Cependant dans sa version actuelle, *Alois* ne fait ni l'un ni l'autre.

3.3. La représentation des problèmes

Alois comprend un langage formel de représentation des problèmes. Ce langage est basé sur la logique et permet la modélisation d'un large éventail de problèmes d'optimisation sans pour autant nécessiter une réimplantation de la technique d'apprentissage. Cette section a pour but de présenter ce langage. Nous étudierons d'abord les deux types de langages qui sont généralement utilisés en raisonnement à partir de cas. Nous présenterons ensuite le langage retenu pour *Alois*. Nous consacrerons un troisième paragraphe à la représentation et à la mise en forme des cas de *Alois*.

3.3.1. Langages de représentation en raisonnement à partir de cas

En raisonnement à partir de cas, différents langages de description peuvent être utilisés pour la représentation des cas. Nous verrons que le choix du langage a des conséquences importantes sur les capacités d'un système. Si la plupart d'entre eux se fondent sur des langages basés sur la *logique des propositions*, certains s'appuient sur la *logique des prédicats*.

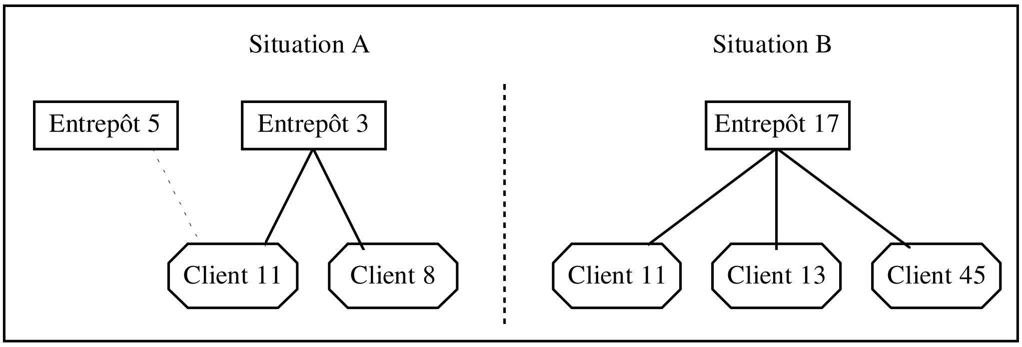


Figure 9. Deux exemples simplifiés du problème de placement et d'allocation d'entités

A partir de l'exemple de la figure 9 du problème de placement et d'allocation d'entités (cf. p. 166), nous présenterons les conséquences du choix d'un type de langage de représentation plutôt qu'un autre. Les deux situations A et B font apparaître des fragments de la configuration, pris à des moments différents, d'un même réseau de distribution. Dans la situation A, l'entrepôt *Entrepôt_3*, fermé par l'opérateur `closeWarehouse`, desservait deux clients, *Client_11* et *Client_8*. Notons que le *Client_11* est aussi desservi par l'entrepôt *Entrepôt_5*. Mais cette liaison n'est pas mise en cause par la fermeture de *Entrepôt_3*. Dans la situation B l'entrepôt *Entrepôt_17* approvisionne les trois clients *Client_11*, *Client_13* et *Client_45*. Nous supposons que la situation A, antérieure à la situation B, fait partie de la description d'un cas. Il s'agit alors de vérifier, en comparant les deux situations, si le cas contenant la situation A est réutilisable dans la situation B.

a) Représentation en ordre 0+

Une *représentation en ordre 0+*, ou en logique de propositions, est composée d'un ensemble de couples (attribut = valeur). Cette représentation est dominante en raisonnement à partir de cas [Kolodner 1993]. Le tableau 14 donne une représentation des situations de la figure 9 dans un langage d'ordre 0+. Chaque description contient des d'attributs comme `entité` ou `Client1`, et une valeur est associée à chaque attribut.

Situation A :	Situation B :
`entité` = *Entrepôt_3*	`entité` = *Entrepôt_17*
`Client1` = *Client_11*	`Client1` = *Client_11*
`Client2` = *Client_8*	`Client2` = *Client_13*
	`Client3` = *Client_45*

Tableau 14. Exemples de représentations en logique des propositions

Dans une telle représentation, la similitude entre deux situations est déterminée par la comparaison des valeurs que possèdent les attributs du même nom. Selon ce critère, les situations A et B sont assez différentes, puisqu'un seul attribut (`Client1`) a la même valeur. Pour un système de raisonnement à partir de cas qui emploie une représentation de ce type, la réutilisation, dans la situation B, d'un cas contenant la situation A, ne serait donc pas nécessairement judicieuse. Le problème de ce type de représentation est que les attributs d'une situation se voient attacher d'office un partenaire auquel ils doivent être comparés. Ainsi, l'attribut `Client2` de la situation A est par défaut comparé à l'attribut `Client2` de B, ce qui conduit à une similarité basse entre eux s'ils ne partagent pas la même valeur[3].

Le système d'optimisation à partir de cas qui s'applique au problème du voyageur de commerce (chapitre 2, cf. p. 71), souffre de ce type de problème [Cunningham & al. 1995]. Les cas n'y sont ainsi réutilisés que dans des situations très proches de celles contenues dans leur description. Les auteurs en concluent qu'il faudrait un nombre exorbitant de cas pour couvrir suffisamment l'espace de recherche auquel est confronté le système.

---

[3] Certains systèmes de raisonnement à partir de cas emploient des connaissances, propres à leur domaine d'application, afin de justifier une comparaison entre attributs de nom différent [Koton 1988].

b) Représentation en ordre 1

Les *représentations en ordre 1*, à base de langages de la logique des prédicats, sont employées moins fréquemment en raisonnement à partir de cas que ne le sont les représentations en ordre 0+. En constituent des exemples *Prodigy/Analogy* [Veloso 1994] ou [Bunke & Messmer 1993] et [Plaza 1995]. Les situations A et B peuvent être formulées dans une représentation d'ordre 1 (tableau 15). Une situation est composée d'un ensemble de littéraux (p. e. `entité(...)`), chaque littéral comportant des attributs. Lors de la comparaison de deux situations, le système construit des *mises en correspondance* entre les attributs. Représenteront une mise en correspondance possible, par exemple, (*Entrepôt_3*, *Entrepôt_17*). Dans une telle représentation, les deux situations A et B s'avèrent assez similaires. En effet, les trois mises en correspondance (*Entrepôt_3*, *Entrepôt_17*), (*Client_11*, *Client_11*) et (*Client_8*, *Client_13*) créent des associations entre tous les attributs à l'exception de *Client_9* et de *Entrepôt_5*.

Situation A :	Situation B :
`entité`(*Entrepôt_3*)	`entité`(*Entrepôt_17*)
`lié`(*Entrepôt_3*, *Client_11*)	`lié`(*Entrepôt_17*, *Client_11*)
`lié`(*Entrepôt_3*, *Client_8*)	`lié`(*Entrepôt_17*, *Client_13*)
`entité`(*Entrepôt_5*)	`lié`(*Entrepôt_17*, *Client_45*)
`lié`(*Entrepôt_5*, *Client_11*)	

Tableau 15. Exemples de représentations en ordre 1

Un système de raisonnement à partir de cas qui emploie une telle représentation est donc plus flexible lors de la réutilisation de ses cas. Une représentation en langage d'ordre 1 présente toutefois un inconvénient majeur. Ainsi, lors de la comparaison de deux descriptions représentées dans un langage d'ordre 1, il est nécessaire de construire ces mises en correspondance entre les attributs [Ganascia 1987]. De la sorte, dans l'exemple ci-dessus, il faut établir les mises en correspondance entre les différents clients. Or, le problème de la construction des mises en correspondance, étant équivalent au problème d'isomorphisme de sous-graphes, il est en conséquence NP-complet [Bunke & Messmer 1993]. Nous détaillerons le lien qui existe entre l'évaluation de la similarité et la construction des mises en correspondance au chapitre 5 (cf. p. 127).

3.3.2. Le langage de représentation des problèmes

L'intégration d'un langage formel de représentation des problèmes, basé sur la logique, est chose classique en résolution de problèmes. Elle l'est moins en optimisation heuristique. En recherche tabou par exemple, chaque problème d'optimisation fait l'objet d'une implantation originale et exclusive. La raison en est que la performance d'une implantation est d'une importance supérieure à celle de son potentiel de réutilisation.

Dans *Alois*, nous employons un langage d'ordre 1, car les langages d'ordre 0+ requièrent un nombre trop important de cas, ne serait ce que pour assurer une couverture minimale de l'espace de recherche. Le système *Alois*, entamant chaque optimisation avec une mémoire de cas vide, il devrait effectuer un nombre important d'itérations avant de se doter du nombre de cas nécessaire.

a) Un langage en ordre 1

Le langage de modélisation de problèmes de *Alois* est une extension du langage de représentation du moteur d'inférence classique *Clips* [Riley 1991, Giarrantano & Riley 1994]. Ce langage est basé sur la logique des prédicats, dont il ne reste qu'un sous-ensemble d'éléments, dépourvu de la négation et des symboles de fonctions [Ganascia 1987]. Une instance de problème est décrite par un ensemble de *littéraux* positifs, les faits, dont les arguments sont des constantes. De même, les schémas d'opérateur contiennent, dans leur partie condition, un ensemble de littéraux. Cependant, ces littéraux peuvent contenir comme arguments, outre des constantes, des variables. La partie condition d'un opérateur, en résultant d'une instanciation d'un schéma d'opérateur, aura des constantes en lieu et à place de ces variables (figure 10, cf. p. 96).

Les extensions que nous avons apportées au langage de *Clips* sont documentées à l'annexe (cf. p. 227). Dans cette représentation, les faits ont la forme `(NomDePredicat [attribut]*)`[4]. Ils commencent par le nom du *prédicat*, suivi d'un nombre variable *d'attributs*. On définit la *position pos*(a) d'un attribut a dans un prédicat P comme étant le numéro de sa place, numérotée de gauche à droite, dans l'ensemble des attributs de P[5]. Nous donnerons des exemples de représentation ci-dessous.

Le langage de représentation de *Alois* est d'ordre 1. En conséquence, le problème de la construction des mises en correspondance se pose lors de la comparaisons de deux descriptions. S'agissant d'un problème NP-complet, la complexité de ce problème de mise en correspondance croît de manière exponentielle avec le nombre d'attributs que contient l'instance de problème. Si ce nombre d'attributs est proportionnel à la taille de l'instance de problème, la mise en correspondance est une opération aussi com-

---

[4] S'il s'agit là d'une représentation habituelle des faits dans les systèmes à base de règles, il n'en va pas de même pour la logique des prédicats où on préfère généralement une représentation telle que celle adoptée au tableau 15 ci-dessus. Bien évidemment, ces deux représentations sont équivalentes.

[5] Le langage de représentation de *Clips* connaît les *prédicats multivalués* où une même position peut être occupée par tout un ensemble d'attributs. Nous conservons cette faculté dans *Alois*, bien qu'elle soit restreinte, pour des raisons d'implantation, à la toute dernière position du prédicat.

plexe que le problème d'optimisation lui-même[6], ce qui est tout à fait indésirable. Afin de pallier au problème de complexité des mises en correspondance, deux techniques sont habituellement employées : le typage des attributs et la réduction du nombre des attributs. Nous utiliserons les deux techniques dans *Alois*.

b) Le typage des attributs

Le *typage des attributs* réduit l'espace des mises en correspondance en n'autorisant que des mises en correspondance entre attributs *du même type*. Il s'agit là d'une technique couramment utilisée en apprentissage symbolique inductif [Michalski 1983, Ganascia 1987]. Elle est également employée en raisonnement à partir de cas [Veloso 1994]. Outre la réduction de la complexité “réelle” du problème des mises en correspondance, le typage des attributs permet l'introduction de connaissances supplémentaires dans les descriptions [Ganascia 1987].

On peut distinguer trois types *ordonné*, *hiérarchique* et *contenu d'ensemble*. La typage des attributs du système d'apprentissage symbolique *Charade* [Ganascia 1987] introduit et prend explicitement en compte ces trois types, tandis que le système de raisonnement à partir de cas *Prodigy/Analogy* [Veloso 1994] ne connaît que des attributs symboliques, donc du type contenu d'ensemble. Les problèmes d'optimisation possèdent en général tout autant des attributs symboliques que des attributs numériques, de type numérique. En revanche, les attributs de type hiérarchique y sont plutôt rares. Dans cette optique, *Alois* recourt à ces deux derniers types, en excluant délibérément le type hiérarchique. En outre, on peut répartir les attributs, en fonction de leur sémantique, en deux classes : les attributs objet et les attributs valeur. Les attributs objet correspondent aux entités du domaine, alors que les attributs valeur décrivent ces objets et les liens existant entre eux [Bisson 1992]. Cette répartition n'est pas indépendante des types des attributs. Nous supposons dans *Alois* que les attributs objet sont toujours de type contenu d'ensemble. Il nous parait en effet dépourvu d'intérêt de considérer des attributs objet d'un type numérique[7]. En ce qui concerne les attributs valeur, les exemples dans cette thèse montrent qu'ils peuvent être des deux types retenus (chapitre 6).

Le typage des attributs ne réduit pas la complexité “au pire” du problème de la construction de mises en correspondance. Ceci devient évident lorsque le domaine n'est composé que d'attributs d'un seul type. Dans des domaines où il existe plusieurs types

---

[6] Nous supposons bien sûr avoir affaire à un problème d'optimisation NP-difficile dont le problème de décision associé est NP-complet, en fonction de la *taille* des instances.

[7] Tout du moins, nous n'en avons pas rencontrés lors de nos expériences.

d'attributs, on peut par contre espérer une réduction de la complexité “réelle” d'un algorithme de construction de mises en correspondance (chapitre 5, cf. p. 127).

c) Un exemple

Considérons la représentation du problème de placement et d'allocation d'entités comme exemple de modélisation. Le tableau 16 représente les attributs de ce domaine[8]. Nous considérons les entrepôts et les clients comme étant des attributs objet. Ils sont respectivement du type `Warehouse` et `Client` qui sont des types contenu d'ensemble. Les coûts et les flux sont alors des attributs valeur. Ils ont chacun leur type, `Cost` et `Flow`, des types numériques dont le domaine des valeurs contient les flottants.

Attribut	Classe	Type	Exemple
Entrepôt	Objet	`Warehouse`	`warehouse17`
Clients	Objet	`Client`	`client13`
Coûts	Valeur	`Cost`	`234`
Flux	Valeur	`Flow`	`765`

Tableau 16. Les attributs du domaine de UFLP et CFLP

La définition des prédicats du domaine figure au tableau 17. Le prédicat `warehouse` décrit pour un entrepôt le flux, ainsi que les coûts, qui lui sont associés ; `clientArc` décrit la liaison, en termes de coûts et de flux, entre un entrepôt et un client ; et `clientList` donne pour un entrepôt, la liste des clients qu'il dessert. Ce dernier littéral est un exemple de littéral multivalué. En effet, sa dernière position contient un ensemble d'attributs qui sont tous du même type. Un exemple de littéral de ce domaine est :

```
(clientArc warehouse17 client13 234 765)
```

qui indique que l'entrepôt `warehouse17` dessert `765` unités de marchandise au client `client13` pour un coût total de `234`. Dans ce littéral, *pos*(`client13`) = 2. La modélisation complète de ce problème se trouve en annexe (cf. p. 215).

Prédicats	Attributs			
`(warehouse`	`Warehouse`	`Cost`	`Flow)`	
`(clientArc`	`Warehouse`	`Client`	`Cost`	`Flow)`
`(clientList`	`Warehouse`	`Client`	`Client … )`	

Tableau 17. La définition des prédicats pour le problème de placement et d'allocation

---

[8] La modélisation des problèmes d'expérimentations est donnée en anglais.

3.3.3. La mise en forme des cas

La *mise en forme* de cas (“*situation assessment*”) concerne la construction de la description des cas. Son but est de réaliser une description du cas qui est caractéristique de l'événement que le cas représente [Kolodner 1993]. Pour ce faire, l'approche typique consiste à ne retenir qu'un sous-ensemble des éléments descripteurs en lieu et place de la description de l'état du système en son entier. Une telle technique est donc tout à fait à même de réduire la complexité de notre problème de mises en correspondance en réduisant la taille des descriptions, et ainsi, le nombre d'attributs.

a) Le footprinting

Le “*footprinting*” est la principale technique de mise en forme de cas pour les cas relatifs aux enchaînements d'opérateurs [Veloso 1994]. Il est défini dans le domaine de la génération de plans. Schématiquement, considérons un état initial S, un but G et un plan P qui mène de S à un état satisfaisant à G. Le plan P est composé d'un ensemble d'opérateurs o_i, $P = \{o_1, \ldots, o_n\}$. La technique du footprinting construit C, l'ensemble des conditions auxquelles satisfont les opérateurs du plan, $o_i \in P$, et qui sont présentes dans S. La description caractéristique du cas représentatif de P comprend alors C comme description.

Dans *Alois*, un cas représente l'application d'un seul opérateur. Une technique du type footprinting produit les littéraux instanciant la condition de l'opérateur dans l'état où celui-ci s'applique. C ne contient que les littéraux instanciés par l'opérateur. Or, nous avons précédemment supposé que c'est tout particulièrement ce contexte qui est utile à l'intensification.

Considérons encore une fois la situation A de la figure 9 (cf. p. 88) et sa description, telle que présentée au tableau 15 (cf. p. 90). La particularité de cette situation tient probablement au fait que l'un des clients est desservi simultanément par deux entrepôts. Cependant, la technique du footprinting n'inclut pas cette information dans la description d'un opérateur qui ferme l'un des deux entrepôts. En effet, la description du cas ressemblerait à celle présentée au tableau 15, à laquelle on aurait retiré les deux derniers littéraux.

b) Le contexte d'instanciation

La description de cas telle que conçue par la technique de footprinting est trop restrictive, comme nous venons de le voir. Il nous parait donc judicieux de l'élargir de manière à ce qu'elle englobe davantage d'informations concernant les liens entre les entités du domaine. Plus précisément, nous souhaiterions inclure l'ensemble des descriptions, sur les entités qui sont concernés par l'application de l'opérateur, disponibles. Dans l'exemple ci dessus, la mise en forme du cas devrait permettre d'inclure les deux derniers littéraux décrivant la situation A au tableau 15.

Considérons comme exemple le schéma d'opérateur `closeWarehouse` du problème de placement et d'allocation d'entités. La définition en est donnée dans la figure 10, accompagnée d'une instanciation de ce schéma d'opérateur sur la situation B introduite ci-dessus. Ce schéma d'opérateur a pour but de fermer une entité[9]. Sa partie condition comprend, outre le test du statut tabou de l'opérateur, trois littéraux. Le premier littéral `(role ?W)` indique que l'entrepôt qui instancie la variable `?W` est celui qui va être fermé. Le deuxième littéral détient les informations sur les caractéristiques de ce même entrepôt, à savoir le flux de marchandise qui le traverse et le coût inhérent à ce flux. Le troisième littéral contient la liste de l'ensemble des clients associés à l'entrepôt. La partie action de `closeWarehouse` enlève le deuxième et le troisième littéral de la description de l'état et y ajoute les faits indiquant que l'entrepôt `?W` est fermé en remettant à zéro le flux, le coût et la liste de clients.

Le littéral `role` a une fonction particulière. Il n'est pas indispensable d'en tenir compte pour décider si une instanciation du schéma d'opérateur est applicable dans un état donné. Son rôle est plutôt d'ajouter des informations supplémentaires concernant la description du schéma. Celles-ci seront, plus tard, utilisées pour réduire la complexité de la construction des mises en correspondance (chapitre 5, cf. p. 127). Notons que l'ajout de ce type d'informations est facultatif.

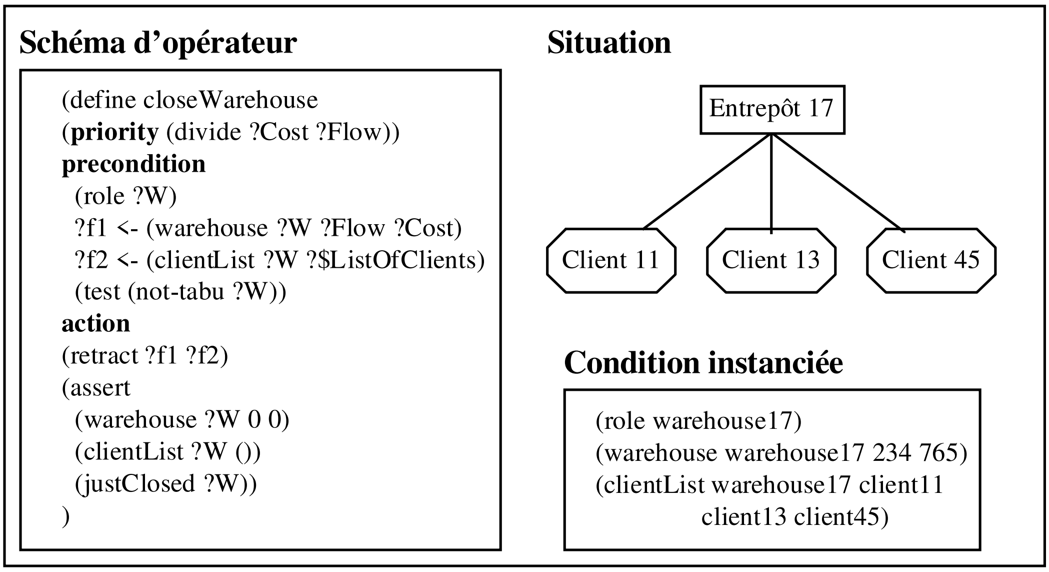


Schéma d'opérateur

```
(define closeWarehouse
(priority (divide ?Cost ?Flow))
precondition
  (role ?W)
  ?f1 <- (warehouse ?W ?Flow ?Cost)
  ?f2 <- (clientList ?W ?$ListOfClients)
  (test (not-tabu ?W))
action
(retract ?f1 ?f2)
(assert
  (warehouse ?W 0 0)
  (clientList ?W ())
  (justClosed ?W))
)
```

Situation

Condition instanciée

```
(role warehouse17)
(warehouse warehouse17 234 765)
(clientList warehouse17 client11
            client13 client45)
```

Figure 10. Le schéma d'opérateur `closeWarehouse` avec une instanciation

---

[9] Nous considérons que les entités sont des entrepôts.

La figure 10 indique quelle est la condition instanciée pour un opérateur du schéma `closeWarehouse` qui ferme `warehoue17`. L'état où l'opérateur s'applique correspond à celui de la situation B de la figure 9, à la différence près que la description inclut maintenant des attributs valeur. Avant sa fermeture, `warehoue17` avait comme client `Client11`, `Client13` et `Client45`. Les informations qui nous paraissent faire défaut dans cette condition instanciée sont relatives aux caractéristiques de ces clients. Dans la représentation des états ne se trouvent, en effet, que les informations concernant les flux de marchandise de l'entrepôt vers ses client, et leurs coûts. Or, comme nous l'avons vu, il est envisageable qu'un client soit desservi par plusieurs entrepôts.

Ces informations supplémentaires peuvent être identifiées dans la description de l'état en cherchant les littéraux qui contiennent les attributs objet présents dans la condition instanciée de l'opérateur. Plus précisément, nous définissons pour tout opérateur le *contexte d'instanciation* de niveau n, CI_n, comme un ensemble de strates St_i, $CI_n = \{St_i\}$, $i = 1 \ldots n$. Une strate est composée d'un ensemble de littéraux L_{ij}, $St_i = \{L_{ij}\}$. Nous appelons $A(CI_j)$ l'ensemble des attributs contenus dans les littéraux de CI_j, et $O(CI_j)$ l'ensemble des attributs objet de $A(CI_j)$. Nous définissons en outre, que la strate St_1 contient les littéraux `role` dans l'ordre dans lequel ils figurent dans la condition de l'opérateur. La strate St_2 regroupe l'ensemble des littéraux, hors les littéraux `role`, présents dans la condition instanciée. La strate St_i, $i > 2$, comprend l'ensemble des littéraux qui ne sont pas présents dans CI_{i-1} et qui contiennent des objets de $O(CI_{i-1})$.

Le contexte d'instanciation de niveau trois de l'opérateur présenté dans la figure 10 est présenté dans la figure 11. Le troisième niveau est composé de l'ensemble des littéraux, non présents dans la condition instanciée, mais qui contiennent les attributs objet `warehoue17`, `Client11`, `Client13` ou `Client45`.

Dans l'exemple, CI_4 serait vide, puisqu'aucun nouvel attribut objet n'est ajouté dans St_3. Ceci n'est généralement pas vrai. La plupart du temps, plus le niveau du contexte d'instanciation augmente, plus le nombre de littéraux qu'il contient croit, et plus nombreux sont les attributs. Une question se pose alors concernant le niveau maximal utile à la description des cas. En effet, un contexte d'instanciation d'un niveau élevé donne une description détaillée de l'état dans lequel l'opérateur s'est appliqué. Il conduit cependant à une complexité élevée de la construction des mises en correspondance.

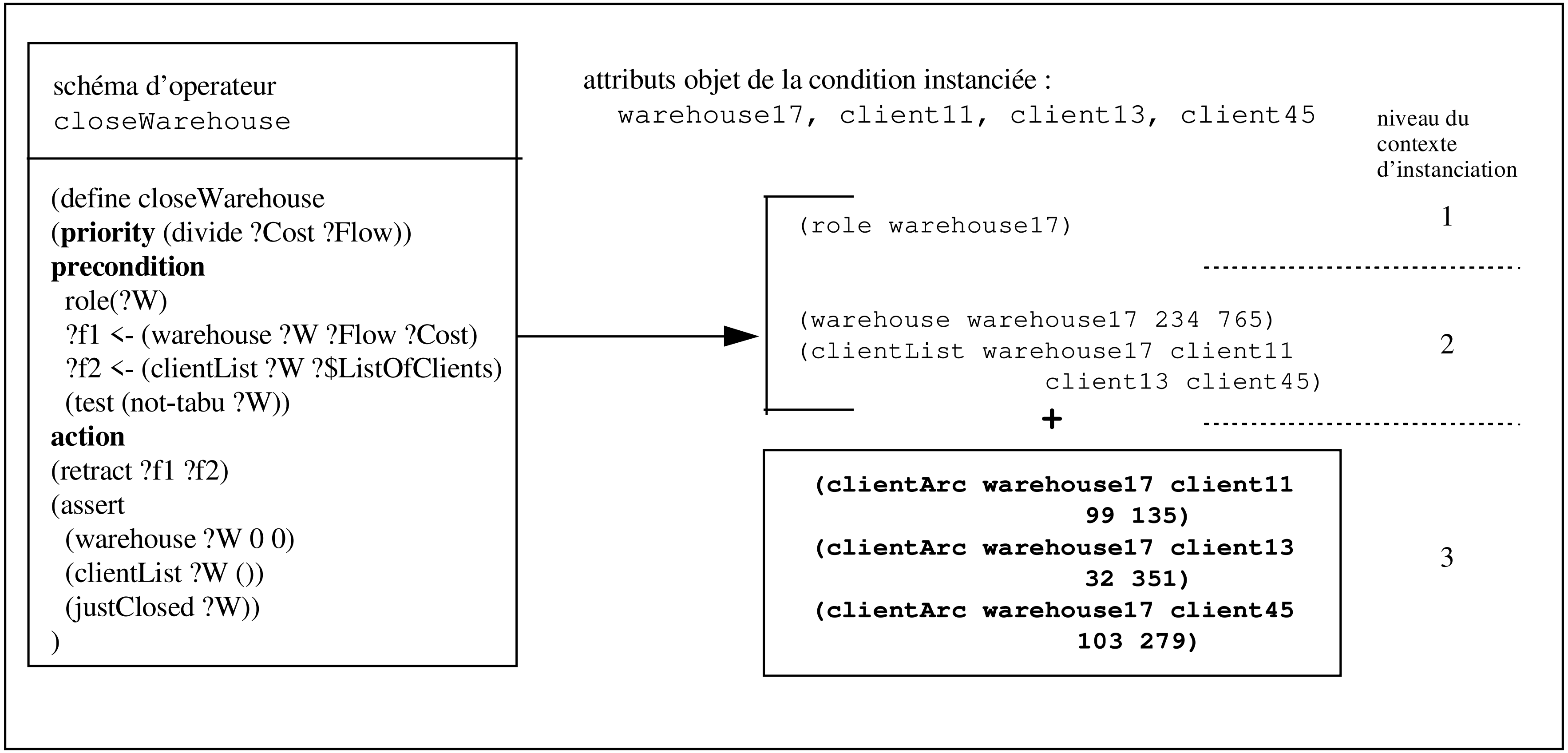


schéma d'operateur `closeWarehouse`

```
(define closeWarehouse
(priority (divide ?Cost ?Flow))
precondition
  role(?W)
  ?f1 <- (warehouse ?W ?Flow ?Cost)
  ?f2 <- (clientList ?W ?$ListOfClients)
  (test (not-tabu ?W))
action
(retract ?f1 ?f2)
(assert
  (warehouse ?W 0 0)
  (clientList ?W ())
  (justClosed ?W))
)
```

attributs objet de la condition instanciée :
`warehouse17, client11, client13, client45`

niveau du contexte d'instanciation

1

```
(role warehouse17)
```

2

```
(warehouse warehouse17 234 765)
(clientList warehouse17 client11
               client13 client45)
```

+

3

```
(clientArc warehouse17 client11
                99 135)
(clientArc warehouse17 client13
                32 351)
(clientArc warehouse17 client45
               103 279)
```

Figure 11. Le contexte d'instanciation pour le schéma d'opérateur `closeWarehouse`

Récemment, une technique de mise en forme, semblable à celle proposée ici, a été employée dans *RIBL* [Emde & Wettschereck 1996] afin de construire la description concise et caractéristique d'une situation à partir d'une description d'état beaucoup plus large. Les auteurs préconisent dans *RIBL* une profondeur de niveau 2, ce qui correspond dans notre définition à un contexte d'instanciation de niveau 3[10]. Ceci nous parait être un bon compromis. Nous l'avons donc retenu dans l'ensemble des expérimentations figurant dans cette thèse.

c) La représentation des cas

Au début de ce chapitre, nous avons indiqué qu'un *cas* est composé du contexte relatif à l'application de l'opérateur qu'il représente, et de la récompense attribuée à cette application. Ayant déjà décrit la récompense, nous sommes maintenant en mesure de présenter le contexte.

De prime abord, le contexte du cas est basé sur le contexte d'instanciation de l'opérateur dont le cas représente l'application. Cependant, pour des raisons opérationnelles concernant l'implantation du calcul de la similarité, les attributs objet du contexte des cas sont renommés par rapport au contexte d'instanciation de l'opérateur (fonction *Paramètre* du tableau 10, ligne 3, cf. p. 84). Lors de la réutilisation du cas, il n'y a ainsi pas de problème de confusion de noms entre le contexte du cas et le contexte d'instanciation de l'opérateur applicable. La figure 12 représente le cas qui pourrait être produit en suivant l'exemple ci-dessus.

St$_1$:	`(role W1)`	Bonus
St$_2$:	`(warehouse W1 234 765)` `(clientList W1 C1 C2 C3)`	

St$_3$:	`(clientArc W1 C1  99 135)`
	`(clientArc W1 C2  32 351)`
	`(clientArc W1 C3 103 279)`	

Figure 12. Un cas de `closeWarehouse`

La représentation d'un cas ne contient pas l'opérateur lui-même. Inclure l'opérateur n'apporterait pas d'information pertinente, puisque dans une exploration de l'espace

---

[10] RIBL ne connaît pas les littéraux `role`.

de recherche, telle que pratiquée par une méthode tabou, le même opérateur ne s'applique le moins de fois possible.

Cependant, le cas est en rapport avec d'autres opérateurs appartenant au même *schéma d'opérateur*. En outre, la réutilisation d'un cas pour l'évaluation d'un opérateur relevant d'un autre schéma d'opérateur serait dénué de sens. Pour illustrer ce propos, considérons l'exemple d'ordonnancement (cf. p. 181). La réutilisation d'un cas, créé à partir d'un opérateur `enlève`, ne nous parait pas nécessairement être intéressante pour l'évaluation d'un opérateur du schéma `place`. *Alois* comporte en conséquence une mémoire de cas indépendante pour chaque schéma d'opérateur. Il n'est donc pas nécessaire d'inclure le schéma d'opérateur dans un cas, son lieu de mémorisation étant suffisamment discriminatif.

Conclusion

Dans ce chapitre, nous avons présenté l'algorithme d'optimisation de *Alois*. Il procède à une exploration de l'espace de recherche en recourant à la méthode tabou. L'exploration, elle-même, est contrôlée par une technique d'intensification faisant appel au raisonnement à partir de cas. L'originalité de l'approche réside en deux points. Concernant la méthode tabou, cette approche propose une intensification qui est indépendante du problème d'optimisation, mais qui se base directement sur la modélisation de ce dernier, par l'utilisateur. Pour ce faire, *Alois* inclut un langage de représentation des problèmes d'optimisation basé sur la logique d'ordre 1.

S'agissant du raisonnement à partir de cas, l'application de ce dernier dans *Alois* comme méthode d'optimisation ne porte pas sur un problème de réparation, mais est intégrée dans une technique qui cherche de manière autonome des bonnes solutions. Ceci distingue *Alois* des autres systèmes d'optimisation à partir de cas tels, que *Cabins* ou *Ease*. Si à ce stade des développements, les grandes lignes de notre intégration du raisonnement à partir de cas dans la méthode tabou sont présentées, il nous reste néanmoins à évoquer la mémoire de cas et le calcul de la similarité.

Chapitre 4.

La mémoire de cas

La notion de mémoire de cas est étroitement liée à celle de similarité entre cas. En effet, la mémoire de cas n'a pas pour seule tâche d'emmagasiner l'expérience dont dispose le système, elle doit également faciliter à ce dernier l'accès à cette expérience. Or, le système demande, dans le seul but de la réutilisation, l'expérience en mémoire qui est la plus *proche* du problème cible. Cette ressemblance entre un problème cible et un cas est établie par une mesure de similarité qui peut être soit explicitement calculée, soit présentée de manière implicite dans l'organisation même de la mémoire. Notre système utilise une représentation des problèmes qui est basée sur la logique d'ordre 1 (chapitre 3). Ceci est peu commun en raisonnement à partir de cas où la plupart des systèmes se réfèrent à des représentations équivalentes à la logique des propositions. Or, en anticipant nos développements sur la similarité et l'appariement du chapitre 5 (cf. p. 127), nous pouvons déclarer que l'évaluation de la similarité entre deux descriptions en ordre 1 nécessite un appariement des éléments de ces descriptions [Ganascia 1987]. La mémoire de cas doit en conséquence disposer d'une mesure de similarité, explicite ou implicite, qui intègre l'appariement.

Une deuxième exigence, à laquelle doit satisfaire la mémoire, porte sur sa performance opérationnelle. En effet, une mémoire de cas, surtout quand elle comporte un nombre important de cas, doit assurer que les opérations de récupération et de mémorisation de cas soient réalisées aussi efficacement que possible. Cette contrainte est particulièrement importante dans notre système. En explorant l'espace de recherche par une méthode de recherche locale, la mémoire ne doit pas empêcher celle-ci d'engendrer le maximum d'états solutions dans le laps de temps de calcul imparti. En outre, le nombre de cas acquis par *Alois* augmente proportionnellement avec le nombre d'états visités.

L'objet de ce chapitre est de présenter la mémoire de cas de notre système. Nous approfondirons, dans la première section, la corrélation qui existe entre l'organisation d'une mémoire de cas et la mesure de similarité qu'elle emploie. Au travers d'un bref aperçu des principaux types de mémoires de cas, nous pourrons constater qu'aucune ne satisfait entièrement aux besoins de notre système. Ceci nous conduira, dans la deuxième section, à introduire une nouvelle forme d'organisation de la mémoire de cas basée sur un arbre de prototypes dynamique.

4.1. La relation entre l'organisation de la mémoire et la similarité

Nous venons de constater que la mémoire de cas dispose d'une *mesure de similarité* afin de pouvoir déterminer le degré de ressemblance entre chacun des cas en mémoire et le problème cible. De cette manière, elle pourra mettre à disposition du processus d'adaptation les cas qui sont le plus *similaires* au problème cible. Deux approches concernant cette mesure de similarité ont été développées [Porter 1989]. La première, que l'on dit "basée sur la représentation", estime que les cas contenant des attributs communs se ressemblent. La deuxième, dite de "similarité par calcul", utilise une fonction d'évaluation de similarité explicite.

Une mémoire de cas peut contenir un grand nombre de cas. Il s'avère alors fastidieux de comparer chaque cas avec le problème cible. Afin d'éviter ceci, la mémoire est souvent organisée. Les cas qui se ressemblent sont regroupés ; ainsi, lors de la récupération, le système accède rapidement au groupe de cas le plus proche du problème cible. Dans une deuxième phase, il identifie dans ce groupe le cas qui lui parait le plus semblable au problème cible (le *tri* ou le "*ranking*"). Cette approche est à la recherche d'un compromis entre le temps passé à la recherche du cas le plus proche et le temps supplémentaire nécessaire à l'adaptation d'un cas qui ne serait *pas* le plus proche [Kolodner 1993].

La mesure de similarité employée par un système conditionne l'organisation de sa mémoire de cas[1]. La dichotomie existant en matière de méthode d'évaluation de la similarité trouve son pendant dans les deux grands principes d'organisation de la mémoire. Les similarités basées sur la représentation permettent de recourir à des formes d'organisation de la mémoire regroupant les cas en fonction des attributs qu'ils partagent. L'approche basée sur les similarités par calcul débouche, elle, sur d'autres formes d'organisation, où plus d'importance est attribuée à la conformité de la mesure de similarité au domaine d'application du système.

Cette section présente ces deux familles de mémoire de cas. Le premier paragraphe traitera de l'organisation de la mémoire dans le cadre des similarité basées sur la représentation. Le deuxième paragraphe sera lui consacré aux mémoires de cas basées sur les similarités par calcul. Le troisième paragraphe enfin abordera le problème d'utilité, commun à toutes les mémoires de cas indépendamment de leur forme d'organisation.

---

[1] C'est tout autant vrai dans le sens inverse.

4.1.1. Les mémoires basées sur la représentation

La première forme d'organisation de la mémoire de cas repose sur une *similarité basée sur la représentation* (“*representational similarity*”). Cette approche se fonde sur deux hypothèses. D'une part, elle suppose que la similarité est fonction de la représentation des cas. En conséquence, la similarité augmente avec le nombre d'attributs que les descriptions de cas ont en commun. D'autre part, selon cette approche, il n'est pas nécessaire de récupérer le cas le plus similaire au problème cible, mais il suffit que le cas soit *suffisamment* proche du problème cible pour que le processus d'adaptation réussit à construire une solution réalisable.

a) Les réseaux de discrimination

A partir de ces deux hypothèses, il est possible d'organiser la mémoire de cas de manière à regrouper les cas qui partagent des attributs. L'accès à ces groupes de cas se fait par des *index* qui sont discriminants pour chaque groupe. Les techniques classiques de construction de ce type de mémoires sont les *réseaux de discrimination*[2] [Feigenbaum 1963] tels que la *mémoire dynamique* [Schank 1982], les *arbres de décision* [Quinlan 1986] ou les *arbres k-d* [Bentley 1975, Wess & al. 1993].

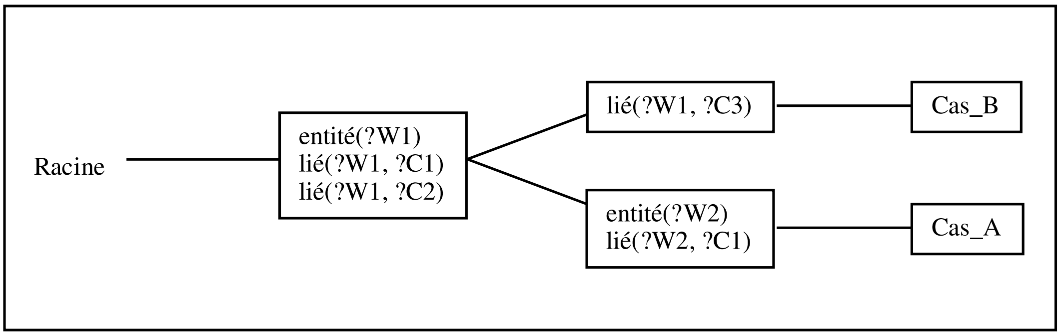


Figure 13. Exemple de mémoire de cas basée sur un réseau de discrimination

Pour illustrer ce type d'organisation, la figure 13 présente deux cas, basés sur les deux situations du problème de placement et d'allocation d'entités, telles que présentés à la figure 9 et au tableau 15 (cf. p. 88 et p. 90), insérés dans une mémoire de cas telle qu'employée dans *Prodigy/Analogy* [Veloso 1994], l'un des rares système de raisonnement à partir de cas qui recourt à un langage d'ordre 1 pour la représentation des cas.

---

[2] Voir [Charniak & al. 1987] pour la mise en oeuvre de réseaux de discrimination.

L'organisation de la mémoire sous forme de réseau de discrimination accélère les opérations de récupération et de mémorisation. Il n'est en effet plus nécessaire de comparer le problème cible à chaque cas en mémoire. Il suffit, en commençant par le sommet racine du réseau, de comparer le problème cible à chaque fils et de choisir comme successeur celui avec lequel la cible partage le plus d'attributs. En raison du réseau de discrimination, l'efficacité de l'interaction avec la mémoire, quant à elle, ne subit pas de détérioration importante lorsque le nombre de cas augmente [Veloso 1994]. Un réseau de discrimination peut avoir la forme d'un arbre (cf. figure 13) ou d'un treillis. Ce dernier présente l'avantage d'offrir plusieurs chemins d'accès redondants à un même cas. Le résultat de la récupération dépend en conséquence moins de l'ordre dans lequel les branchements sont effectués [Kolodner 1993].

Cette organisation de la mémoire donne la priorité à l'efficacité de la récupération sur sa précision. En effet, en organisant les cas en un réseau de discrimination, il n'est plus certain que lors de la récupération le cas le plus proche soit vraiment identifié, même si l'on emploie la stratégie du *ranking*. La récupération peut même s'arrêter sur un cas qui n'est que difficilement adaptable. Une manière de pallier à ce problème consiste à intégrer les connaissances d'adaptation dans la mesure de similarité. Ainsi, la récupération identifie essentiellement des cas qui sont bien adaptables [Smyth & Keane 1995a].

A première vue, une mémoire basée sur un réseau de discrimination parait bien s'intégrer dans le système *Alois*. En effet, les cas dans *Alois* emploient une représentation explicite de leur situation représenté par le contexte d'instanciation. Le recours à une mesure de similarité basée sur cette description n'est donc pas dépourvu de sens. En outre, il est quasiment certain que la mémoire de *Alois* contient un grand nombre de cas, acquis de manière incrémentale, ce qui s'avère également tout à fait compatible avec ce type d'approche, comme le confirme d'ailleurs la mémoire de *Prodigy/Analogy*.

Toutefois, deux problèmes se posent, celui de l'évaluation de la similarité qui fait obligatoirement intervenir l'appariement entre attributs, et celui des attributs numériques. Le système *Prodigy/Analogy* détermine la similarité entre le cas récupéré et la cible au fur et à mesure qu'il parcourt le réseau de discrimination. Il commence au niveau de la racine de la mémoire et descend itérativement jusqu'à atteindre une feuille. A chaque niveau du réseau, il construit un appariement entre les attributs présents dans la description cible et ceux présents dans le noeud du réseau. Or, ces appariements sont construits de manière exhaustive, systématique et complète. Le système s'arrête après avoir trouvé un appariement qui lui parait être de bonne qualité. Selon l'ordre dans lequel ces appariements sont construits, ceci peut arriver plus ou moins tard.

Si à ce stade, on peut imaginer qu'il serait possible de trouver une heuristique qui, en contrôlant la construction des appariement, accélérerait ce processus, un deuxième problème, relatif aux *attributs numériques* vient néanmoins compliquer les choses. Considérons la description du cas présenté dans la figure 12 (cf. p. 99). Il se distingue de ceux illustrant la mémoire de *Prodigy/Analogy* par la présence d'attributs numériques dans la description du contexte[3]. Pour qu'un réseau de discrimination puisse prendre en considération les attributs numériques, il y a deux manières de faire : discrétiser ou saturer (les considérer tels quels mènerait à un réseau de discrimination extrêmement large).

Dans la *discrétisation* [Fayyad & Irani 1993] d'attributs numériques, le domaine des valeurs des attributs est partitionné, et leurs valeurs sont caractérisées par l'intervalle auquel elles appartiennent. Le point important de toute partition réside au niveau des valeurs limites des intervalles. Soit ces valeurs sont données par l'utilisateur, ce que nous écartons d'office, soit elles sont déterminées par le système lui-même. Dans ce cas, une analyse préalable des valeurs des attributs numériques est nécessaire afin que le système puisse déterminer par lui-même les valeurs limites de la partition. Or, cette analyse requiert que les cas soient préalablement disponibles, ce qui n'est pas le cas dans le système *Alois*.

La *saturation* [Ganascia 1987] d'attributs numériques consiste à introduire l'ensemble des valeurs du domaine d'un attribut dans la description du cas. Prenons comme exemple un cas contenant l'attribut x, défini sur le domaine des entiers dans $[0\ldots10]$ et qui ait la valeur $x = 5$. A la description de ce cas serait ajouté les descripteurs $x > 1$, $x > 2$, … $x < 10$. Le réseau de discrimination resterait probablement d'une taille restreinte, mais la description des cas pourrait rapidement devenir large. Pour ces diverses raisons, nous n'envisageons pas l'utilisation d'une mémoire du type réseau de discrimination dans notre système.

b) La factorisation des cas

Une autre organisation de la mémoire, basée sur la représentation des cas, s'appuie sur une représentation graphique des cas, dans laquelle les attributs sont les sommets et les littéraux les arcs [Bunke & Messmer 1993]. L'approche consiste à *factoriser* ces graphes en arcs puis à partir de ceux-ci à construire un réseau de jointures à la façon du réseau Rete [Forgy 1982]. La figure 14 présente un tel réseau en employant les mêmes cas que ci-dessus. A l'entrée du réseau se trouvent les noeuds représentant les deux attributs. Les deux noeuds intermédiaires décrivent des sous-graphes qui sont communs aux deux cas, tandis que les deux noeuds terminaux représentent les cas eux-mêmes.

Les performances d'une mémoire de ce type sont excellentes en ce qui concerne la récupération. L'identification d'un cas qui représente un sous-graphe du problème cible

---

[3] Rappelons que Prodigy/Analogy ne reconnaît pas les attributs numériques.

est d'une complexité réelle constante, c'est à dire indépendante du nombre de cas en mémoire. Cependant, la méthode présente divers inconvénients. Premièrement, la construction du réseau de jointures est un processus lourd. La méthode ne se prête donc pas à une acquisition incrémentale de la base de cas telle que nous l'envisageons dans *Alois*. Deuxièmement, la méthode identifie des cas qui sont des sous-graphes exacts du problème cible. Lorsqu'elle est étendue au sous-graphes inexacts (comme nous le ferons au chapitre 5) la méthode s'avère beaucoup moins efficace [Messmer & Bunke 1993]. En dernier lieu, l'incorporation d'attributs numériques pose ici le même problème qu'en matière des réseaux de discriminations (cf. ci-dessus). Pour ces trois raisons, nous n'introduisons pas ce type de mémoire dans le système *Alois*.

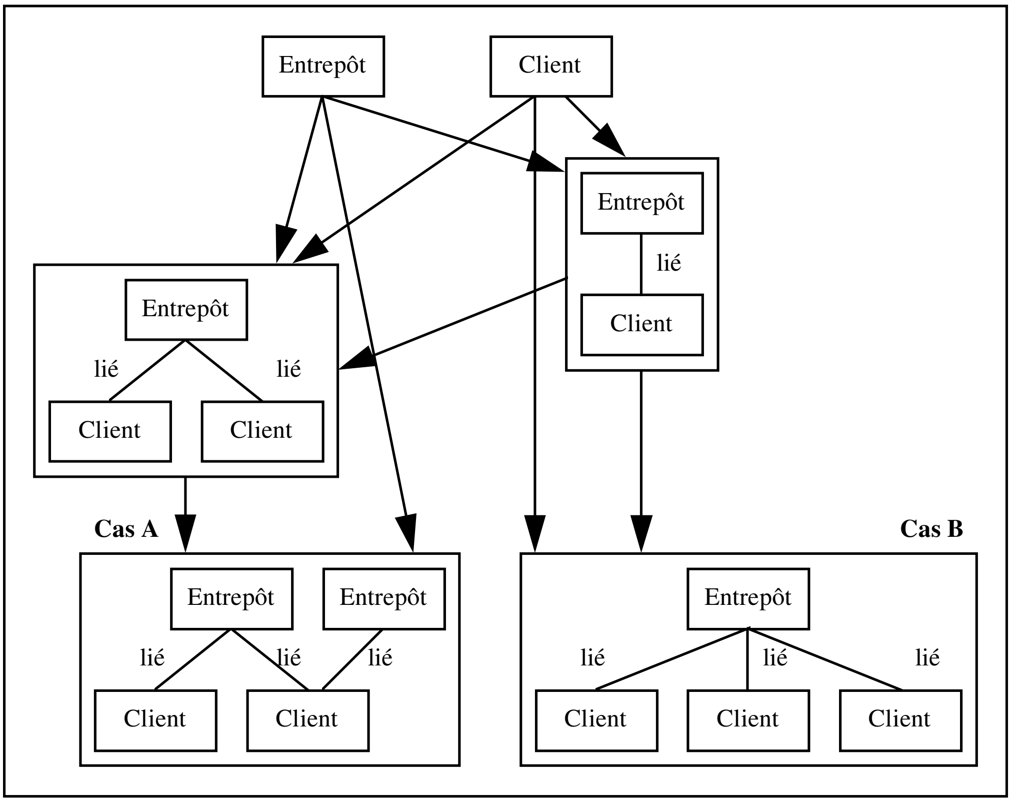


Figure 14. Une mémoire du type réseau de cas factorisés

c) Les hiérarchies de cas

Une troisième organisation de la mémoire de cas, basée sur la représentation des cas, consiste à disposer les cas en une *hiérarchie* [Levinson 1984, Lieber & Napoli 1996, Ellis 1993]. Seule condition, la méthode requiert l'existence d'une relation *d'ordre partiel*, notée π, entre les cas. Si cette approche ne fait a priori pas intervenir la représentation des cas, l'ordre partiel est néanmoins souvent défini sur leur espace de descriptions.

La figure 15 présente une mémoire de cas qui utilise une telle hiérarchie. Les cas sont organisés en un treillis. A la différence de l'approche précédente, chaque noeud d'une hiérarchie est lui-même un cas. Notons *C* l'ensemble des cas en mémoire. Pour chaque cas c, $c \in C$, son ensemble de fils F_C est défini par :

$$F_c = \left\{ c' \middle| c' \in C, c \,\pi\, c', \neg\exists c'' \in C, c \,\pi\, c'' \,\pi\, c' \right\} \qquad (21)$$

La construction d'une hiérarchie de cas est simple. Il suffit d'insérer un cas après l'autre de manière à ce qu'il respecte l'ordre partiel vis à vis de ses parents et de ses fils. La récupération d'un cas consiste en une opération d'insertion, dont les parents et les fils constituent le résultat. Une implantation de cette mémoire dans le domaine des échecs indique que l'opération d'insertion est efficace [Ellis 1993].

Par rapport au système *Alois*, il nous parait important de noter que la subsomption est l'ordre partiel entre deux descriptions quand elles sont représentées en ordre 1. Son évaluation fait intervenir une phase d'appariement [Ganascia 1987]. Or, le problème d'appariement, comme nous le verrons plus loin, est NP-complet (cf. p. 127). En conséquence, l'utilité de ce type de mémoire nous parait être limitée pour ce qui concerne notre système.

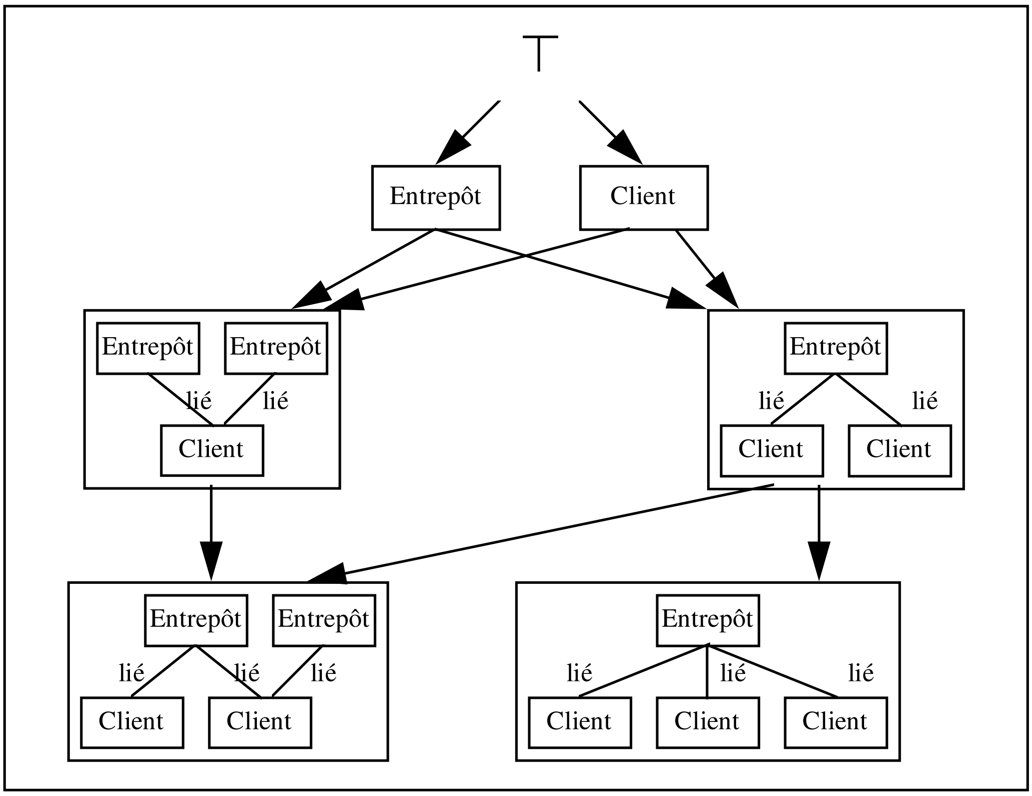


Figure 15. Une mémoire de cas hiérarchique

4.1.2. Les mémoires du type plus proches voisins

La seconde forme d'organisation de la mémoire de cas emploie une fonction explicite d'évaluation de la similarité pour établir la ressemblance entre deux descriptions de cas. Appelée *évaluation de la similarité par calcul* (“*computational similarity*”), cette approche se fonde sur les méthodes des plus proches voisins [Aha & al. 1991, Dasarathy 1991]. L'organisation sous-jacente de la mémoire est plutôt simple. En effet, ces techniques n'utilisent que des mémoires de cas séquentielles qui alignent les cas les uns derrière les autres dans une simple liste. Lors de la récupération, toute la mémoire est parcourue afin d'identifier le cas le plus proche. Il en va ainsi lors de l'insertion de cas, puisqu'il s'agit là de s'assurer que le nouveau cas accroît réellement les connaissances du système.

Cette approche présente donc l'avantage de garantir l'identification du cas, qui est effectivement le plus proche à la cible. Néanmoins, et sauf pour les systèmes particuliers qui emploient des ordinateurs massivement parallèles [Stanfill & Waltz 1986], elle entraîne une dégradation de l'efficacité des interactions avec la mémoire lorsque cette dernière contient un nombre important de cas.

a) Le principe

Introduites au début des années cinquante [Fix & Hodges 1951], les méthodes des *plus proches voisins* s'appliquent essentiellement aux problèmes de catégorisation. Malgré leur simplicité apparente, ces méthodes donnent souvent des résultats d'une précision et d'une robustesse remarquables. La communauté d'apprentissage symbolique automatique s'y intéresse depuis quelques temps [Aha & al. 1991], maintenant. Cet intérêt tardif s'explique certainement par l'absence de capacité de généralisation, inhérente à ces méthodes.

Les méthodes des plus proches voisins se basent sur un ensemble d'exemples pré-classés, appelé ensemble d'apprentissage, et sur une mesure de distance. Lorsque survient un nouvel exemple à classer, on recherche, parmi les exemples de l'ensemble d'apprentissage, les *k* exemples qui sont les plus proches, en termes de mesure de distance. A l'exemple à classer est ensuite attribuée la classe désignée par vote majoritaire des *k* exemples les plus proches.

Cet algorithme de base a évolué à maintes reprises et de différentes manières. Les présenter ici dépasserait le cadre que nous nous sommes posé[4]. Cependant, indiquons que deux grands axes de perfectionnement se sont dégagés. L'un tente d'augmenter la précision de catégorisation, et l'autre de diminuer la complexité de l'opération de la

---

[4] Dasarathy (1991) en fait un exposé très complet

catégorisation. Ce dernier point est le point faible des méthodes des plus proches voisins, puisque la complexité de catégorisation est proportionnel au nombre d'exemples qui sont nécessaires à l'apprentissages des concepts. Si ce nombre est important, la vitesse de catégorisation se dégrade.

Ces deux axes ne sont pas nécessairement contradictoires. Pour preuve, une approche appelée “correction de l'ensemble d'apprentissage” (“*edited data*”) permet d'améliorer ces méthodes dans les deux sens. Nous l'examinons ci-dessous, puisqu'elle a récemment retenue l'attention des chercheurs dans le domaine de l'apprentissage symbolique. Mais d'abord, nous étudierons les mesures de distance qui sont souvent employées dans les méthodes des plus proches voisins.

b) Les mesures de distance

Soit *dis* une *mesure de distance* et A, B et C des exemples. Les mesures de distance peuvent avoir pour caractéristiques la *minimalité*, la *symétrie* et *l'inégalité triangulaire* :

Minimalité $$dis(A, B) \geq dis(A, A) = 0 \tag{22}$$

Symétrie $$dis(A, B) = dis(B, A) \tag{23}$$

Inégalité triangulaire $$dis(A, B) + dis(B, C) \geq dis(A, C) \tag{24}$$

On appelle *indice de distance* la mesure de distance qui satisfait (22) et (23), et *métrique* celle qui satisfait (22) - (24)[5]. Il est intéressant de noter ici que les humains n'utilisent pas des mesures de distance symétriques. Ainsi, une ellipse est perçue comme plus similaire à un cercle qu'un cercle à une ellipse [Tversky 1977]. De même, le principe de minimalité n'est pas nécessairement cognitivement plausible [Biberman 1994]. A priori, les techniques de plus proches voisins ne sont pas liées à un type particulier de mesure de distance. Pour des raisons de “compréhensibilité” des classifications, on fait souvent appel pour le moins à des indices de distances.

En raisonnement à partir de cas, on emploie le plus souvent une mesure de similarité, et non pas une distance. Le passage de l'une à l'autre est cependant direct et peut être réalisé de plusieurs manières. Ainsi, Bisson (1993) propose

$$dis(A,B) = \left|S_{\max} - sim(A,B)\right|, \tag{25}$$

---

[5] Notons que si on emploie une mesure de distance qui est une métrique, l'algorithme *Aesa* ramène l'identification de l'élément (le plus) proche se trouvant en mémoire à une complexité constante [Vidal 1986, 1994]. Il pourrait d'ailleurs être intéressant de concevoir une mémoire de cas à partir de cette technique.

où S_{max} est la valeur maximale de la similarité, tandis que Aha & al. (1991) utilisent

$$sim(A,B) = -\sqrt{\sum_{i=1...n} dis(A_i,B_i)}, \qquad (26)$$

où A_i et B_i sont respectivement les attributs de A et B.

En apprentissage symbolique et en raisonnement à partir de cas, dans le cadre de l'*apprentissage à partir d'instances*, on utilise le plus souvent des fonctions de similarité basées sur la *distance euclidienne* pondérée par la dimension β (27), avec $\beta = 2$ [Aha & al. 1991, Salzberg 1991, Wettschereck & Aha 1995].

$$D_\beta(A,B) = \left(\sum_{i=1}^{p} w_i \cdot (d_i)^\beta \right)^{\frac{1}{\beta}} \qquad (27)$$

$$\text{avec :} \quad d_i = \begin{cases} A_i \neq B_i, & si \text{ nominal} \\ \left|A_i - B_i\right|, & si \text{ numé rique} \end{cases} \qquad (28)$$

La distance euclidienne requiert que les exemples soient représentés dans un formalisme basé sur la logique propositionnelle. Ce n'est que récemment qu'elle a été modifiée afin de prendre en compte des structures représentées en logique de prédicats [Emde & Wettschereck 1996].

c) Les ensembles d'apprentissage corrigés

Les techniques de *correction des ensembles d'apprentissage* réduisent le nombre d'exemples d'apprentissage tout en augmentant la capacité de la catégorisation. Elles peuvent être classées en deux familles. La première, et la plus ancienne [Hart 1968, Gates 1972], consiste à n'utiliser lors de la catégorisation qu'un sous-ensemble de l'ensemble d'apprentissage. L'ampleur de la phase de recherche des voisins proches se trouve donc automatiquement réduite. Ces techniques se basent sur l'hypothèse selon laquelle il existe des groupes d'exemples appartenant à la même classe. Un tel groupe peut alors être représenté avec un nombre réduit d'exemples sans pour autant diminuer la précision de la catégorisation. Récemment encore, Aha & al. (1991) confirme que l'on obtient des bons résultats avec une telle approche. Nous allons ci-dessous considérer la technique de Hart (1968), appelée *condensation*, dans le cadre de la présentation de l'organisation de la mémoire de cas du système *Alois* (cf. p. 117).

La deuxième famille d'approches traite de la modification de la mesure de distance en vue d'optimiser la précision de la catégorisation. Elle part du principe que les différents attributs sont d'une importance variable pour la catégorisation. De manière générale, on distingue les méthodes de correction *globale* de celles de correction *locale* [Myles & Hand 1990, Dasarathy 1991]. Les méthodes globales modifient la mesure de distance sur l'ensemble de l'espace des exemples, alors que les méthodes locales la modifient de manière différente selon les régions de l'espace. Les techniques de correction des ensembles d'apprentissage ont récemment retenu l'attention en apprentissage symbolique, au sein des deux axes de recherche que sont la sélection d'attributs et la pondération d'attributs.

La *sélection d'attributs* traite du problème d'attributs qui ne sont pas nécessaires à la catégorisation et qui peuvent même en altérer la précision. Il s'agit alors d'identifier ces attributs et de les ignorer lors de la catégorisation [Aha & Bankert 1995]. Différentes approches, surtout du type global, ont été proposées, telles que la sélection d'attributs par l'utilisation d'un arbre de décision du type ID3 [Cardie 1993], ou par le recours à une procédure de recherche locale aléatoire [Skala 1994]. Ces approches ont un rôle important dans les domaines où les exemples sont composés d'un nombre important d'attributs [Moulinier & al. 1996]. Elles permettent de focaliser l'apprentissage sur les attributs prépondérants, et d'accélérer en même temps l'apprentissage. En ce qui concerne le système *Alois*, on peut a priori nier l'existence d'attributs inutiles. Au contraire, l'utilisateur a la possibilité, grâce au langage de représentation des problèmes, de définir lui-même la description des exemples, qui sont ici les situations des cas.

La *pondération des attributs*, quant à elle, améliore la performance de la catégorisation en modifiant la mesure de distance. Elle vise essentiellement les poids w_i de l'équation (27) qui sont adaptés afin de tenir compte de l'importance des différents attributs [Wettschereck & al. 1996]. Par ailleurs, ce type de technique peut permettre d'importantes réductions de la taille de l'ensemble d'apprentissage. Intuitivement, en adaptant les poids, un exemple peut à lui tout seul couvrir une bonne partie du groupe d'exemples auquel il appartient. Ainsi Ricci & Avesani (1996) font état d'élagages de l'ensemble d'apprentissage pouvant atteindre 90%, sans pour autant diminuer la précision de la catégorisation. Les gains en termes de vitesse de catégorisation sont alors conséquents.

Toutefois, les techniques de correction des ensembles d'apprentissage présentent deux inconvénients majeurs. D'une part, leur mise en oeuvre nécessite un pré-traitement qui est d'une complexité non négligeable puisqu'il considère tout un espace de mesures de similarités. D'autre part, ces techniques ne s'appliquent qu'à des problèmes de catégorisation. En effet, elles exploitent la redondance qu'il y a dans les ensembles d'apprentissage où chaque classe est représentée par plusieurs exemples. Ceci rend ces méthodes inadaptées au raisonnement à partir de cas où les cas ne font pas partie de classes (voire à l'extrême, où chaque cas représente sa propre classe). Ce moyen ne permet donc pas à réduction de la taille de l'ensemble des cas. Pour ces deux raisons, nous n'incluons pas d'algorithme de correction de l'ensemble d'apprentissage dans le système *Alois*.

4.1.3. Le problème d'utilité

Le *problème d'utilité* concerne les systèmes ayant recours aux techniques d'apprentissage pour améliorer leurs performances et/ou leurs compétences, mais dont il s'avère que les performances se dégradent au fur et à mesure que des connaissances sont apprises. Si ce problème se manifeste tout particulièrement en apprentissage analytique (chapitre 2, cf. p. 53), il n'en est pas moins présent en raisonnement à partir de cas [Francis & Ram 1993]. En effet, un système qui acquiert des cas afin d'enrichir son expérience peut être confronté à une dégradation de la vitesse d'interaction avec la mémoire (récupération et mémorisation) en raison de l'augmentation du nombre de cas en mémoire.

Le problème d'utilité n'est, a priori, pas lié à la forme d'organisation de la mémoire de cas, même si un système se heurtera plus ou moins tard à ce problème en fonction du type d'organisation choisi. L'utilisation d'ordinateurs massivement parallèles pour procéder à l'interrogation de la mémoire, devrait en principe conduire à une récupération en temps constant [Stanfill & Waltz 1986]. Cependant, il s'avère qu'elle ne fera que retarder l'apparition du phénomène, sans pour autant l'annihiler [Francis & Ram 1993]. La seule solution au problème d'utilité consiste alors en l'adoption de stratégies d'acquisition sélective de nouveaux cas [Smyth & Keane 1995b].

Dans le modèle général du raisonnement à partir de cas, un nouveau cas est acquis lorsqu'un cas récupéré doit être profondément transformé pendant la phase d'adaptation (cf. p. 58) [Hammond 1986, Kolodner 1993]. L'acquisition d'un nouveau cas, rendant compte de cette adaptation, le système devrait en être dispensé à l'avenir. Cependant, cette stratégie ne suffit pas pour écarter le problème d'utilité. Il est préférable de s'assurer de l'originalité d'un nouveau cas par rapport aux cas déjà présents dans la mémoire. Un cas est alors ajouté seulement s'il se révèle suffisamment original pour pouvoir prendre part à une augmentation de la compétence du système[6] [Smyth & Keane 1995b].

---

[6] Le travail de Smyth & Keane (1995b) concerne l'effacement de cas d'une mémoire affectée par le problème d'utilité. Comme les auteurs le laissent entendre, leur technique s'étend au processus d'acquisition de nouveaux cas et constitue alors un moyen d'éviter le problème.

4.2. La mémoire de cas de *Alois*

Dans la section précédante nous avons évoqué les avantages et les limites des deux principales organisations de la mémoire de cas. On a pu constater que les mémoires basées sur la représentation des cas sont intéressantes en raison de leur organisation, reposant sur une indexation selon attributs communs, qui permet la prise en compte d'un nombre important de cas. Les mémoires du type plus proches voisins se sont, quant à elles, avérées dignes d'intérêt de par la simplicité de leur mise en oeuvre, surtout grâce à l'incorporation d'une mesure de similarité explicite. Toutefois, face à un nombre de cas important, elles perdent de leur attrait.

Le système *Alois* a besoin d'une mémoire de cas qui allie les avantages de ces deux approches. La mémoire de cas de *Alois* doit être rapide. En outre, elle requiert l'emploi d'une mesure de similarité explicite, puisqu'une mesure de ce type n'est guère sensible au problème de la prise en compte d'attributs numériques. Dès lors, la mémoire de cas devient un point important dans nos travaux. En effet, aucune organisation de mémoire de cas actuelle ne satisfait à l'ensemble des besoins du système.

L'objet de cette section est de présenter une organisation de la mémoire de cas originale qui repose sur les *arbres de prototypes* (“*template trees*[7]”) [Ramapriyan 1976, Brown 1995]. Appelée *arbres de prototypes dynamiques*, (“*dynamic template trees*”), elle répond aux besoins de *Alois* en combinant les avantages des deux principales approches en matière d'organisation de mémoires de cas. En effet, les arbres de prototypes organisent les cas en un arbre, ce qui accélère la récupération de cas, tout en employant une mesure de similarité explicite.

Le premier paragraphe de cette section sera consacré à la présentation des arbres de prototypes classiques, alors que le second traitera de notre extension. La mesure de similarité explicite qu'emploie *Alois* sera, quant à elle, présentée dans le prochain chapitre (cf. p. 127).

4.2.1. Les arbres de prototypes

Les arbres de prototypes sont apparus dans le domaine de la reconnaissance de caractères imprimés durant les années soixante-dix [Ramapriyan 1976]. Reconnaître un caractère consiste à placer dans une classe prédéfinie, telle que 'a', 'A', 'b', etc., une image de

---

[7] Notre traduction du mot anglais "template" par le mot "prototype" est discutable. En effet, le concept du prototype contient la notion de typicalité [Porter & al. 1990, Rifqi 1996], absente dans celui du "template".

celui-ci. Ce problème de catégorisation comporte une phase de comparaison, où l'on recherche parmi un ensemble de caractères pré-classifiés, appelés *prototypes* (“*templates*”), celui qui ressemble le plus au caractère à classer. L'objectif des arbres de prototypes est d'accélérer cette identification tout en assurant une bonne prédiction [Voisin & Devijver 1987, Brown 1995].

Les techniques d'indexation basées sur les attributs communs des caractères prototypes, telles que les arbres k-d [Bentley 1975], ne sont pas adaptées à la catégorisation de caractères imprimés. Ceci pour deux raisons [Brown 1995]. D'une part, si leur vitesse de catégorisation est très importante il n'en demeure pas moins que les descriptions des caractères contiennent des centaines d'attributs, ce qui conduit à des structures d'indexation très profondes, et donc inefficaces. D'autre part, les mesures de similarité dont on dispose sont très spécifiques du domaine et leur intégration dans une technique d'indexation basée sur les attributs communs serait donc fort difficile. Les arbres de prototypes remédient aux deux problèmes et permettent d'atteindre des taux de prédiction allant jusqu'à 90% et plus [Brown 1995].

a) Description des arbres de prototypes

Les arbres de prototypes traitent du problème de la *catégorisation*, où le système doit prédire l'appartenance d'un *objet* o_c, d'un *univers* O, $o_c \in O$, à un sous-ensemble c de O, $c \subseteq O$, c étant une *classe* prédéfinie. Il existe en général plusieurs classes qui ne sont pas nécessairement disjointes. A l'image des techniques des plus proches voisins, une fonction d'évaluation de la similarité entre deux objets est requise. Sans limiter la généralité de la présentation, nous supposons que cette fonction de similarité `sim` évalue la ressemblance de deux objets sur une échelle absolue, dans l'intervalle [0…1] (29), avec 1 pour deux objets identiques et 0 pour deux objets complètement différents. En effet, il serait tout aussi possible d'employer des mesures de similarité relatives qui établissent un ordre de ressemblance sur un ensemble fini d'objets [Richter 1992].

$$\mathtt{sim} : O \times O \rightarrow [0 \ldots 1] \tag{29}$$

Les objets préclassifiés o_i sont organisés dans un arbre (figure 16). Chaque noeud *N* de l'arbre, à l'exception de la racine, contient un objet o_i et l'ensemble de classes C_i auxquelles l'objet appartient. Chaque noeud ne peut avoir plus de *max_Degree* fils. En effet, *max_Degree* est le seul paramètre d'un arbre de prototypes.

Tout noeud père ayant des fils, voit nécessairement l'un d'entre eux contenir le même objet et les mêmes classes que le père. Cette contrainte assure que tout objet de l'arbre est contenu dans l'une de ces feuilles, ce qui est important pour la récupération et la mémorisation. Mise à part cette relation père-fils, les arcs d'un arbre de prototypes n'ont pas de signification. Ils ne présentent, notamment, aucun rapport avec les objets contenus dans les noeuds, et pas davantage avec les représentations de ceux-ci. Ceci distingue les arbres de prototypes de toutes les autres techniques de catégorisation proposées en apprentissage symbolique ou de toutes les mémoires de cas basées sur la représentation (cf. p. 103). En effet, dans ces approches, soit les noeuds contiennent des descriptions partielles des objets [p. e. Quinlan 1986, Veloso 1994], soit les liens sont relatifs à l'ordre partiel entre les objets contenus dans deux noeuds adjacents [p. e. Levinson 1984], soit encore les noeuds renferment l'ensemble des attributs contenus dans l'ensemble de leurs fils [Fisher 1987].

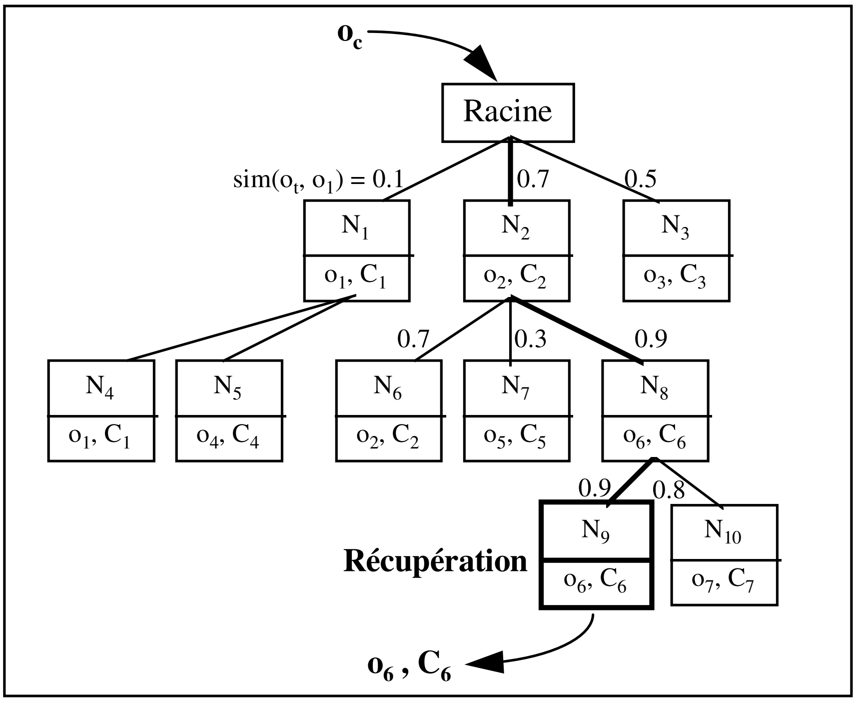


Figure 16. Un arbre de prototypes avec *max_Degree* = 3

b) La récupération dans un arbre de prototypes

Le but de la récupération est l'identification de l'objet de l'arbre le plus proche d'un objet cible o_C (tableau 18). En partant du noeud racine, l'objet contenu dans chaque noeud fils est comparé avec o_C. Le noeud fils contenant l'objet qui ressemble le plus à o_C est choisi comme successeur, et le cycle se répète en descendant progressivement l'arbre. Bien évidemment il n'est pas nécessaire de répéter le calcul de la similarité entre o_C et un objet qui a déjà été comparé à un niveau supérieur. La récupération s'arrête lorsqu'elle atteint un noeud qui est une feuille de l'arbre. Dans ce cas, l'objet contenu dans la feuille est considéré comme étant celui qui est le plus proche de o_C. En conséquence, on prédit que o_C appartient aux classes contenues dans la feuille.

Prenons l'exemple de la figure 16 relative à la récupération pour l'objet cible o_c. Tout d'abord, o_c est comparé à o_1, o_2 et o_3, car ces objets appartiennent aux fils de la racine. Supposons que les trois évaluations de la mesure de similarité donnent respectivement $\mathtt{sim}(o_c, o_1) = 0.1$, $\mathtt{sim}(o_c, o_2) = 0.7$ et $\mathtt{sim}(o_t, o_3) = 0.5$. Le noeud N_2, contenant l'objet o_2 le plus similaire à o_c, est choisi comme successeur. Lors de la seconde itération, o_c est comparé aux objets o_2, o_5 et o_6 (en ne calculant que les similarités avec o_5 et o_6). A présent o_6 est l'objet qui ressemble le plus à o_c. La récupération continue donc dans le noeud N_8, d'où elle atteint le noeud feuille N_9, qui lui aussi contient o_6. C'est donc l'objet o_6 qui est identifié comme étant le plus proche de o_c. La catégorisation prédit en conséquence que o_c appartient aux mêmes classes que o_6, c'est à dire $C_c := C_6$.

```
fonction Récupère(cible): l'objet le plus proche, et son ensemble de classes
actuel := noeud Racine
tant que (actuel n'est pas une feuille) fait
    pour tout (fils de actuel)
        calcule sim(cible, objet de fils)
        retient fils avec la similarité maximale
    fin pour tout
    actuel := le meilleur noeud fils
fin tant que
revoi objet et ensemble des classes de actuel
```

Tableau 18. Récupération dans un arbre de prototypes

c) La mémorisation dans un arbre de prototypes

Un arbre de prototypes peut être construit de manière incrémentale par une modification de la technique de *condensation* [Voisin & Devijver 1987, Brown 1995] qui a été proposée initialement par Hart (1968) pour la réduction de l'ensemble d'apprentissage dans le cadre des plus proches voisins (tableau 19). *L'apprentissage supervisée* commence avec un arbre vide. Un objet peut être inséré dans l'arbre à n'importe quel moment. La mémorisation d'un nouvel objet o_c, et de son ensemble de classes C_c, débute par une récupération de l'objet qui lui est le plus proche. Appelons N_l le noeud feuille dans lequel cette récupération se termine, et C_l l'ensemble des classes de N_l. A la manière de la technique de condensation, on compare les ensembles de classes C_c et C_l. Si les deux sont identiques, $C_c = C_l$, l'objet o_c n'est pas ajouté à l'arbre (tableau 19, linge 4). Ceci nous donne une simple technique du type “ensemble d'apprentissage corrigé” qui évite l'ajout d'objets inutiles. Toutefois, si $C_c \neq C_l$, o_c est ajouté à l'arbre. Il s'agit alors de déterminer l'endroit où o_c doit être inséré. De préférence, o_c est ajouté comme frère de N_l. Cependant, ceci n'est possible que si le père de N_l est de degré inférieur à *max_Degree*. Si tel n'est pas le cas, o_c est inséré comme fils de N_l, et le contenu de N_l est lui aussi inséré comme nouveau fils de N_l. Il est ainsi certain que tout objet de l'arbre fait partie d'un noeud feuille.

```
fonction Insère(object, classes)
si (arbre = vide)
    alors insère object et classes comme fils de racine et renvoi
NL := noeud feuille de récupère(object)
si (classes de NL = classes) alors renvoi
F := père de NL
si (degré de F < max_Degree) alors
    insère object comme fils de F et renvoi
sinon
    copie le contenu de NL vers un nouveau fils
insère object et classes comme fils de NL et renvoi
```

Tableau 19. Mémorisation dans un arbre de prototypes

Reprenons l'exemple précédant afin d'illustrer la mémorisation (figure 17) du même objet o_c et de son ensemble de classes C_c. D'abord, une récupération est effectuée pour o_c. Elle s'achève dans le noeud N_9. Si $C_6 \neq C_c$, o_c est inséré dans l'arbre. Puisque N_8, le père de N_9, est du degré 2, et 2 < *max_Degree*, un nouveau noeud N_{11}, contenant o_c et C_c, est construit et inséré comme fils de N_8.

Les arbres de prototypes suscitent quelques remarques concernant la complexité des opérations de récupération et de mémorisation, et la précision de la récupération. Nous traiterons d'abord de la complexité.

d) La complexité algorithmique des arbres de prototypes

Considérons d'abord la complexité de l'espace mémoire qu'occupe un arbre de prototypes en fonction du nombre d'objets présents dans l'arbre. Il s'agit de distinguer ici entre l'espace occupé par les objets et l'espace utilisé par la structure d'accès, c'est à dire les noeuds et les arcs de l'arbre. En effet, les objets ainsi que l'ensemble des classes dont ceux-ci font partie, peuvent être stockés dans une simple liste, et être liés aux noeuds par des pointeurs. Si la représentation des objets et de leurs classes est de taille fixe, l'espace mémoire nécessaire à leur représentation est de complexité $O(n)$, n le nombre d'objets stockés.

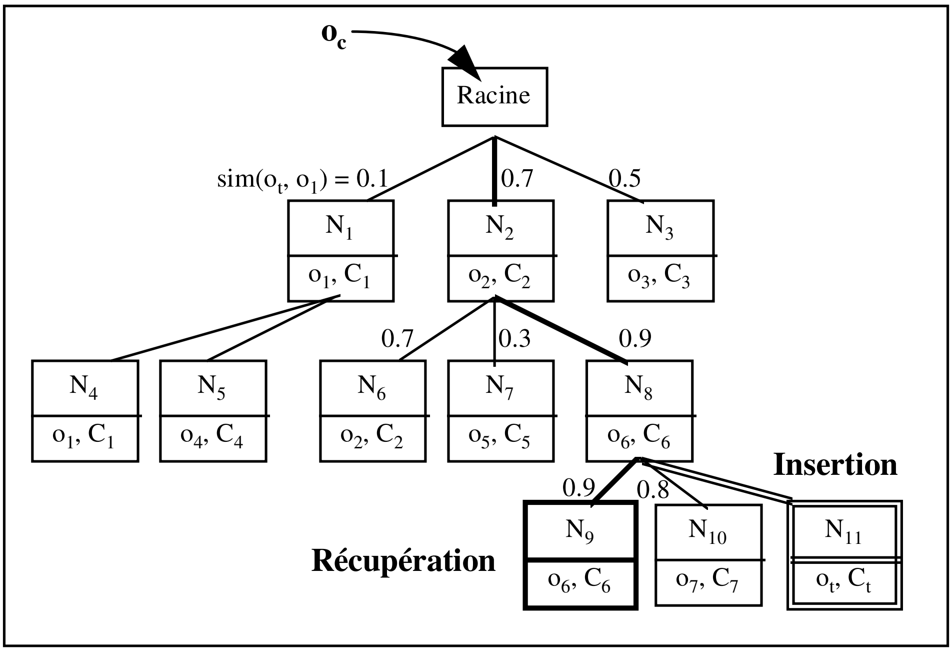


Figure 17. Exemple de mémorisation dans un arbre de prototypes avec *max_Degree* = 3

En ce qui concerne l'espace de mémoire qu'occupe l'arbre, les arcs peuvent être inclus dans la représentation des noeuds, puisqu'aucun noeud n'a plus que *max_Degree* arcs menant vers ses fils. Le nombre de noeuds dans un arbre est borné par une fonction linéaire du nombre de feuilles de l'arbre. Pour un arbre ayant *max_Degree* = 2, le nombre maximal de noeuds est de $2m$-1, m le nombre de feuilles. Dans un arbre de prototypes, il y a autant de feuilles que d'objets stockés, $m = n$. La complexité de l'espace mémoire occupé par l'arbre est donc $O(n)$, elle aussi. Ceci distingue de manière positive les arbres de prototypes d'autres formes d'organisation de la mémoire du type plus proches voisins, et en particulier de la méthode *Aesa* [Vidal 1986]. Celle-ci, outre qu'elle nécessite une mesure de similarité qui est une métrique, s'avère d'une complexité $O(n^2)$ pour ce qui est de l'espace mémoire.

La complexité de l'opération de récupération dans un arbre de prototypes dépend de la forme de l'arbre. En effet, un arbre dégénéré aura la forme d'un peigne, composé d'une seule longue branche dont chaque noeud ne contient qu'un seul fils feuille. Dans ce cas, la complexité “au pire” de la récupération est la même que dans une liste séquentielle, c'est à dire $O(n)$, n le nombre d'objets stockés. A l'opposé, la complexité minimale est atteinte lorsque l'arbre est équilibré. Elle est en conséquence de $O(log_{max\_Degree}\ n)$. En pratique, aucun des deux extrêmes ne sera atteint, et des accélérations considérables de la récupération, comparée la récupération dans une liste séquentielle d'objets, ont été présentées [Brown 1995]. La mémorisation étant une extension mineure à la récupération, il n'est pas nécessaire de la considérer explicitement.

On pourrait imaginer d'équilibrer un arbre de prototypes ayant une forme dégénérée. Toutefois, l'équilibrage d'arbres est une opération complexe, et le gain d'efficacité qu'engendre une opération d'équilibrage n'est pas très important pour des arbres qui ont une répartition aléatoire des noeuds [Wirth 1975]. En conséquence, nous n'envisageons pas l'équilibrage d'un arbre de prototype dégénéré, et nous ne connaissons pas de travaux menés en ce sens.

e) La précision de la récupération

Il convient de noter que l'utilisation d'un arbre de prototypes, tel qu'introduit ci-dessus, ne garantit pas la récupération de l'objet qui est le plus proche de l'objet cible. Il n'est même pas établi qu'un objet d'apprentissage, bien classé lors de la mémorisation et donc non ajouté à l'arbre, soit classifié ultérieurement de manière correcte.

Ce phénomène est typique des méthodes de condensation. Il est possible d'y remédier à la fin de la phase d'apprentissage, c'est à dire une fois l'ensemble des objets présentés pour mémorisation, en présentant une deuxième fois les objets qui étaient bien classifiés auparavant. En effet, l'arbre reconnaît l'ensemble des objets d'apprentissage si on les lui re-présente suffisamment de fois [Hart 1968]. Cependant, il en résulte que la mémorisation n'est plus incrémentale, puisque l'ensemble des objets d'apprentissage doivent être retenus pendant toute la durée de la phase de construction de l'arbre de prototypes.

En ce qui concerne la classification d'objets qui ne se trouvent pas dans l'arbre de prototypes, la récupération n'assure pas l'identification de l'objet le plus proche. En effet, il est possible que lors de la récupération l'algorithme sélectionne un “mauvais” fils successeur, et en conséquence manque le sous-arbre qui contient l'objet le plus proche de la cible. Pour illustrer notre propos, considérons l'exemple ci-dessus (figure 17) et supposons que l'objet le plus similaire à o_c soit o_4. Au cours de la récupération, o_c n'est jamais comparé directement à o_4, mais seulement au père de celui-ci, o_1, auquel o_c ne ressemble que peu. En raison de ce phénomène, il est possible que l'objet cible soit mal classifié. Ceci se produit, dans l'exemple, au cas où C_4 est différent de C_6.

Un troisième problème que rencontrent les arbres de prototypes concerne la catégorisation d'après le principe des k plus proches voisins, avec $k > 1$. Ce problème est lui aussi un “héritage” de la technique de condensation utilisée lors de la mémorisation [Hart 1968]. Il est, en effet, difficile de concevoir une technique de condensation pour classifier des objets par la règle des k plus proches voisins [Gates 1972].

4.2.2. Les arbres de prototypes dynamiques

Les arbres de prototypes, tels que présentés ci-dessus, s'appliquent cependant à la catégorisation et pas au raisonnement à partir de cas. Le lien entre ces deux techniques a fait l'objet de multiples travaux. Ils concernent par exemple l'utilisation de techniques de catégorisation comme composant d'un système de raisonnement à partir de cas [Aha & al. 1991], où l'application du raisonnement à partir de cas à la classification (chapitre 2, cf. p. 65). La principale différence entre la catégorisation et le raisonnement à partir de cas tient à ce que les cas n'appartiennent en général pas à des classes. A l'exception de la catégorisation à partir de cas, les systèmes de raisonnement à partir de cas ne s'intéressent guère à la classe auquel un cas pourrait appartenir, mais plutôt au cas lui-même, puisque c'est lui qui constitue l'élément central à partir duquel le processus d'adaptation construit la solution au problème cible.

Quelques adaptations s'imposent néanmoins pour rendre les arbres de prototypes plus efficaces dans le contexte du raisonnement à partir de cas (entre autres celle qui consiste à écarter les ensembles de classes C_i comme n'ayant plus de raison d'être). Ceci nous conduit à introduire les *arbres de prototypes dynamiques* qui nous permettent de poursuivre deux buts. D'une part, il est possible d'accélérer encore les opérations de récupération et de mémorisation. D'autre part, on peut mieux contrôler la forme que l'arbre prend afin d'éviter une dégénérescence en peigne. Notons que ces extensions des arbres de prototypes diffèrent de celles qu'introduit Brown (1995) pour améliorer la précision de la catégorisation.

a) L'accélération de la vitesse d'interaction

Lors de la catégorisation, il est important récupérer l'objet *le plus similaire* de l'objet cible afin de pouvoir défendre au mieux les classes qui sont attribuées à celui-ci. Il ne doit pas être en ainsi en raisonnement à partir de cas. En effet, dès lors qu'un cas récupéré permet à l'adaptation de construire une solution au problème cible, la récupération elle-même peut être considérée comme un succès. Il suffit donc que le cas récupéré soit *assez similaire* pour satisfaire à l'objectif de la récupération. Il est, en conséquence, possible de réduire la précision de la récupération, sous réserve du respect de cette condition du “assez similaire”, si en échange on obtient une amélioration de la vitesse d'interaction avec l'arbre de prototypes. Pour ce faire, nous introduisons ci-dessous trois extensions aux arbres de prototypes dynamiques.

La mémorisation nécessite tout d'abord une présentation multiple des objets d'apprentissage afin de pouvoir garantir la récupération correcte de ceux-ci. Elle n'est dès lors plus incrémentale. Cependant, le gain en précision que l'on obtient en procédant ainsi n'est que minime [Brown 1995]. Nous abandonnerons donc, pour ces deux raisons, la présentation multiple des objets dans les arbres de prototypes dynamiques.

Ensuite, une technique d'accélération simple de la récupération consiste à arrêter celle-ci avant qu'elle n'atteigne une feuille, mais dès qu'un objet assez similaire à la cible est identifié. Dans cette perspective, nous introduisons le *seuil de récupération RT*. La récupération pour l'objet cible o_c est terminée dès qu'un objet o_i est trouvé dont la similarité avec o_c est plus grande que RT, `sim`(o_c, o_i) $\geq RT$. L'objet o_i est alors récupéré. La fonction de récupération incluant ce paramètre est présentée au tableau 20.

```
fonction Récupère(cible): l'objet le plus proche
actuel := noeud Racine
tant que (actuel n'est pas une feuille) fait
    pour chaque (fils de actuel)
        calcule sim(cible, objet de fils)
        si ($sim \geq RT$) alors renvoi objet de fils
        retient fils avec la similarité maximale
    fin pour chaque
    actuel := le meilleur noeud fils
fin tant que
renvoi objet de actuel
```

Tableau 20. La récupération de cas dans une mémoire d'arbres de prototypes dynamiques

Suivant un approche comparable, la mémorisation peut être accélérée. Nous définissons IT de manière à ce que la mémorisation d'un objet o_c soit interrompue dès qu'elle rencontre un objet o_i dont la similarité avec o_c est plus grande que IT, `sim`(o_t, o_i) $\geq IT$. Ainsi, ne sont acquis que des objets qui sont suffisamment différents, et donc originaux, par rapport à ceux se trouvant déjà dans l'arbre [Smyth & Keane 1995b].

b) L'amélioration de la robustesse

Nous avons constaté que l'efficacité de la récupération varie en fonction de la forme que prend un arbre de prototypes, la différence entre un arbre dégénéré et un arbre équilibré étant considérable. La forme de l'arbre, quant à elle, dépend de l'ordre dans lequel les objets sont mémorisés. En effet, l'arbre est construit à partir de la racine, les objets arrivant les premiers occupent les niveaux les plus proches de celle-ci.

Cependant, si ces objets sont très similaires les uns aux autres, ils n'arriveront pas à bien discriminer les objets qui arrivent ultérieurement. Ces derniers peuvent alors être dirigés vers des sous-graphes auxquels ils ne devraient pas appartenir. Il nous parait donc judicieux de vérifier que les objets fils d'un même père soient assez différents les uns des autres. Pour ce faire, nous définissons un *seuil de divergence des noeuds NDT* de sorte que, lors de la mémorisation, un objet o_C est inséré comme fils d'un noeud N seulement si aucun autre fils de N n'est plus similaire de o_C que NDT. En notant $F(N)$ les fils de N, la condition suivante doit être vérifiée afin que o_C puisse être inséré comme nouveau fils de N :

$$\forall N' \in F(N),\ \mathtt{sim}(o_C, \mathtt{objet}(N')) \leq NDT \qquad (30)$$

```
fonction Insère(object)
si (arbre = vide)
    alors insère object comme fils de racine et renvoi
actuel := noeud Racine
tant que (actuel n'est pas une feuille)
    pour chaque (fils de actuel)
        calcule sim(cible, objet de fils)
        si ($sim \geq IT$) alors renvoi
        retient fils avec la similarité maximale
    fin pour chaque
    si ($sim$ maximale $\leq NDT$) alors break
    actuel := le meilleur noeud fils
fin tant que
NL := actuel
F := père de NL
si (degré de F < max_Degree) alors
    insère object comme fils de F et renvoi
sinon
    copie le contenu de NL vers un nouveau fils
insère object comme fils de NL et renvoi
```

Tableau 21. La mémorisation de cas dans une mémoire d'arbres de prototypes dynamiques

Le tableau 21 présente la nouvelle fonction de mémorisation des cas. Notons que l'introduction de ce seuil peut entraîner la récupération vers un comportement qui est tout à fait indésirable. En effet, il se peut qu'une bonne partie d'un sous-arbre devienne inaccessible après l'ajout d'un nouvel objet à un niveau supérieur.

Considérons l'exemple d'un espace d'objets à deux dimensions. L'arbre présenté dans la figure 18.a) correspond à la représentation graphique de l'espace des objets de la figure 18.b) où les objets de l'arbre sont représentés par un losange, et le seuil *NDT* par un cercle. L'objet o_3, se trouvant très légèrement à l'intérieur du seuil de l'objet o_1, est un fils de ce dernier. Considérons maintenant l'ajout de l'objet o_c. Très proche de o_3, o_c est néanmoins au dessus du seuil *NDT* de o_1 et satisfait donc à (30). En conséquence, il est ajouté comme fils de la racine. Lors des récupérations ultérieures, l'objet o_3, et s'il y a lieu tout le sous-arbre de racine o_3, a de forte chance d'être caché par o_c, puisque ce dernier se trouve à un niveau supérieur à o_3. Nous serions cependant en mesure de constater, lors de l'évaluation expérimentale des cette mémoire de cas que ce problème de l'objet écran, si même il se manifeste, ne diminue pas la précision de la récupération de cas (chapitre 6, cf. p. 178).

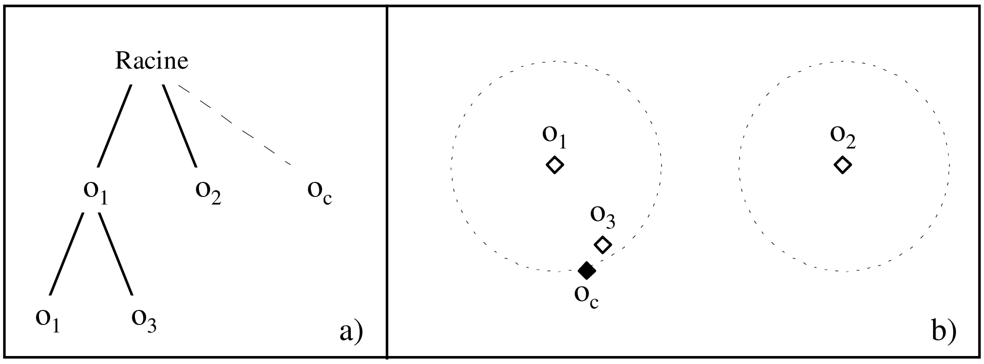


Figure 18. Le problème de l'objet écran dû au seuil de divergence des noeuds

Lors de l'ensemble des expérimentation présentées au chapitre 6, nous avons employé un arbre de prototypes dynamique qui configurée avec un nombre maximal de fils *max_Degree* = 20, un seuil d'insertion *IT* = 0.9, un seuil de récupération *RT* = 0.9 et un seuil de divergence des noeuds *NDT* = 0.2. Quelques expériences préliminaires, que nous ne retranscrivons pas ici, nous ont conduit à retenir ces valeurs pour les paramètres. Brièvement, nous avons évalué *IT* et *RT* pour les valeurs 0.8 et 0.9, et *NDT* pour 0.1 et 0.2. Les performances de la mémoire ne variaient pas sensiblement en fonction des diverses combinaisons de ces paramètres, quand bien même les paramètres retenus s'avéraient être de peu les meilleurs.

Une mémoire de cas qui est basée sur un arbre des prototypes dynamique réunit les avantages des deux organisations principales de mémoires de cas. Elle permet donc la prise en compte d'une mesure de similarité explicite sur des bases de cas larges. C'est précisément de cette mesure de similarité dont on va traiter le prochaine chapitre.

Chapitre 5.

Similarité et Appariement

Le concept de similarité est l'un des éléments clés de tout système de raisonnement à partir de cas. Etre capable de reconnaître un cas comme proche, ou *similaire*, au problème cible est un premier pas vers sa résolution. On fait appel à l'évaluation de la similarité dans deux situations précises : premièrement pendant la récupération, lorsque l'on recherche dans la mémoire de cas, le (ou les) cas le plus proche du problème cible, et deuxièmement pendant la mémorisation, lorsque l'on acquiert un nouveau cas et que l'on tente de le placer dans la région de la mémoire qui contient les cas qui lui ressemblent le plus.

Dans le chapitre précédent, nous avons traité du lien qu'il y a entre la mesure de similarité et l'organisation de la mémoire de cas. Nous avons, à cette occasion, expliqué les avantages d'un recours, dans notre système, à une mesure de similarité explicite. Nous présenterons maintenant la mesure de similarité elle-même. Le langage de représentation des problèmes adopté dans *Alois*, basé sur la logique des prédicats, rend l'évaluation de la similarité difficile. En effet, la comparaison de deux descriptions, représentées dans ce langage, requiert un appariement préalable des éléments les composant. En effet, la ressemblance varie en fonction de l'appariement retenu [Ganascia 1987]. Nous verrons que la construction des appariements peut être assimilée au problème d'isomorphisme de sous-graphes partiels maximaux qui est un problème d'optimisation NP-difficile.

Le présent chapitre est construit de la manière suivante : dans une première section, nous expliquerons le lien fondamental entre la mesure de similarité et l'appariement. Après quoi, dans la seconde section nous modéliserons l'évaluation de la similarité comme un problème d'optimisation, avant de présenter, dans les sections trois et quatre des techniques utiles à la résolution de celui-ci.

5.1. La notion de la similarité

Lors de la récupération de cas, ainsi que lors de leur mémorisation, il faut établir une ressemblance entre un cas et un opérateur applicable (récupération), ou entre deux cas (mémorisation). Cette ressemblance est fonction, d'une part de la *situation* dans laquelle se trouve l'opérateur applicable, et d'autre part de la *situation* dans laquelle le cas a été engendré.

La *mesure de similarité* compare ces deux situations. Ces situations sont représentées dans un langage formel basé sur la logique des prédicats. La définition précise de ces situations, que nous appelons *contextes d'instanciation*, a été donnée au chapitre 3 (cf. p. 95). Selon cette définition, une situation est décrite par un ensemble de littéraux. Evaluer la ressemblance entre deux situations revient donc à comparer ces deux ensembles de littéraux. Leur représentation en logique des prédicats rend cette tâche non-triviale, puisqu'elle requiert un appariement entre les éléments des deux descriptions [Kodratoff & Ganascia 1986, Ganascia 1987]. L'explication des problèmes qui surgissent constitue l'objet de cette première section.

5.1.1. Le lien entre similarité et appariement

Etablir la ressemblance entre deux descriptions en logique de prédicats est un problème complexe et dichotomique. Il s'agit d'abord de *mettre en correspondance*, ou d'*apparier*, les éléments contenus dans ces deux descriptions. Compte tenu du langage de représentation, cette mise en correspondance n'est, en général, pas unique ; il s'agit plutôt d'un espace de mises en correspondance [Ganascia 1987]. La ressemblance entre deux descriptions est fonction de chaque mise en correspondance. Parmi toutes les mises en correspondance possibles, il parait néanmoins judicieux de retenir celle pour laquelle les deux descriptions se ressemblent le plus. C'est, en effet, cette mise en correspondance qui est la plus révélatrice de la similarité entre les deux descriptions. En quelque sorte, cela revient à maximiser, sur l'espace des mises en correspondance possibles, la ressemblance entre les descriptions[1].

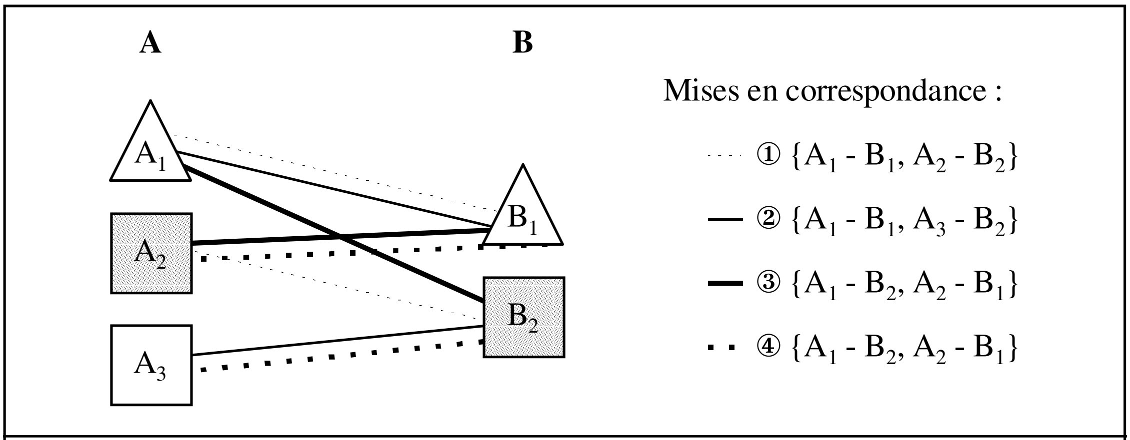
Avant d'être plus formel, considérons l'exemple d'un monde hypothétique de polygones, présenté dans la figure 19. La scène A (à gauche) contient un triangle blanc au dessus d'un carré noir, qui lui se trouve au dessus d'un carré blanc, tandis que la scène B contient un triangle au dessus d'un carré, tous deux blancs. Au dessous de chacune des deux scènes figurent des descriptions qu'il est possible d'en faire en logique des prédicats. Ainsi le littéral `(forme` A_1 `blanc 3)` indique que le polygone A_1 est de couleur `blanc` et possède `3` cotés.

Les mises en correspondance concernent les entités des deux scènes, ici les polygones. On distingue deux types de mises en correspondance, celles qui mettent en relation deux entités une à une, et celles qui mettent en relation plusieurs entités avec une seule. Nous allons, dans ce qui suit, nous restreindre aux mises en correspondance 1-à-1. Différentes mises en correspondance 1-à-1 sont possibles entre les entités des scènes A et B. La figure 19 en indique quatre. La première ①, en pointillé, associe A_1 à B_1 et A_2 à B_2. La deuxième ②, en ligne

---

[1] Ce qui est distinct du but de la récupération de cas, à savoir l'identification du cas *le plus proche* se trouvant dans la mémoire de cas.

continue, associe A_1 à B_1 et A_3 à B_2, la troisième ③, en gras, A_2 à B_1 et A_1 à B_2, et la quatrième ④, en pointillé gras, A_2 à B_1 et A_3 à B_2.



(forme A_1 blanc 3)
(forme A_2 noir 4)
(forme A_3 blanc 4)
(sur A_1 A_2)
(sur A_2 A_3)

(forme B_1 blanc 3)
(forme B_2 noir 4)

(sur B_1 B_2)

Figure 19. Des mises en correspondance 1-à-1 dans le monde des polygones

Etudions maintenant la similarité entre deux scènes. Deux scènes sont similaires par référence aux objets qu'elles contiennent et qui sont mis en relation. Cette similarité est perçue de manière différente pour chacune des quatre mises en correspondance. On peut par exemple constater que, sous ③, les deux scènes sont très dissimilaires, puisque des triangles sont associés à des carrés et les entités ne sont pas toujours de la même couleur. La ressemblance des scènes pour les trois autres mises en correspondance est plus grande. Ainsi ① apparie des objets de forme et de disposition identiques, et ② apparie des objets identiques, mais disposés différemment puisque, dans la scène A, les objets appariés ne sont pas voisins. Ainsi, et en ne s'appuyant que sur une définition intuitive de la notion de similarité, il est possible d'attribuer différentes ressemblances à ces deux situations, selon la mise en correspondance que l'on considère. Cependant, lorsque l'on détermine cette similarité, on recherche spontanément la mise en correspondance qui la maximise. Dans l'exemple, ③ est moins approprié, pour évaluer la similarité entre A et B, que ne le sont ①, ② ou ④.

Considérée de cette manière, l'évaluation de la similarité devient un *problème d'optimisation*. Nous avons vu, au premier chapitre, qu'un problème d'optimisation est constitué d'une région de faisabilité et d'une fonction objectif (cf. p. 18). Ici, la région de faisabilité est définie par l'espace des mises en correspondance, et la fonction objectif, que l'on tente de maximiser, correspond à la fonction de similarité. Ce problème d'optimisation est donc étroitement lié au problème général de l'appariement. Nous considérerons ci-dessous d'abord ce dernier, avant de revenir sur l'évaluation de la similarité.

Avant de procéder, on peut constater que la situation se présente ici de manière similaire à ce que l'on rencontre en apprentissage symbolique inductif. Le processus d'apprentissage peut, en effet, être séparé en deux phases. La première est relative à la construction d'appariements entre les exemples d'apprentissage, et la deuxième procède à la généralisation des exemples appariés [Ganascia 1987, Zucker 1996].

5.1.2. Le problème d'appariement

Le problème d'appariement se rencontre dans des domaines qui contiennent des objets complexes et qui disposent d'une structure. Ces *objets structurés*, dont les deux scènes de la figure 19 sont des exemples, se caractérisent d'une part par le fait que leur structure varie d'un objet à l'autre. D'autre part, par des composants en nombre variable mais qui, pour la tâche que l'on se fixe, se ressemblent et, en conséquence, ne peuvent pas être distingués avec la précision requise. Afin de décrire ces deux particularités, les objets structurés sont représentés dans des langages de la logique de premier ordre, tel que APC [Michalski 1983]. Ce sont des langages qui sont plus expressifs que les langages basés sur la logique propositionnelle. Au chapitre 3, nous avons exposé les raisons nous ayant amené à utiliser un langage d'ordre 1 pour la représentation des problèmes dans *Alois* (cf. p. 88).

a) Une représentation graphique

Il est assez commun de recourir, en raison de son expressivité visuelle, à une représentation graphique des objets structurés, ou des cas. Il en va ainsi dans des systèmes comme *Protos* [Porter & al. 1990], dans les systèmes utilisant la mémoire dynamique [Schank 1982] ou encore dans *SME* [Falkenhainer et al. 1989]. La plus naturelle des représentations graphiques est peut-être celle des *graphes d'instances* [Haussler 1989]. Un graphe d'instance est un graphe $G = (N, A)$, avec N un ensemble de noeuds et A un ensemble d'arcs orientés, $a_{ij} = (N_i, N_j)$. Les noeuds correspondent aux entités du domaine. Ils sont étiquetés par les prédicats unaires. Les prédicats binaires constituent les arcs, et leurs noms forment les étiquettes des arcs. Les prédicats n-aires, $n > 2$, formeraient alors des hyperarcs. Ici, l'arité d'un prédicat est le nombre d'entités qu'il contient. Ainsi, le prédicat `forme`, de l'exemple précédant, est d'arité 1, bien qu'il contienne 3 attributs. Au chapitre 3, nous avons défini ce que nous entendons par entité du domaine (cf. p. 91).

Les graphes d'instances ont la particularité d'être équivalents à une représentation en langage de premier ordre. On remarque que lorsqu'il y a absence de prédicats unaires, et qu'il n'y a qu'un seul prédicat binaire, qui de plus est booléen, les graphes d'instances sont équivalents aux graphes orientés non-étiquetés classiques [Haussler 1989].

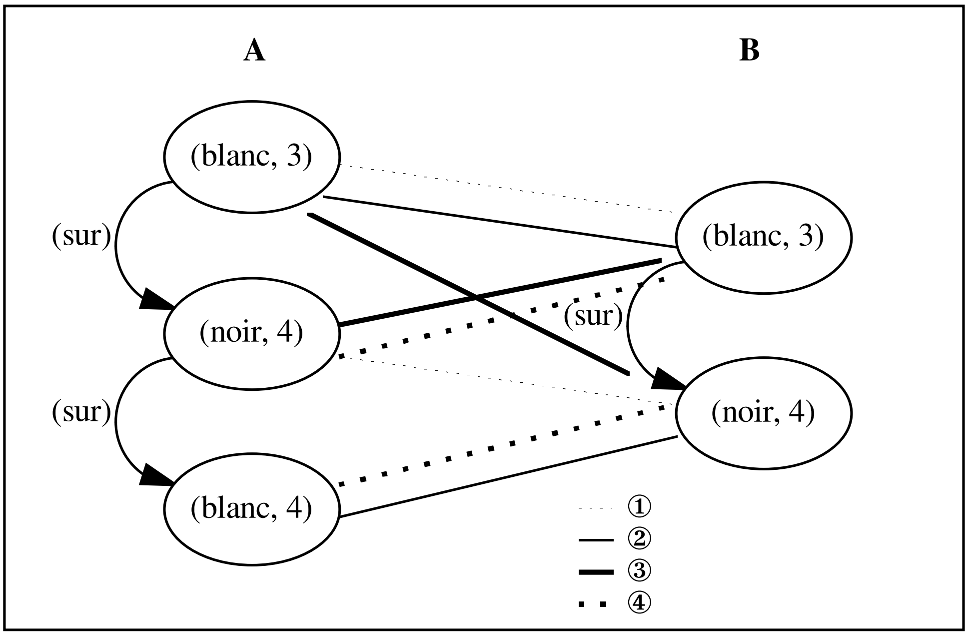


Figure 20. Des graphes d'instances du monde des polygones

La figure 20 montre les deux situations de la figure 19, représentées sous forme de graphes d'instances. Ici, les entités du domaine sont les polygones. En conséquence, chaque polygone correspond à un noeud. Les prédicats unaires `forme` composent les étiquettes de ces noeuds. Les prédicats binaires `sur` décrivent les relations entre ces noeuds et sont représentés par des arcs orientés. Les mises en correspondance entre graphes d'instances, mettent en relation des noeuds entre eux. Ainsi, la figure 20 montre les mises en correspondance ①, ②, ③ et ④.

b) La définition de l'appariement entre graphes

Le recours à une représentation graphique nous permet d'utiliser des éléments de la théorie des graphes pour décrire le problème d'appariement dans un cadre formel[2]. A titre préliminaire, il convient de rappeler brièvement quelques notions basiques de la théorie de graphes [Berge 1970]. Etant donné le graphe $G = (N, A)$, avec N un ensemble de noeuds et A un ensemble d'arcs. G est un graphe simple, c'est à dire non-orienté et non-étiqueté (figure 21.a). Un *graphe partiel* $G'_1 = (N', A')$ de G contient les mêmes noeuds que G, $N' = N$, mais seulement un sous-ensemble des arcs, $A' \subseteq A$ (figure 21.b). Un *sous-graphe* $G'_2 = (N', A')$ de G contient un sous-ensemble des noeuds de G, $N' \subseteq N$, ainsi que l'ensemble des arcs entre les noeuds de N' (figure 21.c). Un *sous-graphe partiel* $G'_3 = (N', A')$ de G est la combinaison d'un graphe par-

---

[2] Il est, en effet, plus usité de définir l'appariement entre objets structurés sans le passage par la théorie des graphes, mais directement dans un formalisme basé sur la logique [Kodratoff & Ganascia 1986, Ganascia 1987].

tiel et d'un sous-graphe. *G'₃* contient alors un sous-ensemble des noeuds de *G*, $N' \subseteq N$, ainsi qu'un sous-ensemble des arcs entre les noeuds de *N'* (figure 21.d).

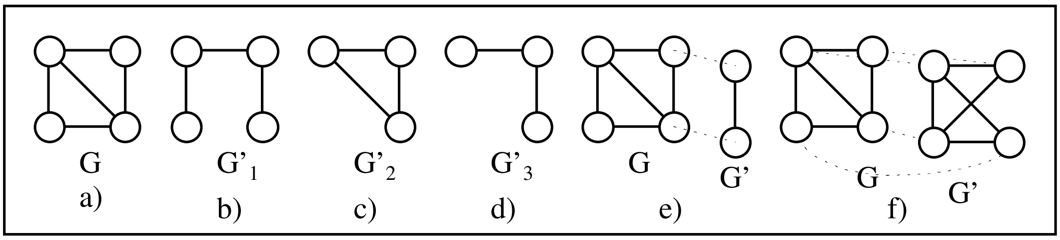


Figure 21. Des propriétés élémentaires entre graphes simples

Considérons maintenant les relations entre deux graphes. On appelle *mise en correspondance* (“*mapping*”) entre deux graphes *G'* et *G* une fonction *m* qui donne des noeuds de *G'*, c'est à dire *N'*, vers les noeuds de *G*, *N*. Un *homomorphisme* est une mise en correspondance *m* qui associe à tout noeud de *G'* un noeud de *G*, de sorte qu'à tout arc de *G'* soit associé un arc de *G*. Cette relation s'appelle aussi *appariement* (“*matching*”), ou encore *appariement exact* [Vosselmann 1992] (figure 21.e). Un *isomorphisme* entre *G* et *G'* est une mise en correspondance qui implique que pour toute bijection entre *N'* et *N*, il y a une bijection entre *A'* et *A* [Köbler & al. 1993] (figure 21.f).

En ce qui concerne les graphes orientés et étiquetés, un homomorphisme est une mise en correspondance qui respecte les orientations des arcs ainsi que les étiquettes des noeuds et des arcs. Un isomorphisme entre graphes étiquetés est un *appariement structurel* [Ganascia 1987]. Si l'on considère les mises en correspondance de la figure 20, on se rend compte qu'il n'y a que ① qui soit un homomorphisme de B vers A et donc un appariement exact. En revanche, ④ serait un appariement exact de B vers A si l'on ne tenait pas compte des étiquettes des noeuds.

Ceci nous conduit à la définition suivante. Nous appelons *appariement inexact* toute mise en correspondance qui est un appariement exact si ce n'est pour les étiquettes des noeuds. Selon ce critère, ④ est un appariement inexact[3]. Intuitivement, on peut constater qu'il y a beaucoup plus d'appariements inexacts que d'appariements exacts.

L'ultime mise en correspondance que nous considérons ici est *l'isomorphisme entre sous-graphes partiels*. Celui-ci met en évidence un sous-graphe partiel qui est commun aux deux graphes. Comme pour l'appariement inexact, il est possible de définir l'*isomorphisme inexact entre sous-graphes partiels* comme étant un isomorphisme entre sous-graphes partiels à l'étiquetage des noeuds près. L'appariement ① de la figure 20 est un isomorphisme entre sous-

---

[3] Il n'y a pas unanimité quant à la définition de l'appariement inexact. Nous adoptons ici la même définition de Bunke & Messmer (1993), qui diffère de celle employée en reconnaissance de formes [Shapiro & Haralick 1990, Vosselmann 1992].

graphes partiels, tandis que les mises en correspondance ②, ③ et ④ sont des isomorphismes inexacts.

c) La complexité de l'appariement entre graphes

Lorsque l'on projette de construire des appariements entre graphes, se pose automatiquement la question de la complexité de ces problèmes. Nous avons rappelé quelques notions de base de la théorie de la complexité au premier chapitre (cf. p. 19). En ce qui concerne la complexité de la construction de ces différentes mises en correspondance entre graphes, le problème de décision concernant l'existence d'un isomorphisme entre deux graphes fait partie de NP, bien que la question reste encore ouverte quant à savoir s'il est aussi dans NP-complet [Köbler & al. 1993]. Le problème de décision qui consiste à trouver un sous-graphe partiel composé d'au moins k éléments est, quant à lui, un problème NP-complet [Garey & Johnson 1979]. Le problème d'optimisation associé à ce problème de décision, visant à trouver le sous-graphe partiel *maximal*, est en conséquence *NP-difficile*. L'existence d'un algorithme efficace pour sa résolution exacte est alors peu probable (seulement si P = NP).

Considérons maintenant les mises en correspondance entre graphes particuliers. Dans un premier temps, examinons les graphes *racinés*. Le fait qu'un graphe soit raciné implique que les racines soient nécessairement mises en correspondance entre elles. On connaît donc deux noeuds qui a priori doivent être communs à tous les sous-graphes partiels. Intuitivement, cela devrait conduire à une réduction de la complexité du problème de l'identification du sous-graphe maximal. Néanmoins, il n'en va pas ainsi [Fu 1995]. Le problème du sous-graphe partiel maximal entre deux graphes orientés, étiquetés et racinés, reste NP-difficile. En ce qui concerne les *arbres* qui sont aussi des graphes racinés où tout sous-graphe commun doit obligatoirement contenir la racine. En outre, la structure des arbres est plus simple que celle des graphes racinés généraux. Mais là encore, cela ne conduit pas à une réduction de la complexité du problème de l'identification du sous-graphe partiel maximal commun ; ce problème étant NP-difficile [Shasha & al. 1994]. Néanmoins, il n'en va pas de même avec les *graphes ordonnés*. Un graphe est ordonné s'il existe un ordre entre les fils de chaque noeuds. Cet ordre doit alors être respecté par les mises en correspondance. En conséquence, le problème de l'identification du sous-graphe partiel maximal entre deux graphes ordonnés, racinés (ainsi qu'entre deux arbres) est polynomial [Shasha & al. 1994, Fu 1995].

d) L'appariement en raisonnement à partir de cas, en analogie et en apprentissage symbolique

Le *problème d'appariement*, tel qu'esquissé, est un problème tout à fait général. Il apparaît dès que l'on tente d'établir des liens entre éléments appartenant à deux ensembles, et que ces liens ne sont pas nécessairement univoques. Nous considérons maintenant l'un après l'autre les domaines du raisonnement à partir de cas, de l'analogie et de l'apprentissage symbolique dans leurs rapports avec les problèmes d'appariement. Toutefois, le problème d'appariement en tant que tel est bien plus vaste, et dépasse ces trois domaines. En le rencontre aussi par exemple en matière de reconnaissance de formes [Shapiro & Haralick 1990, Vosselmann 1992], ou encore en construction de compilateurs [Fu 1995].

Les systèmes de raisonnement à partir de cas ont recours à différents langages de représentation des cas. Un bon nombre de ces systèmes utilisent une représentation des cas en logique de proposition. A titre d'exemples, il en va ainsi dans *Battle Planner* [Goodmann 1989], *Protos* [Porter & al. 1990], *Casey* [Koton 1988], et *Clavier* [Barletta & Hennessy 1989]. Dans ces représentations, les descriptions sont des couples `(attribut valeur)`, et les attributs de même nom sont appariés d'office. Le problème d'appariement ne se pose donc pas. Cependant, il est possible d'introduire des connaissances qui permettent la mise en correspondance entre attributs ayant des noms différents. C'est par exemple le cas dans *Protos* ou *Casey*.

D'autres systèmes font usage des représentations qui s'approchent de la logique des prédicats. C'est le cas des systèmes basés sur les *frames* et la *mémoire dynamique* [Schank 1982], tels que *Cyrus* [Kolodner 1993], *Chef* [Hammond 1986], *Julia* [Hindrichs 1988] et *Mediator* [Kolodner & Simpson 1989]. Dans ces systèmes, il peut exister un problème d'appariement. Cependant, ces systèmes évitent le problème car des connaissances du domaine, incorporées lors de la conception du système, le désamorcent. Une vraie représentation en logique de premier ordre est utilisée dans *Prodigy/Analogy* [Veloso 1994] et dans *Caplan/Cbc* [Muñoz-Avila & Huellen 1995]. L'appariement doit alors être explicitement pris en compte lors de l'évaluation de la ressemblance entre cas. En outre, Plaza (1995) propose l'adoption d'une représentation des cas en un langage d'ordre 1.

L'appariement est un élément essentiel de tout système de *raisonnement par analogie* [Gentner 1983, Holyoak & Thagard 1989, Py 1994]. C'est, en effet, la mise en correspondance de relations émanant de différents domaines qui constitue la base de l'analogie telle qu'elle est par exemple mise en oeuvre dans la *Structure Mapping Engine* [Falkenhainer et al. 1989].

Le problème d'appariement est bien connu en *apprentissage symbolique automatique*. Il se manifeste en apprentissage inductif, dès que les exemples sont représentés en logique de premier ordre [Winston 1975, Michalski 1983, Ganascia 1987, Quinlan 1990, Zucker 1996], en programmation logique inductive [Bergadano & Gunetti 1996] ou en regroupement conceptuel [Stepp & Michalski 1986]. Dans ces domaines, l'appariement est un passage obligé pour la construction ultérieure de généralisations. Cependant, l'objectif n'est pas nécessairement de construire des appariements maximaux, mais plutôt des appariements qui donnent lieu à de bonnes généralisations par la suite [Ganascia 1987, Zucker & Ganascia 1996].

5.1.3. La similarité conçue comme problème d'optimisation

Grâce aux définitions des appariements entre graphes, présentées ci-dessus, nous sommes maintenant en mesure de clarifier la notion de similarité telle que nous entendons l'utiliser. En premier lieu, nous aborderons l'évaluation de la similarité en termes de résolution d'un problème d'optimisation. Se posera alors, dans un second temps, le problème du choix du type de mise en correspondance le plus adapté. Après avoir considéré ces deux points, il sera possible, dans la prochaine section, de formuler de manière exacte l'évaluation de la similarité comme étant un problème d'optimisation se rapprochant du problème d'isomorphisme inexact maximal entre sous-graphes partiels.

a) Maximisation de la partie commune à deux graphes

Différents *types de mises en correspondance* entre graphes, tels que l'appariement exact, l'appariement inexact, ou encore les différentes formes d'isomorphisme ont été décrits ci-dessus. Ces types de mises en correspondance laissent tous paraître, d'une manière ou d'une autre, des parties qui sont communes aux deux graphes qu'ils mettent en relation. Supposons l'existence d'une fonction $similarité(IC_1, IC_2)$ qui quantifie la taille de la partie commune maximale entre deux situations IC_1 et IC_2. Appelons M_T l'espace des mises en correspondance du type T. Ainsi par exemple, si T correspond à un appariement exact, $M_{AppExact}$ contiendrait l'ensemble des appariements exacts entre deux situations. Ensuite, appelons m_T une mise en correspondance du type T, $m_T \in M_T$, et $c(IC_1, IC_2, m_T)$ la partie commune entre IC_1 et IC_2 sous la mise en correspondance m_T.

Il est alors possible de formuler l'évaluation de la similarité comme un problème d'optimisation sur l'espace des mises en correspondance, avec :

(PSIM)
$$similarité(IC_1, IC_2) = \max_{\forall m_T \in M_T} \left[\left| c(IC_1, IC_2, m_T) \right| \right] \tag{31}$$

Ainsi, $|c(\cdot)|$ est la fonction objectif du problème d'optimisation, et M_T corresponds à sa région de faisabilité. Il reste néanmoins à déterminer le type de mise en correspondance T qui conviendrait le mieux.

Aborder de manière explicite l'évaluation de la similarité comme un problème d'optimisation est chose assez rare dans les systèmes de raisonnement à partir de cas. A notre connaissance, seuls Bunke & Messmer (1993) adoptent cette démarche, bien que leur approche ne semble jamais avoir été mise en oeuvre à l'intérieur d'un système opérationnel.

b) Le type de la mise en correspondance

Le choix du type de mise en correspondance dépend naturellement des caractéristiques auxquelles doit satisfaire la mesure de similarité. Dans le cadre de notre système, ces caractéristiques sont au nombre de deux :

1. Le principe même du raisonnement à partir de cas veut que d'anciennes solutions à des problèmes similaires soient réutilisées lors de la résolution du problème cible. Par contre, il est rare que la mémoire de cas contienne un problème identique au problème cible[4]. En réalité, deux problèmes sont similaires lorsqu'ils partagent des caractéristiques primordiales, même s'ils divergent sur des éléments secondaires. Cependant il est très peu fréquent qu'un problème soit inclut dans un autre.

   Ce sont des *parties* de cas qui sont responsables de la pertinence d'un cas. En termes de graphes, il s'agit de sous-graphes partiels. Il est donc possible d'exclure, dès à présent, les mises en correspondance entre graphes entiers, et de n'admettre que des mises en correspondance entre sous-graphes partiels. La même réflexion est menée dans la plupart des systèmes de raisonnement à partir de cas, peut-être de la manière la plus explicite par Bunke & Messmer (1993), et dans les systèmes de raisonnement par l'analogie [Falkenhainer et al. 1989].
2. Le domaine d'application du système à l'étude est l'optimisation combinatoire où les problèmes contiennent en général un grand nombre d'attributs numériques (cf. p. 166). Si l'on considère des mises en correspondance exactes, ceci impose que deux valeurs numériques soient identiques pour qu'il puisse y avoir mise en correspondance entre elles[5]. Ceci est une contrainte par trop restrictive. Elle peut être abandonnée en autorisant des mises en correspondance inexactes.

C'est ainsi que l'*isomorphisme inexact entre sous-graphes partiels* est le type de mise en correspondance que nous retenons dans le cadre de notre système *Alois*. Dès à présent, nous le désignons indifféremment comme “appariement” ou “mise en correspondance”.

5.2. Modélisation de la similarité

Nous avons identifié l'évaluation de la similarité entre deux situations comme étant un problème d'optimisation classique consistant à trouver le sous-graphe partiel maximal entre deux graphes. Néanmoins, nous n'avons pas encore précisé ce que nous entendions par *maximal* ; en d'autres termes, nous n'avons pas établi comment est définie la similarité entre deux situations sous un appariement donné. Cette similarité dépend de manière naturelle (de la taille) de la description qui leur est commune. Mais de plus, puisque l'appariement peut être inexact et peut donc apparier des éléments qui ne sont pas identiques, la similarité dépend aussi de la ressem-

---

[4] D'où le besoin d'une phase d'adaptation [Kolodner 1993].

[5] Rappelons que nous avons écarté la transformation des attributs numériques par des techniques de discrétisation ou de saturation (chapitre 4, cf. p. 100).

blance entre les éléments qui sont appariés entre eux. Préciser, ou modéliser, la notion de similarité en tant que problème d'optimisation sur l'espace des mises en correspondance est le but de cette section. C'est seulement dans les deux sections qui suivent que nous traiterons des techniques, a priori heuristiques, permettant de trouver des appariements qui débouchent sur une similarité de bonne qualité.

5.2.1. La similarité entre deux situations

La situation d'un cas ou d'un opérateur applicable est appelée *contexte d'instanciation*. Reprenons brièvement sa définition (chapitre 3, cf. p. 95). Un contexte d'instanciation CI est composé d'un ensemble de strates St_i, $CI = \{St_i\}$, $i = 1 \ldots n$. Une strate est composée d'un ensemble de littéraux L_{ij}, $St_i = \{L_{ij}\}$. Pour un CI, chaque littéral est unique. Les littéraux sont eux-mêmes composés d'un nom et d'un nombre variable d'attributs a. Nous appelons A l'ensemble des attributs a_k contenus dans les littéraux d'un CI, $A = \{a_k\}$. En outre, les attributs sont typés ; c'est ainsi que les attributs objet, de type contenu d'ensemble, désignent les entités du domaine, tandis que les attributs valeur, de type ordonné ou contenu d'ensemble, servent à les décrire.

a) Les éléments à apparier

Si l'on considère deux situations CI_1 et CI_2, le calcul de leur similarité nécessite une mise en correspondance des éléments qui les composent. En choisissant la représentation des situations sous forme de graphes d'instances, ce sont les attributs objet désignant les entités du domaine qui composent les noeuds de ces graphes. C'est donc pour ces attributs objet que, a priori, il s'agit d'établir une mise en correspondance. Nous appelons cette fonction de mises en correspondance σ_O. Sous σ_O les deux situations sont appariées. Ainsi, de σ_O découlent les liens entre les étiquettes des noeuds, qui sont ici les attributs valeur, et les liens entre les arcs, ici les littéraux positifs. En ce sens, le choix de σ_O est lourd de conséquences.

Rappelons que nous avons affaire à un problème NP-difficile, et que pour des raisons d'efficacité, il n'est guère possible d'envisager autre chose qu'une technique heuristique pour la construction de σ_O. En anticipant sur les développements de la prochaine section, nous adaptons la modélisation de la similarité aux impératifs de sa résolution[6]. En effet, ce sont des *algorithmes gloutons* qui devront être employés pour la construction des mises en correspondance. Un algorithme glouton parcourt l'espace de recherche en optant à chaque étape pour le chemin qui lui parait le plus perspicace. Ne mettant jamais en cause les choix antérieurs, il débouche en général sur un optimum local. Le recours à une telle technique n'est possible que

---

[6] Modéliser un problème d'optimisation selon les techniques de résolution disponibles est chose commune en optimisation [Williams 1993a].

lorsque l'on estime que les optimaux locaux ne sont pas trop éloignés de l'optimum global. Une heuristique simple visant à améliorer les algorithmes gloutons fait parcourir l'espace de recherche à petit pas. Ainsi, on espère que l'erreur faite à chaque étape sera de moins de porté que si l'on faisait des choix plus importants.

Toutefois, le choix des éléments de σ_O implique, outre les liens entre les attributs objet, des liens entre attributs valeur et aussi celles entre les littéraux. En conséquence, il serait peu judicieux d'utiliser un algorithme glouton qui construit directement σ_O. Mieux vaut prendre un algorithme qui “fait des pas plus petits”, et d'élargir son champs d'action des attributs objet aux attributs valeur et aux littéraux. Pour cette raison nous introduisons les deux mises en correspondance σ_L et σ_{A} ; σ_L est une relation entre littéraux, et σ_A est une relation entre attributs (objet et valeur). Dans la suite de cette section, nous traiterons de l'espace de définition de ces deux fonctions, mais non de la construction de σ_L et σ_A elles-mêmes.

b) La normalisation de la fonction de similarité

La similarité entre CI_1 et CI_2, $sim(IC_1, IC_2)$, est fonction du sous-graphe que CI_1 et CI_2 ont en commun. Nous appelons c_{12} ce sous-graphe. Supposons pour l'instant que c_{12} soit connu. Afin d'améliorer la comparaison de plusieurs situations, la fonction de similarité est, en général, normalisée dans l'intervalle [0…1]. Il devient ainsi possible de faire des affirmations du type

si $sim(\mathrm{A}, \mathrm{B}) \geq sim(\mathrm{C}, \mathrm{D})$, alors A est plus proche de B que C ne l'est de D

Différentes normalisations ont été proposées. L'une d'elles (32) est souvent utilisée tant en raisonnement à partir de cas qu'en apprentissage à partir d'instances. Une autre encore, la *mesure de Jaccard* (33), est fréquemment employée en analyse de données [Roux 1985]. Il existe par ailleurs d'autres normalisations, cf. [Tversky 1977] ou [Michalski & al. 1981].

$$sim(IC_1, IC_2) = \frac{|c_{12}|}{\max\{|IC_1|, |IC_2|\}} \tag{32}$$

$$sim(IC_1, IC_2) = \frac{|c_{12}|}{|IC_1| + |IC_2| - |c_{12}|} \tag{33}$$

Ici, $|IC|$ correspond à la taille pondérée du contexte d'instanciation telle que définie par (34). Notons que, a priori, les coefficients de pondération des littéraux α_L peuvent varier dans [0…1] en fonction du nom du littéral. Pour toutes les expérimentations rapportées dans cette thèse, nous avons, cependant, toujours choisi $\alpha_L = 1$.

$$|IC| = \sum_{L \in IC} \alpha_L \tag{34}$$

La figure 22 compare ces deux normalisations. Nous supposons que $|IC_1| = 10$ fixe. Pour chacune des deux normalisations, trois courbes, correspondant à $|IC_2| = 5$, $|IC_2| = 8$ et $|IC_2| = 10$, sont présentées. L'ordonné représente $|c_{12}|$ dans $[0\ldots|IC_1|]$, tandis que l'abscisse indique la valeur de *sim*. Les lignes en gras correspondent à la normalisation (32), et les trois courbes fines reproduisent la mesure de Jaccard (33). Chaque courbe s'arrête aux points $|IC_2| = |c_{12}|$. Les trois carrés noirs marquent les points d'arrêt pour chaque valeur de $|IC_2|$. On remarque que les trois courbes de (32) se superposent. Ceci s'explique du fait qu'elles sont indépendantes de $|IC_2|$. La mesure de Jaccard, cependant, différencie la similarité normalisée selon la valeur de $|IC_2|$, comme en atteste les trois courbes distinctes. Elle est donc plus discriminative. Pour cette raison, nous l'adoptons comme fonction de normalisation de la similarité. Un exemple de cette normalisation figure plus loin dans ce chapitre (cf. p. 152).

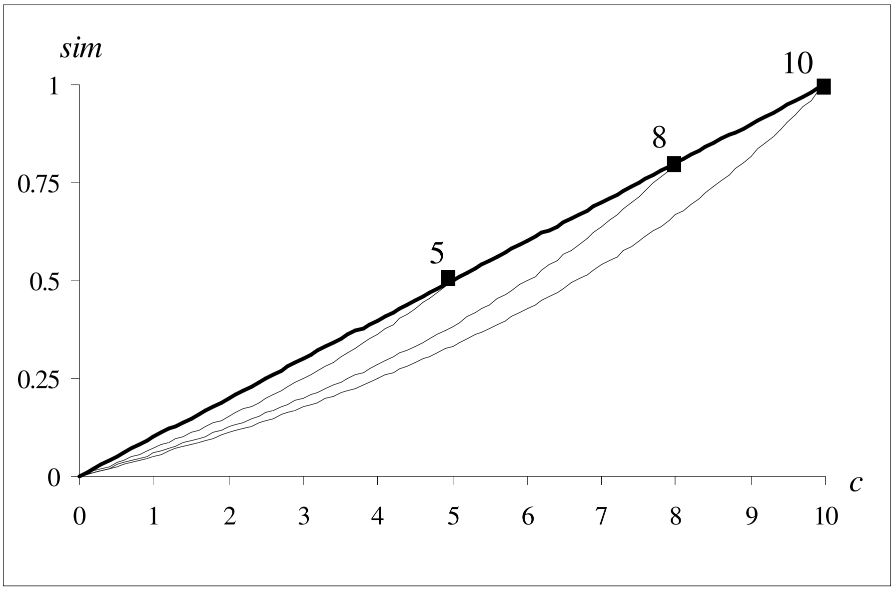


Figure 22. Deux normalisations de la mesure de similarité, en gras (32) et en clair (33)

c) Une deuxième formulation de la similarité

Arrêtons nous maintenant sur c_{12}, la partie commune des deux situations IC_1 et IC_2. Le problème d'appariement exposé dans la section précédente nous apprend d'une part que c_{12} varie selon les mises en correspondance qu'il y a entre les éléments de IC_1 et de IC_2, donc selon σ_L et σ_A, et d'autre part que l'on cherche les mises en correspondance qui maximisent c_{12}. Appelons c^*_{12} cette partie commune maximale et $|c^*_{12}|$ sa taille. Ainsi sont réunis les éléments nécessaires à la clarification de la fonction de similarité précédente (31) (cf. p. 136), qui était la première formulation du problème d'optimisation. Nous définissons sa fonction objectif, la fonction de similarité *similarité*(IC_1, IC_2), comme produit cartésien de deux espaces des contextes d'instanciation vers [0…1], tel que :

$$similarit\acute{e}(IC_1, IC_2) = \frac{|c^*_{12}|}{|IC_1| + |IC_2| - |c^*_{12}|} \qquad (35)$$

Cette mesure de similarité explicite est donc normalisée suivant la mesure de Jaccard (33). Si on a *similarité*(IC_1, IC_2) = 1, les deux situations IC_1 et IC_2 sont identiques, si par contre on a *similarité*(IC_1, IC_2) = 0, elles sont complètement dissimilaires.

d) Définition de la partie commune de deux situations

Avec cette deuxième formulation de la fonction de similarité, peu de choses sont acquises, puisqu'il reste à établir c^*_{12}. Etant fonction de σ_L et σ_A, sa taille sera définie par (36) à (40). Ainsi, $|c^*_{12}|$ est égale à la valeur maximale de la taille de la partie commune sur l'ensemble des mises en correspondance possibles. La troisième et dernière formulation du problème d'optimisation a pour fonction objectif (36).

(PSIM2)
$$\left|c_{12}^{*}\right| = \max_{\forall \sigma_L,\sigma_A}\left\{\left|c_{12}(\sigma_L,\sigma_A)\right|\right\} \tag{36}$$

avec :

$$\left|c_{12}(\sigma_L,\sigma_A)\right| = \sum_{\substack{i=1\ldots n\\ (L_{il},L_{im})\in\sigma_L}} \alpha\cdot simL(L_{il},L_{im},\sigma_A) \tag{37}$$

$$simL(L_1,L_2,\sigma_A) = \frac{1}{p}\sum_{\substack{i=1\ldots p\\ A_{1i}\in L_1,A_{2i}\in L_2}} simA(A_{1i},A_{2i},\sigma_A) \tag{38}$$

$$simA(A_1,A_2,\sigma_A) = \begin{cases} A_1 = \sigma_A(A_2), & \text{si } A_1 \text{ de type contenu d'ensemble} \\ \min\left\{ A_1 \big/ \sigma_A(A_2),\ \sigma_A(A_2) \big/ A_1 \right\}, & \text{si } A_1 \text{ de type numérique} \end{cases} \tag{39}$$

et
$$\sigma(A) = \left\{A' \middle| (A',A)\in\sigma, \text{ou } (A,A')\in\sigma\right\} \tag{40}$$

Pour des raisons de cohérence avec la définition de la taille des contextes d'instanciation $|IC|$ (34), la taille de la partie commune $|c_{12}|$ (37) est définie comme somme pondérée des littéraux qui sont mis en correspondance par σ_L, avec les mêmes coefficients α que ceux de (34). Puisque les littéraux peuvent s'apparier de manière inexacte, la définition de la taille de la partie commune (37) inclut la fonction *simL* (38), qui détermine la similarité entre deux littéraux appariés. Elle constitue, essentiellement, la moyenne des similarités entre attributs *simA*. Notons que *simL* a l'intervalle [0…1] comme domaine de valeurs. Si deux littéraux L_1 et L_2 sont identiques, et si leurs attributs sont appariés entre eux, on a un appariement exact et *simL*(L_1, L_2, σ_A) = 1. En revanche, si L_1 et L_2 sont dissimilaires, l'appariement est inexact. En conséquence, le plus L_1 et L_2 sont différents, le plus *simL*(L_1, L_2, σ_A) s'approche de 0.

La fonction de similarité entre attributs *simA* (39) a, elle aussi, comme domaine de valeurs l'intervalle [0…1]. Elle est fonction du type des attributs. Ainsi, pour des attributs de type contenu d'ensemble, *simA* = 1 si les attributs sont identiques, et *simA* = 0 dans le cas contraire. Pour les attributs numériques *simA* est égal au ratio des deux valeurs dans [0…1]. Des illustrations de cette fonction de similarité, accompagnée des heuristiques utilisées pour son évaluation, sont données plus loin.

Ces définitions de la similarité entre littéraux et entre attributs appellent quelques remarques. La première nous conduit à noter que cette mesure de similarité est du type des *similarités littérales* (“*literal similarity*”), tel que introduit par Gentner (1983). Elle compare deux descriptions en pondérant de manière équitable les entités et les relations entre les entités. Elle fait donc partie de la famille des fonctions de similarité qui sont jugées aptes à servir au raisonnement à partir de cas. Deuxième constat, la formalisation décrite ci-dessus ne correspond pas à la définition habituellement utilisée par les algorithmes des plus proches voisins. En effet, la fonction de similarité entre attributs *simA* est différente. La raison en est le problème d'échelle qui concerne les unités dans lesquelles les attributs sont représentés.

e) Le problème d'échelle des attributs

Dans les mesures de similarité habituellement utilisées dans les techniques de plus proches voisins, le *problème d'échelle des attributs* concerne la variation de la similarité entre deux valeurs d'un même attribut, selon la représentation sous-jacente de l'attribut. Pour A et B deux ensembles d'attributs, la similarité entre eux est fonction de la différence d_i entre chaque paire d'attributs A_i et B_i, $A_i \in A$, $B_i \in B$ (équation 28, cf. p. 111). Cette différence d_i est fonction de l'échelle adoptée pour la représentation de l'attribut considéré. Prenons par exemple $A_1 = 49$, $B_1 = 50$, $A_2 = 49'000$ et $B_2 = 50'000$. On a donc $d_1 = 1$ et $d_2 = 1000$, bien que les attributs puissent représenter la même valeur, simplement en différentes unités [Salzberg 1991].

Conscientes de ce problème d'échelle, et soucieuses de l'éviter, les méthodes des plus proches voisins recourent, en général, à une normalisation des attributs numériques [Aha & al. 1991]. Cependant, cette normalisation fait intervenir le domaine des valeurs de l'attribut en cause. Si ce domaine des valeurs est connu dans les systèmes d'apprentissage non-incrémentaux, il n'en va pas ainsi dans le système ici à l'étude. Dans des situations analogues, on recourt, en général, à une adaptation ad hoc de la normalisation des valeurs rencontrées au cours de l'exécution du système [Salzberg 1991], ce qui modifie la fonction de similarité au cours de l'utilisation du système. Le ratio des valeurs des attributs numériques (39), auquel nous recourons dans *Alois*, constitue une façon simple et minimaliste de remédier au problème d'échelle.

5.2.2. Restrictions sur les mises en correspondance

Nous avons jusqu'ici présenté la définition de la similarité de deux contextes d'instanciation. Ce modèle n'est pas complet en l'absence de caractérisation des espaces des mises en correspondance σ_L et σ_A. Notons que, a priori, le domaine de définition de σ_L englobe l'ensemble les mises en correspondance 1-à-1 entre tous les littéraux qui font partie de IC_1 et de IC_2. De même, le domaine de définition de σ_A comporte, quant à lui, toutes les mises en correspondance 1-à-1 entre l'ensemble des attributs des deux contextes d'instanciation.

Néanmoins, grâce au langage de représentation, par les moyens du typage des attributs ainsi que par la structure des contextes d'instanciation, il est possible d'introduire des contraintes tout à fait restrictives sur les mises en correspondance effectivement autorisées. Le but de ces contraintes est d'améliorer, si ce n'est la complexité au pire des cas du problème de mise en correspondance, tout du moins celle en moyenne.

a) Contraintes sur l'espace des mises en correspondance entre attributs

Considérons tout d'abord la mise en correspondance entre attributs. Deux contraintes peuvent être utilisées pour réduire, en pratique, l'espace des mises en correspondance. Premièrement, a la suite du typage des attributs, on peut exclure toute mise en correspondance entre attributs qui ne sont pas du même type [Ganascia 1987]. Sémantiquement, un tel appariement n'aurait pas de sens. Prenons comme exemple le problème de placement et d'allocation (cf. p. 166). Le domaine contient des attributs objet de type `Warehouse` et de type `Client`. Compte tenu des rôles orthogonaux de ces deux entités dans le problème, un appariement entre eux n'est guère envisageable. On peut faire le même constat pour des attributs valeur de type `Flow` et `Cost`.

Deuxièmement, chaque attribut a une position fixe à l'intérieur d'un littéral. Cette position n'est pas dénuée de sens ; elle est choisie par l'utilisateur lors de la modélisation du problème. Envisager un appariement entre deux attributs qui, dans un même littéral, occupent deux positions différentes serait maladroite. En conséquence, on peut exclure toutes ces mises en correspondance de l'espace de définition de σ_A.

b) Contraintes sur l'espace des mises en correspondance entre littéraux

Nous mettons en oeuvre dans *Alois* deux contraintes sur l'espace des mises en correspondance entre littéraux. Premièrement, en ce qui concerne σ_L, on peut immédiatement exclure de son domaine de définition tout mise en correspondance entre littéraux de noms différents [Ganascia 1987]. Deuxièmement, les contextes d'instanciation sont organisés en strates. Cette structure a une sémantique (cf. p. 95) qui peut être utilisée afin d'orienter la construction des mise en correspondance. D'abord, la première strate, qui correspond à la racine des graphes si l'on considère la représentation graphique, peut contenir des littéraux du type `rôle`. Ces littéraux sont introduits par l'utilisateur. Dans cette hypothèse, ces littéraux (et aussi les attributs qu'ils contiennent) peuvent être appariés par défaut, avant tout processus de mise en correspondance. Ensuite, la structuration en strates autorise l'exclusion, a priori, de certaines mises en correspondance. Essentiellement, on peut exclure toute mise en correspondance entre deux littéraux qui ne font pas partie de la même strate.

Considérons un exemple du problème de la coloration de graphes (cf. p. 187). Deux contexte d'instanciation du schéma d'opérateur `recolorNode` sont présentés dans la figure 24 (cf. p. 154). La première strate contient le littéral `role` qui identifie le noeud qui vient d'être recoloré, et qui peut, lors d'une comparaison avec un autre contexte d'instanciation, directement être apparié. Le littéral `node` décrit la couleur et le degré d'un noeud. Il est présent dans les strates deux et trois. Néanmoins, il est clair qu'une mise en correspondance entre ces littéraux, appartenant à des strates différentes, ne serait pas un choix souhaitable.

Précédemment (cf. p. 136), nous avons remarqué que le problème de construction du sous-graphe partiel maximal entre deux graphes orientés, étiquetés et racinés est NP-difficile. Les contraintes sur les espaces de mises en correspondance, que nous avons introduites ci-dessus, ne changent en rien la complexité du problème, pas plus que le fait d'avoir affaire à des graphes racinés [Fu 1995]. Toutes ces considérations concernent la complexité au pire des cas (“*worst case complexity*”). L'introduction du typage des attributs, et les contraintes sur l'espace des mises en correspondance, peut mener à une réduction significative de la complexité réelle [Ganascia 1987, Haussler 1989]. Grâce à ces contraintes, nous verrons ci-dessous que les contextes d'instanciation peuvent, dans certains cas, s'apparier avec une simple heuristique gloutonne. Ainsi, par exemple pour le problème d'ordonnancement (cf. p. 181), il n'est plus du tout besoin d'une technique élaborée de mise en correspondance.

5.2.3. Autres modélisations de la similarité

Différentes mesures de similarité explicites ont été définies pour appréhender la ressemblance de deux descriptions structurées. Nous en examinons quatre. Ces approches peuvent être divisées en deux groupes [Geibel & Wysotzki 1996]. Les deux premières, ainsi que la nôtre, s'intéressent à la similarité entre deux descriptions dans leur entier, tandis que les deux dernières considèrent la similarité de deux descriptions comme étant déterminée par la similarité des entités qui les composent, en prenant en compte la ressemblance des structures dans lesquelles se trouvent les entités.

a) Les distances d'édition de graphes

Dans les métriques d'*édition de graphes*, aussi appelées distances de *Levenshtein* [Thomason 1990], le calcul de la ressemblance entre deux exemples est conçu comme étant un problème de recherche dans un espace d'états, où les deux exemples constituent respectivement l'état initial et l'état final. Des opérateurs, tels que l'ajout ou la suppression de noeuds et d'arcs, permettent de passer d'un état à un autre. En attribuant des poids aux différents opérateurs, la distance entre deux exemples correspond à la somme des poids des opérateurs qui sont nécessaires à la transformation d'un exemple en un autre. Ces distances sont très utilisées en reconnaissance de formes [Bunke & Sanfeliu 1990], et plus rarement en apprentissage symbolique ou en raisonnement à partir de cas [Bunke & Messmer 1993].

b) L'appariement flexible

La technique d'appariement inexact, qui est appelée *appariement flexible* [Esposito & al. 1991], fait appel à une mesure de similarité qui, comme la notre, calcule la ressemblance entre deux descriptions structurées en fonction d'une mise en correspondance. Contrairement aux distances d'édition de graphes, aucune recherche n'est alors nécessaire. La mesure de similarité ainsi employée est originale en ce sens qu'elle s'appuie sur des distributions de probabilités pour établir la ressemblance entre les attributs.

c) Le système KBG

Le système *KBG* [Bisson 1992] emploie une mesure de similarité entre attributs qui n'inclut pas seulement les deux attributs à comparer, mais qui englobe aussi des termes pour les entités qui sont directement connectées aux attributs à l'étude. De cette manière, la structure des deux descriptions est prise en compte au niveau de la modélisation de la similarité entre attributs. Cela s'avère particulièrement intéressant car la mesure de similarité définit, ainsi, un système d'équations qui, une fois résolu, fournit l'ensemble des similarités entre les attributs qui sont appariables.

d) L'approche IBL relationnel

Nous avons auparavant présenté la distance euclidienne telle qu'elle est, en général, employée en *apprentissage à partir d'instances* (chapitre 3, cf. p. 109). Cette distance a pour limite majeure de ne s'appliquer qu'aux descriptions attribut-valeur en ordre 0 ou 0+, et non aux descriptions structurées, représentées en langages d'ordre 1. Ce n'est que récemment que Emde & Wettschereck (1996) l'ont étendue de telle sorte qu'elle s'applique à ce dernier type de représentations. Mise en oeuvre dans le système *RIBL*, celle que nous proposons lui ressemble fortement, puisqu'elle calcule la similarité entre deux structures en fonction de la similarité entre les littéraux qui les composent. La similarité entre deux littéraux dépend de la similarité entre les attributs qui y sont présents.

5.3. Le calcul de la similarité

Au cours des précédentes sections de ce chapitre, nous avons traité de la modélisation de la similarité telle que nous l'employons dans notre système *Alois*. Nous verrons à présent son calcul. En raison de la généricité du problème d'appariement d'objets structurés, il existent d'importants travaux portant soit directement sur ce problème, soit sur les diverses variantes du problème d'isomorphisme de graphes. Les deux problèmes étant NP-complet, il y a eu développement de deux types d'approches : l'une s'appuyant sur des algorithmes exacts, mais peu efficaces, l'autre utilisant des techniques heuristiques. Nous présenterons quelques unes de ces techniques dans la première sous-section. La fin de cette section, ainsi que la prochaine section, traiteront des techniques d'appariement que nous utilisons dans *Alois*.

5.3.1. Techniques de calcul de la similarité

Dans cette sous-section, nous traitons de différentes approches exactes et heuristiques pour la résolution du problème d'appariement et de celui de l'isomorphisme de graphes. Les travaux présentés ont pour origine différents domaines tels que l'apprentissage symbolique, la reconnaissance de formes mais aussi la recherche opérationnelle.

a) Les algorithmes

Tout d'abord, il convient de remarquer que l'ensemble des algorithmes présentés se heurtent à la complexité inhérente aux problèmes traités. Pourtant, cette complexité *au pire* n'empêche pas certains algorithmes d'être remarquablement efficaces sur des variantes particulières de ces problèmes. Un bon exemple en est le réseau *Rete* [Forgy 1982] qui traite du problème d'appariement exact, tel qu'il se présente dans les systèmes à base de règles[7]. En effet, lorsque ces systèmes représentent les schémas d'opérateur en un langage d'ordre 1, la détermination des opérateurs applicables nécessite un appariement impliquant les littéraux contenus dans les conditions des opérateurs et dans la base de faits [Hayes-Roth & al. 1978]. En factorisant les conditions des opérateurs et en les organisant en un réseau de jointures, *Rete* détermine les opérateurs applicables à chaque modification de la base de faits, modifications qui se produisent typiquement lors du déclenchement d'un opérateur. Si ces modifications concernent seulement des sous-parties du réseau de jointures, cette factorisation est un bon moyen d'accélérer l'appariement et donc de déterminer les opérateurs applicables.

---

[7] *Clips*, le système utilisé dans *Alois* pour la représentation et l'application des opérateurs, emploie également un algorithme basé sur *Rete* [Riley 1991].

Les algorithmes d'apprentissage symbolique recourent aussi à des techniques d'appariement. En effet, on peut constater que l'apprentissage se caractérise par une combinaison d'appariement et de généralisation [Ganascia 1987, Zucker & Ganascia 1996]. Le but de l'algorithme d'appariement est alors de proposer des mises en correspondance avec lesquelles l'algorithme de généralisation sait produire des généralisations intéressantes. L'algorithme d'appariement effectue souvent une exploration exhaustive de l'espace des mises en correspondance. Cet espace étant vaste, le processus d'exploration est en général guidé par une heuristique[8], selon que l'on subodore la présence d'appariements intéressants parmi les plus grandes mises en correspondance [Hayes-Roth & McDermott 1977, Kodratoff & Ganascia 1986] ou parmi les plus petites [Zucker & Ganascia 1996].

Dans leur approche appelée *appariement flexible*, Esposito & al. (1991) proposent une exploration complète de l'espace des mises en correspondance. Conscients de la complexité d'une telle approche, ils utilisent un algorithme du type Branch & Bound afin d'exclure certaines régions de l'espace de recherche, dont on sait qu'elles ne contiennent pas la mise en correspondance optimale. L'algorithme est alors plus rapide qu'une recherche exhaustive[9]. Néanmoins, cette réduction de la complexité réelle n'est pas suffisante pour l'application envisagée par les auteurs. Ils proposent donc une heuristique simple qui, en réorganisant la description des exemples, produit des résultats d'une qualité satisfaisante en un temps acceptable. Toutefois, les auteurs ne donnent pas d'informations permettant de juger de la validité générale de leur heuristique.

Le système de génération de plans *Prodigy/Analogy* fait lui aussi appel à une procédure d'appariement lors de la récupération de cas [Veloso 1994]. L'algorithme procède par énumération de l'ensemble des mises en correspondance suivant une stratégie du type “meilleur d'abord” avec retour en arrière. Néanmoins, l'algorithme s'arrête dès lors qu'une mise en correspondance de qualité acceptable est trouvée.

En reconnaissance de formes, l'utilisation de la *distance d'édition de graphes* pour la comparaison de deux descriptions structurées est très commune. Des algorithmes classiques de recherche d'états, tel que A*, sont employés pour leur calcul [Shapiro & Haralick 1990, Vosselmann 1992]. Une autre approche du calcul des distances de graphes se consacre explicitement au problème d'identification du graphe, parmi un ensemble de graphes en mémoire, auquel s'apparie un graphe de référence [Messmer & Bunke 1993]. Le problème est alors le même que celui rencontré lors de la phase de recherche de cas dans la mémoire de système de raisonnement à partir de cas (chapitre 4, cf. p. 105).

---

[8] Ici, le mot heuristique est employé dans le sens qu'il a en intelligence artificielle (cf. p. 24).

[9] Bien que la complexité au pire des cas reste bien évidemment inchangée.

Une autre famille d'algorithmes, pour le problème d'appariement, est connue sous le nom *d'algorithmes d'étiquetage* [Köbler & al. 1993]. Ils consistent à attribuer à chaque noeud une étiquette le caractérisant. Il est ainsi possible d'enlever de l'espace des mises en correspondance, chaque élément qui met en relation deux noeuds ayant des étiquettes différentes. L'espace de recherche se trouve alors réduit. Une telle approche est utilisée par Geibel & Wysotzki (1996) pour l'apprentissage inductif de la catégorisation d'objets structurés. Ils décrivent les objets structurés par les étiquettes des éléments qui les composent. Puisque cette description peut être représentée en un langage propositionnel, des algorithmes inductifs classiques tels que ID3 sont applicables.

Il y a eu différents développements d'algorithmes efficaces pour des problèmes qui sont de complexité polynomiale. Il en va ainsi tant pour l'appariement entre graphes racinés orientés[10] et ordonnés [Fu 1995] que pour l'appariement entre arbres ordonnés [Shasha & al. 1994].

En recherche opérationnelle, il existe un *problème d'appariement* qui est différent de celui ici étudié. Etant donné un graphe $G = (N, A)$, ce problème concerne la mise en correspondance des noeuds de N entre eux. Plus précisément, sont appariables deux noeuds n_1 et n_2 de N, s'il existe un arc $a = (n_1, n_2)$ dans A, sous la contrainte que chaque noeud ne puisse être apparié qu'une seule fois. Le but est alors d'identifier la mise en correspondance qui maximise le nombre de noeuds appariés [Padberg 1979]. Ce problème d'appariement est de complexité polynomiale. Plus avant dans ce chapitre, nous baserons une heuristique d'évaluation de la similarité, dans *Alois,* sur un cas spécial de ce problème, celui de l'*appariement bipartite* (cf. p. 156).

b) Les heuristiques

Il existe un nombre important d'heuristiques qui traitent du problème d'appariement. Certaines d'entre elles sont des modifications des techniques exactes présentées ci-dessus. D'autres encore emploient des heuristiques d'optimisation, telles que le recuit simulé [Shasha & al. 1994], afin d'identifier de bonnes solutions. En ce qui concerne l'apprentissage symbolique, deux approches heuristiques méritent notre attention.

Considérons tout d'abord, le système *KBG* [Bisson 1992] qui utilise une heuristique pour calculer la similarité entre chaque paire d'entités appariables, en prenant en compte la structure dans laquelle se trouve les entités. Si l'on représente la similarité de chaque paire d'entités par une équation, calculer la similarité entre les entités revient à résoudre un système d'équations non-linéaires. Ensuite un algorithme glouton itératif, qui choisit à chaque étape la

---

[10] A titre de rappel, indiquons qu'il existe, dans les graphes ordonnés, un ordre préétabli entre tous les noeuds fils du même père.

mise en correspondance présentant la similarité maximale, est utilisé pour réaliser l'appariement entre les éléments des deux structures.

Contrairement au système *KBG* qui compare deux conjonctions de littéraux, l'approche retenue par *RIBL* [Emde & Wettschereck 1996] ne compare que deux littéraux de même nom. Néanmoins, *RIBL* prend en compte le contexte dans lequel se trouvent ces littéraux. Ce contexte est défini de la même manière que le contexte d'instanciation dans *Alois* (cf. p. 95). Pour ce qui est des mises en correspondance, le fait de se limiter dès l'origine à deux littéraux facilite beaucoup la problématique. En effet, *RIBL* suppose que les éléments qui se trouvent à la même position dans chacun de ces deux littéraux soient d'office appariés entre eux. Pour ce qui est des mises en correspondance qui, dans le contexte, restent libre ; un simple algorithme glouton en apparie à chaque pas le meilleur couple.

5.3.2. Une première heuristique gloutonne

Nous avons jusqu'ici évoqué diverses techniques relatives à la détermination de la similarité et des appariements. Cependant, notre système pose des contraintes de telle sorte qu'aucune méthode existante ne s'y applique de manière satisfaisante. Ces contraintes sont :

a) L'évaluation fréquente de la similarité : au cours d'une seule optimisation, la fonction de similarité peut être appelée des milliers, voire des dizaines de milliers de fois. Cela suppose que cette fonction soit *très* rapide.

b) Le caractère particulier de la modélisation de la similarité : elle comporte une organisation en strates des descriptions à comparer (de manière similaire à *RIBL* [Emde & Wettschereck 1996]). Si sa prise en compte ne réduit pas la complexité au pire, elle permet tout de même de diminuer de manière considérable l'espace des mises en correspondance et peut donc entraîner une réduction de la complexité réelle opérationnelle.

Puisque nous ne connaissons pas de technique d'évaluation de la similarité qui prenne en compte ces deux particularités simultanément, nous en proposons une nouvelle qui est une heuristique gloutonne simple et rapide. Elle permet de produire des appariements de bonne qualité pour le problème d'ordonnancement (nous le verrons sur un exemple), et pour celui de placement et d'allocation d'entités. Toutefois, comme nous le constaterons, cette heuristique échoue sur le problème de la coloration de graphes en raison de sa simplicité. C'est alors une deuxième heuristique plus complexe que nous employons dans l'optimisation de ce problème. Introduite dans la section prochaine, elle sera basée sur une relaxation du problème d'appariement entre objets structurés vers le problème de l'*appariement bipartite* qui est de complexité polynomiale.

a) L'heuristique

Notre heuristique gloutonne d'appariement s'attache au problème de déterminer les deux mises en correspondance σ_L et σ_A de sorte que la comparaison de deux contextes d'instanciation IC_1 et IC_2 rende une similarité maximale. Il s'agit donc de résoudre le problème d'optimisation PSIM2 tel que défini par les équations (36) à (40), (cf. p. 141). Conformément à l'usage en raisonnement à partir de cas, nous appelons ces deux contextes d'instanciation `Source` et `Cible`.

L'algorithme *GloutonSimilarité* détermine la partie commune entre la `Source` et la `Cible`, et détermine la taille de cette partie commune, `commun` (tableau 22). Il commence par les deux mises en correspondance σ_L et σ_A vides. Les littéraux sont appariés l'un après l'autre en choisissant à chaque étape la meilleure mise en correspondance candidate. Les choix de mises en correspondance ne sont, une fois arrêtés, plus mis en cause. L'index `[i]` représente la strate `i` d'un contexte d'instanciation. L'heuristique commence à la strate minimale, et fait, pour chaque strate, une boucle sur l'ensemble des littéraux de `Source`. Pour chaque littéral ainsi considéré `Prochain`, le littéral de `Cible` qui est le plus ressemblant du même nom et de la même strate, `Meilleur`, lui est apparié. Le couple est ajouté à σ_L. Les attributs contenus dans ces deux littéraux sont eux aussi appariés entre eux et ajoutés à σ_A. Les attributs sont seulement appariés si les *deux* attributs occupant la même position sont libres encore.

```
fonction GloutonSimilarité(Source, Cible): commun
initialise : i := 1; $\sigma_A$ := {}, $\sigma_L$ := {}, commun := 0
tant que (i ≤ nombre de strates de Source) faire
    tant que (Source[i] non vide et Cible[i] non vide) faire
        Prochain := 1$^{er}$ littéral de Source[i]
        candidats := littéraux de Cible[i] du même nom que Prochain
        Meilleur := lit ∊ candidats avec simL(lit, Prochain, $\sigma_A$) maximal
        $\sigma_L$ := $\sigma_L$ + (Prochain, Meilleur)
        apparie dans l'ordre, les attributs libres de Prochain et Meilleur, ajouts à $\sigma_A$
        commun := commun + $\alpha_{Prochain} \cdot$ simL(Meilleur, Prochain, $\sigma_A$)
        enlève Meilleur de Cible[i] et Prochain de Source[i]
    fin tant que
    i := i + 1
fin tant que
renvoi commun
```

Tableau 22. L'heuristique gloutonne pour la détermination du sous-graphe partiel maximal

Le choix du littéral `Meilleur`, qui est, parmi l'ensemble des littéraux du même nom et de la même strate, `candidats`, le plus proche du littéral considéré `Prochain`, appelle quelques précisions (tableau 22, ligne 6). En effet, la fonction de similarité entre attributs *simA* (39), appelée par *simL* (38), compare, à ce stade, des attributs qui ne sont pas nécessairement appariés. Si les *deux* attributs arguments de *simA* ne sont pas appariés, *simA* va les considérer comme si ils l'étaient entre eux.

La complexité de cette heuristique, au pire des cas, est de $O(n^2)$, n le nombre maximal de littéraux contenus dans `Source` ou `Cible`. Cette complexité est atteinte pour une description dégénérée où les contextes d'instanciation n'ont qu'une seule strate et où tous les littéraux ont le même nom. La borne inférieure de la complexité réelle d'exécution est de $O(n)$. Elle est atteinte lorsque chaque strate ne comporte qu'un seul littéral du même nom.

Au chapitre 3, nous avons présenté différents types de mesures de similarité (cf. p. 110). Ainsi, une mesure de similarité est une *métrique* si elle satisfait aux conditions de minimalité, de symétrie et à l'inégalité triangulaire. La similarité calculée par l'heuristique *GloutonSimilarité* ne satisfait à aucune de ces conditions. Apparemment, elle n'est pas symétrique. En outre s'agissant d'une heuristique, elle ne satisfait pas nécessairement à la minimalité. Ces deux conditions n'étant pas remplies, l'inégalité triangulaire ne peut pas non plus être respectée.

b) Un exemple du problème d'ordonnancement

Considérons l'opérateur `enlève` du problème d'ordonnancement (cf. p. 181). Le tableau 23 présente à gauche, le contexte d'instanciation d'un cas `Source` et, à droite celui d'un opérateur applicable `Cible`. Ils sont composés de trois strates chacun. Le calcul de la similarité à l'aide de l'heuristique *GloutonSimilarité* se déroule de manière suivante :

1. Considération de la strate `i` = 1 qui ne contient qu'un seul littéral. Il y a donc appariement entre les deux seuls attributs `Jobs_2` et `job1` (tableau 22, ligne 8) et `commun` = 1.
2. Passage à la strate `i` = 2, `Prochain` = (before `Jobs_1 Jobs_2 14`) et le littéral de cible qui lui est le plus proche est, car `Jobs_2` et `job1` sont déjà appariés, `Meilleur` = (before `job4 job1 17`). En conséquence, il y a appariement entre `Jobs_1` et `job4` et entre `14` et `17` (ligne 8). La similarité entre ces deux littéraux est :

$$simL = \frac{1}{3} \cdot \left(1 + 1 + {}^{14}\!/_{17}\right) = 0.94 \qquad (41)$$

   ce qui donne `commun` = 1 + 0.94 = 1.94 (ligne 9).
3. Toujours dans `i` = 2, les deux littéraux (before …) sont considérés et leurs attributs sont appariés, c'est à dire `Jobs_3` et `job8` ainsi que `23` et `23`. La similarité entre ces littéraux est *simL* = 1, et on a `commun` = 2.94.
4. Ensuite est considéré `Prochain` = (state `Jobs_2 JobState_1 16 387`). Puisque `Jobs_2` et `job1` sont déjà appariés entre eux, le littéral le plus similaire de `Ci-`

`ble` est `Meilleur` = (state `job1 scheduled 18 445`) et il y a appariement entre `JobState_1` et `scheduled`, entre `16` et `18` ainsi qu'entre `387` et `445`.

5. L'heuristique procède ainsi de suite. La valeur finale de `commun` sera 7.54.

Une fois cette heuristique terminée, la valeur de la similarité entre ces deux situations est déterminée par une normalisation dans [0…1]. Ci-dessus, nous avions opté pour l'utilisation de la mesure de Jaccard comme normalisation (33) (cf. p. 139). On obtient donc comme similarité normalisée :

$$sim = \frac{7.54}{8+8-7.54} = 0.89 \tag{42}$$

On peut constater que l'heuristique *GloutonSimilarité* produit un appariement, entre ces deux situations, avec très peu de recherche. En effet, les appariement partiels qu'elle produit à chaque itération lui permettent de construire un appariement final qui nous parait être l'appariement optimal. D'après nos expériences, tel est toujours le cas dans notre domaine d'ordonnancement. Cependant, il n'en va pas ainsi dans le problème de la coloration de graphes que nous considérerons ci-dessous.

Cas `Source` :	Opérateur `Cible` :
Strate 1	Strate 1
(role Jobs_2)	(role job1)
Strate 2	Strate 2
(before Jobs_1 Jobs_2 14)	(before job4 job1 17)
(before Jobs_2 Jobs_3 23)	(before job1 job8 23)
(state Jobs_2 JobState_1 16 387)	(state job1 scheduled 18 445)
(state Jobs_3 JobState_1 17 420)	(state job8 scheduled 19 483)
Strate 3	Strate 3
(state Jobs_1 JobState_1 15 348)	(state job4 scheduled 17 399)
(before Jobs_4 Jobs_1 11)	(before job5 job4 19)
(before Jobs_3 Jobs_5 19)	(before job8 job17 25)

Tableau 23. Deux situations du problème d'ordonnancement relatives à l'opérateur `enlève`

c) Un exemple du problème de la coloration de graphes

Prenons à présent l'opérateur `recolorNode` du problème de la coloration de graphes (cf. p. 187). Le tableau 24 présente à gauche, le contexte d'instanciation d'un cas `Source` et à droite, celui d'un opérateur applicable `Cible`. Ils sont composés de trois strates chacun. Le calcul de la similarité à l'aide de l'heuristique *GloutonSimilarité* se déroule de la manière suivante :

1. Considération de la strate `i` = 1 qui ne contient qu'un seul littéral. Il y a donc appariement entre les deux seuls attributs `N_1` et `n86` (tableau 22, ligne 8) et `commun` = 1.

2. Dans la strate `i` = 2, le premier littéral considéré est `Prochain` = (node `N_1 Color_1 49`). Le littéral de `Cible` qui en est le plus proche est `Meilleur` = (node `n86 col_6 48`) ce qui conduit aux appariements entre `Color_1` et `col_6` et entre `49` et `48`. La similarité entre ces deux littéraux est *simL* = 0.99.
3. Ensuite, on a `Prochain` = (neighbours `N_1 N_2 N_3 N_4 N_5`) et `Meilleur` = (neighbours `n86 n27 n26 n2`). Si `N_1` et `n86` sont déjà appariés entre eux, tous les autres noeuds sont encore libres, et doivent en conséquence être maintenant appariés. Cependant, à cet instant, l'heuristique ne dispose pas d'informations portant sur ces noeuds autre que leur position dans les littéraux. Comme nous l'avons souligné dans la définition du domaine (cf. p. 187), la deuxième position du littéral "neighbours" est multi-valuée. En conséquence, l'ensemble des attributs `N_x` et `nx` occupent la *même* position. L'appariement entre eux n'est alors pas du tout spécifié, en l'occurrence l'heuristique apparie simplement de gauche à droite.

C'est à ce moment là que l'heuristique *GloutonSimilarité* échoue. En effet, il y a bel et bien des informations disponibles permettant d'orienter l'appariement entre les différents noeuds, seulement, ces informations se trouvent dans la strate 3. En conséquence, l'heuristique les ignore lorsqu'elle apparie les attributs qui interviennent dans la strate 2. Elle s'arrête donc trop vite sur ses choix de mises en correspondance.

Cas `Source` :	Opérateur `Cible` :
Strate 1	Strate 1
(role N_1)	(role n86)
Starte 2	Strate 2
(node N_1 Color_1 49)	(node n86 col_6 48)
(neighbours N_1 N_2 N_3 N_4 N_5)	(neighbours n86 n27 n26 n2)
Starte 3	Starte 3
(node N_3 Color_1 49)	(node n27 col_6 57)
(node N_2 Color_1 49)	(node n26 col_6 52)
(node N_4 Color_1 43)	(node n2 col_6 51)
(node N_5 Color_1 47)	(neighbours n73 n26)
(neighbours N_2 N_3 N_6 N_1)	(neighbours n26 n73 n86 n2)
(neighbours N_3 N_2 N_1 N_4)	(neighbours n27 n86 n2)
(neighbours N_4 N_1 N_3)	(neighbours n2 n26 n86 n27)
(neighbours N_6 N_2 N_5)	
(neighbours N_5 N_1 N_6)	

Tableau 24. Deux situations du problème de coloration de graphes

L'heuristique *GloutonSimilarité* ne permet donc pas, pour l'ensemble de nos problèmes d'expérimentation, l'identification des appariements de bonne qualité. Il en découle le besoin d'introduire une technique d'appariement qui, elle, donne des résultats plus robustes.

5.4. La deuxième heuristique du calcul de la similarité

L'objet de cette section est de proposer une heuristique qui, au cas où la première heuristique gloutonne échoue, permet la construction d'appariements de bonne qualité, au (léger) détriment de l'efficacité. Compte tenu des impératifs de performance, nous persistons à n'utiliser ni des algorithmes exacts ni des heuristiques autorisant une mise en cause des appariements antérieurs suivant une technique de retour en arrière. Ceci limite fortement le choix des approches possibles.

Nous allons donc proposer une heuristique basée sur une *relaxation* du problème d'origine. La relaxation est une technique de construction d'algorithmes classique en recherche opérationnelle. Lorsque l'on a affaire à un problème *P* qui est difficile à résoudre, on tente de relâcher quelques contraintes de *P* afin d'obtenir un problème *P'* qui est plus simple, et pour lequel on dispose d'une technique de résolution efficace. *P'* est alors la relaxation de *P*. On résout alors *P'*. Evidemment, la solution de *P'* n'est en général pas une solution de *P* (ou du moins pas la solution optimale). Il existe alors deux possibilités : soit on essaye d'adapter la solution de *P'* de sorte qu'elle devienne admissible pour *P*, soit on reformule *P* en réduisant sa région de faisabilité de manière à s'approcher de la solution désirée. Dans le premier cas on obtient une heuristique, dans le second cas la résolution peut être exacte au bout de plusieurs itérations. Il est clair que l'exemple le plus classique d'une résolution itérative par relaxation porte sur les problèmes de la programmation (mixte -) entière, résolus par une combinaison de l'algorithme du Simplexe et une technique de Branch & Bound (chapitre 1, cf. p. 18).

Dans la section précédente, nous avons présenté différentes techniques efficaces qui résolvent des problèmes moins complexes que celui de l'isomorphisme de sous-graphes partiels maximaux. Ainsi, les problèmes d'appariement entre graphes ordonnés [Fu 1995] ou entre arbres ordonnés [Shasha & al. 1994]sont de complexité polynomiale. De la même manière, le problème d'appariement bipartite est un problème polynomial [Williams 1993b]. On a donc l'embarras du choix en matière de problème de relaxation *P'*. Le passage d'un graphe non-ordonné vers un graphe ordonné nous parait, en effet, non-trivial. De même, le passage d'un graphe orienté vers un arbre n'est pas davantage évident (cf. [Bournaud & Ganascia 1995]).

Pour le calcul de la similarité, nous proposons donc de recourir à une heuristique basée sur une relaxation du problème initial sur le problème d'appariement bipartite. Cette heuristique nous donnera une évaluation de la similarité entre deux situations qui, nous l'espérons, sera de bonne qualité. Si cette relaxation a été conçue aux seules fins de satisfaire aux besoins immédiats de *Alois*, il pourrait néanmoins s'avérer intéressant de tenter de l'appliquer et de l'évaluer dans un contexte plus général.

Dans cette section, nous présenterons le problème de l'appariement bipartite avant de décrire la relaxation du problème d'isomorphisme, inexact maximal entre sous-graphes partiels, sur l'appariement bipartite. Ensuite, nous décrirons une heuristique utile à la résolution du problème relaxé et à la construction de la solution du problème d'origine, à partir de la solution du problème relaxé.

5.4.1. L'appariement bipartite

Le problème de *l'appariement bipartite* est classique en recherche opérationnelle. Un graphe bipartite est un graphe $G = (N, N', A)$, composé de deux ensembles de noeuds, N et N', et d'un ensemble d'arcs A, où tous les arc $a_{ij} \in A$ donnent de N vers N'. Le problème consiste alors en la sélection d'un nombre maximal d'arcs tel que tout noeud n'ait pas plus qu'un arc adjacent sélectionné (figure 23). Une formulation mathématique avec $m = |N|$ et $n = |N'|$ est donnée en (43)-(46).

(BIP) maximise : $$f = \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij} \cdot x_{ij} \qquad (43)$$

avec :

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad \forall j \qquad (44)$$

$$\sum_{j=1}^{n} x_{ij} \leq 1 \quad \forall i \qquad (45)$$

$$x_{ij} \in \{0,1\} \quad \forall i, j \qquad (46)$$

Les variables du problème sont les variables pseudo-booléennes x_{ij} (46). Chacune représente la sélection d'un arc a_{ij}, avec $x_{ij} = 1$ pour l'arc sélectionné. La fonction objectif (43) est la somme pondérée des arcs sélectionnés. Les coefficients w_{ij} correspondent au poids affecté à la sélection d'un arc. Les contraintes (44) et (45) assurent que pas plus d'un seul arc ne sera sélectionné pour chaque noeud. Elles introduisent une disjonction entre les variables issues du même noeud, avec néanmoins la possibilité que chacune des variables vaille 0.

Le problème d'appariement bipartite est polynomial ; il existe un algorithme de complexité $O(n^{5/2})$, n le nombre maximal de noeuds [Hopcroft & Karp 1973]. En outre, la matrice des coefficients des contraintes, définie par (44) et (45), est unimodale. En conséquence, les algorithmes de la programmation linéaire construisent d'office des solutions entières (chapitre 1, cf. p. 23). La résolution d'un problème unimodal est alors considérée comme étant facile. En outre, le problème d'appariement bipartite est un cas spécial du *problème d'affectation* pour lequel existe l'élégante *méthode hongroise* en $O(n^3)$ [Williams 1993b].

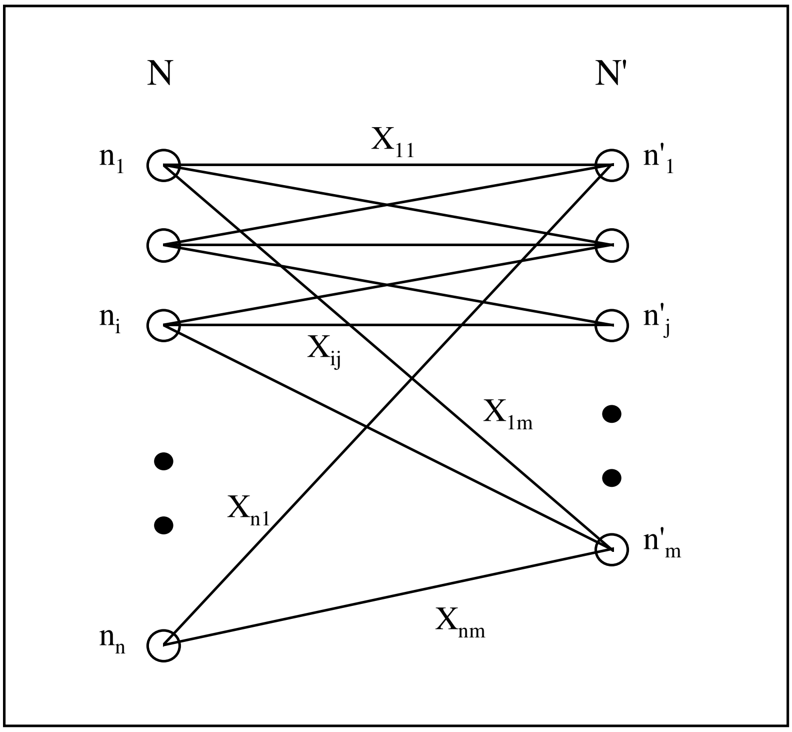


Figure 23. L'appariement bipartite

5.4.2. La relaxation

Il existe plusieurs manières de procéder à une relaxation du problème d'isomorphisme, tel que défini par (37) - (40) (cf. p. 141), au problème d'appariement bipartite. Nous en étudierons trois ; et pour la troisième, nous proposerons une heuristique. Nous associons chacune des deux situations à comparer à une partie d'un graphe bipartite, ceci pour les trois relaxations. Ce qui varie de l'une à l'autre c'est la manière dont se déroule cette association. Concrètement, la fonction *relax* met en relation un contexte d'instanciation *IC* avec un ensemble de noeuds *N*. Le graphe bipartite se formule dès lors $G = (relax(IC_1), relax(IC_2), A)$. Spécifier une relaxation revient alors à définir conjointement cette fonction *relax*, un ensemble d'arcs *A*, qui constituent les variables du problème d'appariement bipartite, et les poids w_{ij} de ces variables dans la fonction objectif. En outre, nous exigeons que l'ensemble des arcs A respecte toujours les contraintes sur les mises en correspondance introduites auparavant (cf. p. 142).

a) La relaxation directe

Une première relaxation que nous appelons $relax_1$ nous parait être évidente. Elle consiste à considérer les entités d'un contexte d'instanciation comme noeuds d'une partie du graphe bipartite. Le domaine de définition de $relax_1$ est alors composé de l'ensemble des attributs objet de *IC*.

Dans le monde des polygones introduit précédemment (figure 19, cf. p. 129), les noeuds du graphe bipartite correspondent aux polygones A_i et B_j. Avec N représentant la scène A et N' représentant la scène B, les polygones A_i correspondent aux noeuds n_i, et les polygones B_j aux noeuds n'_j (figure 24). Ce problème d'optimisation se formule alors avec (47) - (53), où les contraintes (48) et (49) reflètent l'équation (44) de la formulation précédente du problème d'appariement bipartite, et les contraintes (50) - (52) à (45) :

$$\text{maximise : } f = \sum_{i=1}^{3} \sum_{j=1}^{2} w_{ij} \cdot x_{ij} \tag{47}$$

avec :

$$x_{11} + x_{21} + x_{31} \leq 1 \tag{48}$$

$$x_{12} + x_{22} + x_{32} \leq 1 \tag{49}$$

$$x_{11} + x_{12} \leq 1 \tag{50}$$

$$x_{21} + x_{22} \leq 1 \tag{51}$$

$$x_{31} + x_{32} \leq 1 \tag{52}$$

$$x_{ij} \in \{0,1\} \quad \forall i, j \tag{53}$$

Dans cette relaxation, la distinction d'un arc x_{ij} par rapport à un autre se fait par l'intermédiaire des poids w_{ij} au niveau de la fonction objectif (47). Ces poids représentent la similarité de deux entités, tout autant en ce qui concerne la similitude de leurs attributs que la ressemblance de leurs structures.

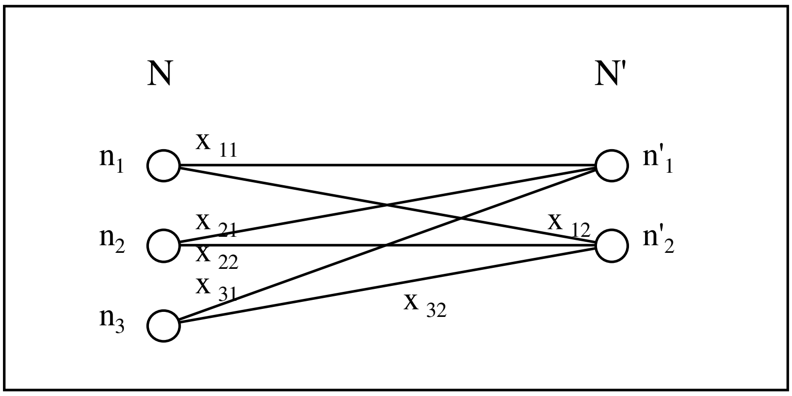


Figure 24. Exemple graphique de l'appariement bipartite, équations (47) - (53)

Néanmoins, les poids w_{ij} peuvent eux-mêmes être fonction d'un appariement partiel. Illustrons ceci par l'exemple des polygones (figure 19). Supposons que les poids entre polygones identiques soient maximaux, $w_{11} = 1$ et $w_{32} = 1$, et supposons que l'on a, en conséquence, retenu la mise en correspondance $x_{32} = 1$[11]. Dès lors, et découlant de la structure entre les polygones, la mise en correspondance x_{21} regagne en intérêt. En effet, puisqu'elle apparie deux entités très différentes, son poids w_{21} est *a priori* petit. Mais comme x_{32} met en relation deux entités qui sont situées directement au dessous des deux entités engagées par x_{21}, le choix de x_{21} permettrait de maintenir une homogénéité, si ce n'est au niveau de la forme des objets, tout du moins au niveau de la structure des scènes. En conséquence, il faudrait augmenter le poids w_{21} pour qu'il reflète le nouvel intérêt, accru, de x_{21}.

Le système *KBG* [Bisson 1992] détermine *au préalable* ces poids w_{ij} et montre que l'appariement optimal dépend fortement d'eux. Il serait en conséquence souhaitable que la relaxation puisse prendre leurs changements en compte. Toutefois, le modèle d'appariement bipartite ne permet pas de modifier les poids des mises en correspondance au cours de l'optimisation.

b) L'appariement entre attributs

Nous sommes en présence du problème, complexe et sensible, de la détermination des poids w_{ij} des variables. Plutôt que de *résoudre* ce problème, nous nous proposons de le *contourner*. Il s'agit à partir d'une définition simple des poids en fonction des ressemblances entre des attributs des entités, de maintenir la structure qu'il y a entre les entités dans l'espace des solutions du problème d'appariement. Pour ceci, il faut modifier notre première relation $relax_1$. Nous définissons donc les poids simplement à l'aide de la fonction de similarité entre attributs *simA* (39) (cf. p. 141), avec :

$$w_{ij} = simA\Big(A_i, A_j, \big(A_i - A_j\big)\Big) \tag{54}$$

L'introduction de la structure dans la relaxation se fait en deux étapes. La première concerne les attributs valeur qu'il faut explicitement considérer comme éléments à apparier. En conséquence, on les introduit dans $relax_2$ comme noeuds du graphe bipartite, et des arcs sont définis entre attributs du même type. Considérons à nouveau l'exemple du monde des polygones (figure 19). Les attributs sont `couleur` et `nombre` de cotés, avec les valeurs `noir`, `blanc`, respectivement `3` et `4` du côté de la scène A, et `blanc`, respectivement `3` et `4` du côté de la scène B. La représentation de ces valeurs comme noeuds du graphe bipartite conduit à un élargissement du modèle défini par (47) - (53). On introduit alors 4 noeuds supplémentaires dans N (n_3 et n_4 pour les couleurs ; n_5 et n_6 pour les nombres de côtés), 3 nouveaux noeuds dans N' (n'_3 pour `blanc` ; n'_4 et n'_5 pour les nombres de côtés) ainsi que les variables et les contraintes qui en découlent. On obtient alors le modèle suivant :

---

[11] Il ne s'agit pas ici d'esquisser un algorithme pour résoudre le problème d'appariement bipartite.

maximise : $f = \sum_{i=1}^{7} \sum_{j=1}^{5} w_{ij} \cdot x_{ij}$ (55)

avec (48) - (53) et :

$$x_{33} + x_{43} \leq 1 \quad (56)$$

$$x_{54} + x_{55} \leq 1 \quad (57)$$

$$x_{64} + x_{65} \leq 1 \quad (58)$$

$$x_{54} + x_{64} \leq 1 \quad (59)$$

$$x_{55} + x_{65} \leq 1 \quad (60)$$

Cette définition de $relax_2$ est insuffisante. En effet, elle ne contient aucun lien, entre les différentes mises en correspondance, qui permettrait de prendre en compte la structure des deux scènes. Par exemple, le prédicat `sur` n'est plus du tout considéré. (Dans $relax_1$ il intervenait dans la définition des poids.) Le but de la troisième formulation de la relaxation, $relax_3$, est donc de réintroduire cette structure entre attributs. Pour ceci, nous définissons le *lien* y_{kl} entre deux attributs a_k et a_l appartenant au même littéral *L*. Plus précisément, si $pos(L, a_k)$ indique la position qu'occupe l'attribut a_k dans *L*, une variable de lien y_{kl} est introduite pour chaque couple d'attributs (a_k, a_l) qui appartiennent au même littéral ; à condition que a_k précède, de gauche à droite, a_l (61). Ainsi, un lien est orienté de gauche à droite. Appelons a_k *l'origine* et a_l la *destination* du lien y_{kl}.

$$\text{lien } y_{kl} = \{(a_k, a_l) | a_k \in L, a_l \in L, pos(L, a_k) \leq pos(L, a_l), a_k \text{ est objet}\} \quad (61)$$

Considérons l'exemple du littéral `(forme A1 blanc 3)` de la figure 19. Il introduit les deux liens (`A1`, `blanc`) et (`A1`, `3`), alors que le littéral `(sur A1 A2)` n'en présente qu'un seul : (`A1`, `A2`). L'ensemble des littéraux de la scène A compte en tout 8 liens. En procédant de la même manière avec la scène B, on dénombre 5 liens.

Nous introduisons ces liens comme nouveaux noeuds de chaque côté des graphes bipartites et créons des variables d'appariements x_{ij} entre eux. Nous supposons que deux liens peuvent s'apparier seulement si d'une part, leurs origines s'apparient et d'autre part, leurs destinations s'apparient. Une variable d'appariement entre deux liens est créée si ces deux conditions sont réunies. De cette manière, une partie de la structure des situations est explicitement intégrée au problème d'appariement. Dans l'exemple, il y a aura une variable d'appariement entre le lien (`A1`, `blanc`) et le lien (`B1`, `blanc`), le lien (`A2`, `noir`) et le lien (`B1`, `blanc`), ou encore entre le lien (`A2`, `A3`) et le lien (`B1`, `B2`). En tout et pour tout, 13 variables d'appariement seront introduites pour ces liens.

Une relaxation définie de cette sorte rends les résultats de l'optimisation moins sensibles aux poids w_{ij}, dans la mesure où la structure des descriptions se présente, sous forme des variables d'appariement entre les liens, dans le problème d'optimisation lui-même.

c) L'extension du modèle d'appariement bipartite

A ce point de nos développements, la relaxation entre encore dans le cadre posé par le problème d'appariement bipartite. En effet, elle ne modifie ni la structure des variables et ni celle des contraintes de ce modèle, mais seulement la définition des noeuds, des arcs et des poids du problème d'appariement bipartite. En conséquence, la matrice des coefficients des contraintes garde sa propriété d'unimodularité, et la résolution du modèle reste de complexité polynomiale.

Néanmoins, cette modélisation comporte une lacune importante. L'interdépendance, entre les variables de mise en correspondance relatives aux liens et celles portant sur les attributs eux-mêmes, échappe au modèle. Plus précisément, mettre en correspondance deux liens entre attributs n'a de sens que lorsque les attributs adjacents aux liens sont eux aussi mis en correspondance. Soient $y_{kl} = (a_k, a_l)$ et $y'_{rs} = (a'_r, a'_s)$ deux liens, nous introduisons les variables d'appariement x_{klrs}, x_{kr} et x_{ls} afin de désigner les mises en correspondance y_{kl} - y'_{rs}, a_k - a'_r et a_l - a'_s. Nous recherchons alors les deux implications $x_{klrs} \rightarrow x_{kr}$ et $x_{klrs} \rightarrow x_{ls}$. Ainsi, si les deux liens y_{kl} et y'_{rs} sont appariés entre eux ($x_{klrs} = 1$), l'appariement entre a_k et a'_r et celui entre a_l - a'_s sont eux aussi fixés ($x_{kr} = 1$ et $x_{ls} = 1$).

En programmation mathématique la relation *d'implication* entre deux variables pseudo-booléennes “$x \rightarrow y$” est formalisée par la contrainte $x \leq y$, ou, en forme canonique, $x - y \leq 0$ [Williams 1993a]. Ainsi, dans l'hypothèse où $x = 0$, la variable y est libre, et si $x = 1$, elle est fixée à $y = 1$. Cette relation d'implication, aussi simple soit elle, n'est pour autant pas présente dans le modèle d'appariement bipartite. Les deux implications introduites ci-dessus s'écrivent donc par $x_{klrs} \leq x_{kr}$ et $x_{klrs} \leq x_{ls}$. Leur introduction dans le modèle d'appariement bipartite ne nécessite pas l'ajout de nouvelles variables.

Cela ne s'avère pas aussi simple pour ce qui et de la *résolution* de ce nouveau problème d'optimisation. En effet, la matrice de contraintes du nouveau problème perd sa propriété d'unimodularité. En conséquence, les techniques efficaces de la programmation linéaire ne parviennent plus à construire des solutions entières, c'est à dire des solutions où l'ensemble des variables x_{ij} prend des valeurs dans l'ensemble $\{0, 1\}$ (cf. p. 23). Par contre, le domaine de valeurs de x_{ij} devient continu, c'est à dire l'intervalle $[0\ldots1]$. Rendre à nouveau les variables x_{ij} discrètes nécessite le recours à des techniques de la programmation linéaire-mixte, telles que les procédures de séparation et d'évaluation (branch & bound). Cependant, ces techniques sont souvent d'une complexité d'exécution importante. Nous ne pouvons tout simplement pas, dans un système tel que *Alois*, *sérieusement* envisager d'utiliser une technique de ce type.

5.4.3. La résolution heuristique de la relaxation

Puisque les méthodes d'optimisation exactes classiques s'avèrent incapables de résoudre *efficacement* le problème d'appariement bipartite tel que modifié ci-dessus, il convient de recourir à une technique d'optimisation heuristique. Nous l'appellerons *RegretMatch*. Lors de l'analyse de *GloutonSimilarité* nous avons pu constater que celle-ci se prononce trop tôt sur le choix des appariements. Mis à part ce défaut, elle est efficace et, en outre, intègre le calcul de la taille de la partie commune aux deux situations. La nouvelle heuristique *RegretMatch* peut donc réutiliser *GloutonSimilarité* pour le calcul de similarité, elle se doit seulement de proposer des appariements de meilleure qualité. En effet, *RegretMatch* doit simplement déterminer l'appariement entre les attributs, σ_A. Nous ajoutons pour cela à la définition de *GloutonSimilarité* (tableau 22, cf. p. 150) la nouvelle ligne 1b appelant *RegretMatch* :

```
1.b    $\sigma_A = RegretMatch$(Source, Cible)
```

Pour des raisons d'efficacité globale du système, nous cherchons une réalisation de *RegretMatch* qui soit *rapide*. Nous écartons donc a priori toute technique remettant en cause les mises en correspondance qu'elle viendrait de construire, et optons pour une heuristique gloutonne sans retour en arrière.

Afin de résoudre le problème d'appariement il faut de mettre en correspondance des couples de noeuds du graphe bipartite G de manière à maximiser la fonction objectif (55). Appelons L l'ensemble des noeuds de G qui ne sont pas encore appariés. *RegretMatch* (tableau 25) a pour principe de sélectionner, de manière itérative, un noeud de L, de le mettre en correspondance avec un autre noeud de L et ensuite d'enlever les deux noeuds de L. Le problème concerne alors le choix de ces deux noeuds.

Nous employons, pour la sélection du couple de noeuds à apparier à chaque itération, une combinaison de deux stratégies classiques. Appelons *degré*(i) le nombre de variables d'appariement x_{ij} (ou x_{ji}) qui conduisent du noeud i de G vers un noeud j de G. Dans un premier temps, nous recourons à la stratégie du *premier échec* ("*first fail principle*"), souvent utilisée dans le domaine de la programmation par contraintes [Van Hentenryck 1989]. Elle préconise le choix des variables dont le domaine de valeur est le plus petit. Dans notre heuristique, ceci revient à préférer les noeuds de degré minimal (tableau 25, lignes 5 et 6).

Il se peut cependant qu'il y ait plusieurs noeuds de degré minimal, et qu'en outre, ces noeuds aient plusieurs alternatives d'appariement. Une deuxième stratégie, celle du *regret*, permet alors de choisir parmi l'ensemble des mises en correspondance qui font intervenir un noeud de degré minimal (tableau 25, lignes 7 à 9). La stratégie du regret est d'usage classique en recherche opérationnelle. Elle préconise la sélection, parmi un ensemble d'alternatives, de celle qui réduit *au moins* le nombre d'alternatives restantes. Compte tenu de la portée restreinte de *RegretMatch* dans nos travaux, l'exemple qui l'illustre se trouve en annexe (cf.

p. 221). Il concerne les deux situations du problème de la coloration de graphes précédemment présentées (tableau 24, cf. p. 154).

```
fonction RegretMatch(Source, Cible) : $\sigma_A$
G(N, N', A) = graphe bipartite créé à partir de Source et Cible
L = liste de tous les noeuds n de N et de N'
$\sigma_A$ = {}
tant que (L non vide) fait
    trie L croissant selon le degré(n) de ses membres
    $D_{min}$ = liste des noeuds de L avec degré minimal
    pour chaque ((n, n') ∈ A avec n ∈ N et n' ∈ N' et (n ∈ $D_{min}$ ou n' ∈ $D_{min}$))
        calcule regret(n, n')
        Meilleur = (n, n') avec regret minimal
    fin pour chaque
    $\sigma_A = \sigma_A$ + Meilleur
    enlève les noeuds de Meilleur de L
fin tant que
renvoi $\sigma_A$
```

Tableau 25. La fonction *RegretMatch*

Chapitre 6.

Evaluation expérimentale

Ce chapitre a pour objet l'évaluation de *Alois*. En l'absence de méthodes formelles de validation des systèmes de raisonnement à partir de cas [O'Leary 1993], nous avons eu recours à une évaluation expérimentale. Son but est double. D'une part, il s'agit de vérifier la cohérence interne de *Alois* en examinant le comportement de ses composants. D'autre part, il faut valider, sur plusieurs domaines d'application l'approche de l'intensification que suit *Alois*.

Pour l'évaluation de notre système, nous avons choisi trois problèmes d'optimisation, classiques en recherche opérationnelle. Ils sont NP-difficiles et à l'origine de toute une famille de problèmes réels. De plus, pour chacun d'entre eux existent des applications réussies de la méthode tabou. Le premier domaine d'expérimentation est celui des problèmes de placement et d'allocation d'entités à charge fixe. Nous appliquerons le système à des instances de problèmes benchmark pour lesquelles les solutions optimales sont connues. Le second domaine choisi, celui de l'ordonnancement simple de tâches sur une machine, est un problème équivalent à celui du voyageur de commerce, problème incontournable en optimisation combinatoire. Le troisième problème est celui de la coloration de graphes. Nous avons ici réimplanté l'algorithme de [Hertz & De Werra 1987], qui est en grande partie à l'origine de la réputation de la méthode tabou.

L'implantation de *Alois*, décrite dans cette thèse, comporte un simple interface utilisateur graphique, utile lors de la modélisation d'un nouveau problème d'optimisation. Le langage de programmation principalement utilisé est le C++. Le système est opérationnel sous Unix, ainsi que sous Windows, bien que l'ensemble des expérimentations présentées ont été conduites sur une station de travail Sun Sparc 10 avec Sun OS 4.1.3. Il convient dès lors de constater que le caractère prototypique de l'implantation de *Alois* ne permet pas à celui-ci de rivaliser avec les meilleures heuristiques d'optimisation, souvent dédiées à un seul problème. Le langage de représentation des problèmes d'optimisation, basé sur la logique d'ordre 1, en est pour une bonne partie responsable. L'incorporation de ce langage nous a, cependant, permis d'employer le même système dans domaines fort différents.

Notons dès à présent que nous n'emploierons pas la méthode de *l'évaluation croisée*[1] pour établir la performance de *Alois*, ce qui ne manquera peut-être pas d'étonner le lecteur rompu à l'apprentissage symbolique automatique. La raison en est cependant que *Alois* recommence chaque optimisation avec une mémoire de cas vide. Il n'y a donc pas de transfert de connaissances d'une instance de problème à l'autre. Une partition des instances de problème en un ensemble test et un ensemble d'apprentissage n'aurait pas de sens.

6.1. Le placement et l'allocation d'entités

Le problème *placement et d'allocation d'entités* (“*facility location problem*”) concerne le placement optimal d'entités sur un ensemble discret et fini de localités de choix. Le problème est très général. Il englobe, par exemple, en logistique les décisions concernant la localisation d'usines, d'entrepôts ou de magasins de ventes [Aikens 1985], en télécommunication le placement de concentrateurs [Aykin 1995], mais il peut tout autant concerner l'installation de services publics tels que des écoles, des bibliothèques ou des stations de sapeurs-pompiers. Selon la nature du problème, différentes modélisations ont été proposées. Beaucoup de ces modèles peuvent être résolus avec les ordinateurs dont on dispose aujourd'hui [Mirchandani & Francis 1990, Daskin 1995]. Pour nos besoins d'évaluation expérimentale, nous avons choisi de travailler sur le problème de placement et d'allocation d'entités à charge fixe, choix que nous justifions ci-dessous, après avoir introduit la formulation précise du problème.

6.1.1. Formalisation mathématique

Le *problème de placement et d'allocation d'entités à charge fixe* est décrit par un ensemble fini d'emplacements et par un ensemble fini de clients dont chacun a une demande. Il tend alors à la sélection optimale d'un nombre inconnu de ces emplacements, sur lesquels, à raison d'une par emplacement, les entités sont installées. Imaginons que ces entités soient des entrepôts. A chaque site sélectionné, c'est à dire à chaque entrepôt ouvert, des clients pourront être associés. Cet entrepôt desservira alors toute ou une partie de la demande des clients qui lui seront rattachés.

---

[1] A savoir la partition des instances de problème en *X* ensembles distincts, et l'évaluation des performances sur l'un de ces ensembles, l'ensemble test, *après* apprentissage à partir des X–1 ensembles restants, puis choix d'un autre ensemble test.

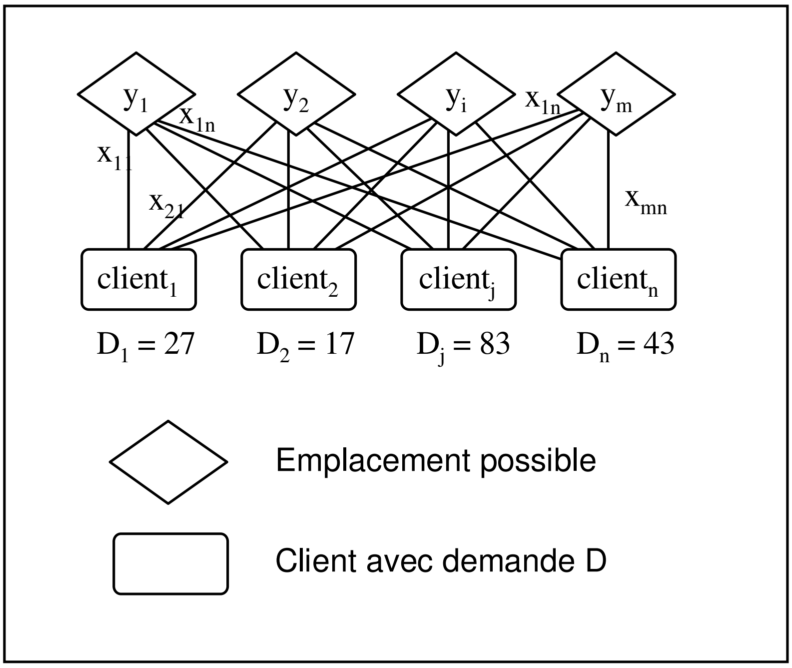


Figure 25. Problème de placement et d'allocation à *m* emplacements et *n* clients

Deux types de coûts sont associés à ces opérations. D'une part, une charge fixe qui est inhérente à toute installation d'un entrepôt sur un site. Cette charge peut varier d'un site à l'autre, mais reste indépendante du nombre de clients associés à l'entrepôt, ou du flux le traversant. D'autre part, des coûts variables unitaires qui sont liés au flux de marchandises entre un site et un client. Ces coûts varient selon la distance parcourue, mais ne dépendent pas de la quantité de marchandises traversant la liaison. Le but de l'optimisation de ces problèmes est d'identifier la configuration du réseau qui, pour un coût minimal, permet de satisfaire les demandes des clients. Le problème composé de *m* emplacements et de *n* clients (figure 25) peut être mathématiquement formulé ainsi :

$$\text{minimise : } f = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} F_i y_i \qquad (62)$$

avec :

$$\sum_{i=1}^{m} x_{ij} = D_j \qquad j = 1, \ldots n \qquad (63)$$

$$\sum_{j=1}^{n} x_{ij} \leq Q_{\max}^{i} \cdot y_i \qquad i = 1, \ldots m \qquad (64)$$

$$x_{ij} \in \Re^{+} \qquad i = 1, \ldots m\,;\, j = 1, \ldots n \qquad (65)$$

$$y_i \in (0,1) \qquad i = 1, \ldots m \qquad (66)$$

Ce modèle linéaire-mixte possède les variables x_{ij} et y_i. Chaque variable réelle x_{ij} (65) représente la quantité de marchandises passant entre le site *i* et le client *j*. Les variables de décision y_i (66) désignent, pour $y_i = 1$, la sélection de l'emplacement i pour l'ouverture d'un entrepôt. Dans la *fonction objectif* $f(\cdot)$, les paramètres c_{ij} désignent les coûts unitaires d'acheminement d'une unité de marchandise entre le site i et le client j. Les constantes F_j représentent les charges fixes relatives à l'installation d'un entrepôt sur le site i. Les contraintes (63) assurent que chaque client j voit sa demande D_j satisfaite. Les contraintes (64) garantissent que seuls les entrepôts ouverts peuvent desservir des clients. Ainsi avec $y_i = 0$ on a

$$\forall j, x_{ij} = 0 \tag{67}$$

et le flux de ce site vers tout client est alors automatiquement 0. Si l'entrepôt est ouvert, $y_i = 1$, le domaine des valeurs de x_{ij} est de $[0 \ldots Q^i_{\max}]$, avec $Q^i_{\max}$ la limite supérieure de capacité du site i. Si maintenant on a

$$Q^i_{\max} \geq \sum_{j=1}^{n} D_j \qquad i = 1, \ldots m \tag{68}$$

les limites de capacité sur les x_{ij} sont toujours respectées et ne vont pas contraindre la solution. De plus, (68) implique l'absence complète de limite supérieure de la capacité des entrepôts. Le problème comportant cette contrainte est la variante la plus simple des problèmes de placement et d'allocation à charge fixe. Il est connu sous le nom de *problème de placement et d'allocation sans restrictions de capacité* (*Uncapacitated Facility Location Problem*, *UFLP*).

Par contre, si (68) n'est pas retenue, il faut néanmoins que la condition

$$\sum_{i=1}^{m} Q^i_{\max} \geq \sum_{j=1}^{n} D_j \tag{69}$$

soit respectée pour que des solutions réalisables puissent exister. Le problème intégrant cette contrainte est connu sous le nom de *problème de placement et d'allocation avec restrictions de capacité* (*Capacitated Facility Location Problem*, *CFLP*).

6.1.2. Résolution et applications

Malgré la simplicité de leur formulation, ces deux modèles UFLP et CFLP ont des applications industrielles directes [Brandeau & Chiu 1989, Monticone & Funk 1994]. Ils se trouvent notamment intégrés dans des systèmes plus complexes tels que ceux utilisés pour l'aide à la décision stratégique [Guedes & al. 1995]. Par ailleurs, ces deux modèles sont à la base de toute une famille de problèmes de placement et d'allocation plus réalistes, et donc plus complexes. Ces modèles étendus intègrent, plusieurs types de marchandises [Klincewicz & al. 1986] ; des charges “fixes” qui varient à seuils discrets selon le flux qui passe à travers l'entité [Hansen & al. 1992] ; des configurations de réseaux plus complexes avec des flux auxiliaires qui passent d'une entité à l'autre, et non pas seulement des entités vers les clients [Crainic & al. 1993a] ; des réseaux avec des points d'arrêts obligatoires au cours des acheminements de marchandises [Aykin 1995] ; ou encore des clients dont la demande varie selon la proximité de l'entité qui les dessert [Hakimi & Kuo 1991].

Les problèmes UFLP et CFLP sont NP-difficiles [Mirchandani & Francis 1990]. Pourtant, leur résolution est considérée comme étant facile [Mirchandani & Francis 1990]. En effet, il existe des heuristiques très efficaces qui construisent des solutions de très bonne qualité en exploitant des propriétés mathématiques qui sont propres à ces deux problèmes [Erlenkotter 1978, Beasley 1993, Labbé & al. 1994]. En ce qui concerne la résolution de problèmes plus réalistes, des heuristiques, basées sur la recherche locale, sont souvent utilisées. Ainsi, le recuit simulé [Magnati & Wong 1994] ou la méthode tabou [Hansen & al. 1992, Crainic & al. 1993a] sont-elles employées pour la résolution des problèmes esquissés.

Nous avons choisi d'évaluer notre système sur les deux problèmes UFLP et CFLP pour deux raisons complémentaires. D'une part, il existe des instances de problèmes benchmark, pour lesquelles les solutions optimales sont connues et répertoriées dans la OR-library [Beasley 1990]. Ceci permet de comparer les performances de la méthode heuristique de *Alois* avec la solution optimale. On obtient ainsi la marge d'erreur absolue produite par l'heuristique. D'autre part, et bien que la résolution de ces deux problèmes ne nécessite pas le recours à la méthode tabou, des approches basées sur celle-ci se sont avérées efficaces sur des variations plus complexes de ce type de problèmes [Hansen & al. 1992, Crainic & al. 1993a].

6.1.3. La méthode tabou dans *Alois*

La modélisation d'un problème d'optimisation, pour une résolution avec la méthode tabou, nécessite la définition d'une part de l'espace de recherche, composé d'états et de transitions, et d'autre part des structures de contrôle propres à la méthode tabou (cf. p. 35). Pour ces deux problèmes d'optimisation, nous fondons notre méthode tabou sur une heuristique simple du type “drop”. L'espace de recherche est composé de l'ensemble des configurations du réseau, et le passage d'une de ces configurations à l'autre se fait par la fermeture d'une entité. Ainsi, à chaque itération une entité est fermée et ses clients sont repartis sur d'autres entités, qui elles soit sont déjà ouvertes, soit sont alors ouvertes.

La figure 26 résume l'emploi des opérateurs. L'opérateur `closeWarehouse` s'applique à un état correspondant à une configuration réalisable du problème. Il ferme un entrepôt. Ensuite, pour chaque client affecté à cet entrepôt, l'opérateur `separateOneClient` libère la demande du client. Celle-ci est alors réallouer à un autre entrepôt, même si ce dernier doit être ouvert, par l'opérateur `affectOneClient`. La solution initiale est construite en affectant chaque client à l'entrepôt qui lui est le plus proche. La recherche se termine si après un nombre maximal de 30 itérations la qualité de la solution, c'est à dire la meilleure valeur de la fonction objectif, n'est pas améliorée.

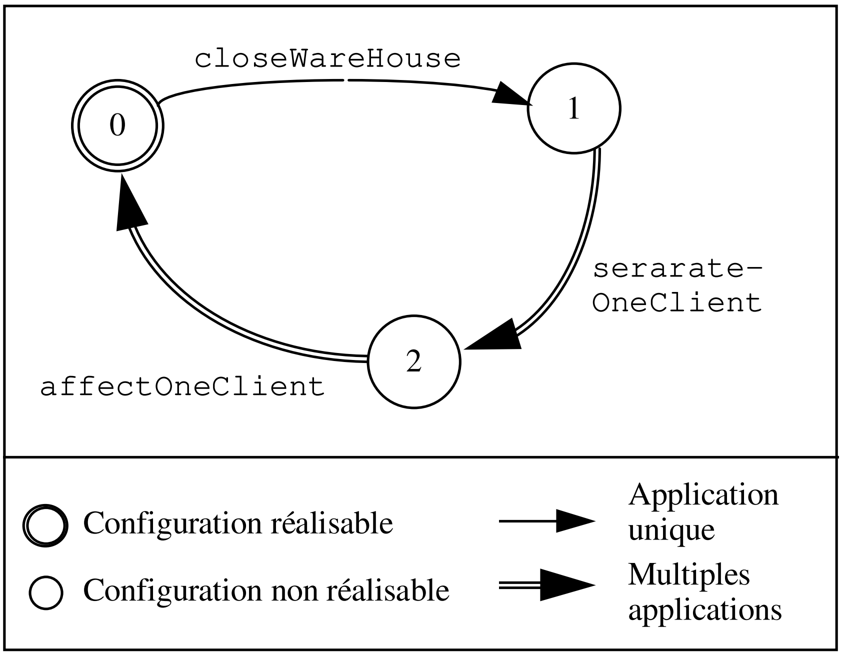


Figure 26. Schémas d'opérateur et structure de contrôle pour les problèmes de placement et d'allocation

L'heuristique de recherche locale est, pour l'essentiel, la même pour UFLP et pour CFLP. La seule différence se situe au niveau de l'opérateur `affectOneClient` qui doit, pour CFLP, s'assurer que la capacité maximale de l'entrepôt n'est pas dépassée lors de l'allocation d'un nouveau client. S'il y a dépassement de la capacité, seule la partie de la demande du client à laquelle l'entrepôt peut faire face est allouée. L'excédent doit alors être affecté à un autre entrepôt.

Numéro	Problème	Conditions Tabou	Forme de la liste
1	UFLP	Ouverture	Fixe
2	UFLP	Ouverture	Variable
3	UFLP	Fermeture	Fixe
4	UFLP	Fermeture	Variable
5	CFLP	Ouverture	Fixe
6	CFLP	Ouverture	Variable
7	CFLP	Fermeture	Fixe
8	CFLP	Fermeture	Variable

Tableau 26. Les 8 différentes implantations de la méthode tabou pour UFLP et CFLP

A cette procédure de recherche locale, nous avons ajouté une méthode tabou simple, la même pour UFLP et CFLP, composée d'une seule liste tabou. Quatre implantations différentes ont été testées sur chacun des deux problèmes (tableau 26). Ainsi, deux différentes conditions tabou sont utilisées, inscrivant les entrepôts qui ont été fermés par `closeWarehouse` dans la liste tabou. La première (*ouverture*) interdit la réouverture de tout entrepôt inscrit dans la liste tabou, tandis que la deuxième (*fermeture*) n'admet pas la refermeture d'un tel entrepôt (mais seulement sa réouverture). De même, deux différentes implantations de la liste tabou sont évaluées. La première (*fixe*) comprend une liste tabou de taille fixe de 7, longueur qui dans une première approche devrait mener à de bons résultats (cf. p. 38). La deuxième (*variable*) emploie une liste de taille variable, évoluant dans [6…10], avec une probabilité d'augmentation et de diminution de la longueur de la liste qui est de 20% à chaque itération.

Les méthodes tabou que nous venons de présenter sont des plus simples dans leur conception. Mais notre seul intérêt pour elles réside en ce qu'elles nous servent de support pour l'évaluation de notre technique d'apprentissage. De façon préméditée, nous avons donc choisi de ne pas faire d'effort pour leur perfectionnement.

Nous avons appliqué ces 8 méthodes tabou à 12 instances benchmark (tableau 27) de la OR-library [Beasley 1990]. Ces instances sont les mêmes pour UFLP et CFLP. Toutefois, pour CFLP, une limite de capacité pour chaque entrepôt est prise en compte. Notons que ces instances sont de taille moyenne, elles comprennent au maximum 50 emplacements et 50 clients. Les techniques citées ci-dessus parviennent à résoudre des instances de taille 1000. Le caractère prototypique de l'implantation de *Alois*, ne nous permet pas de traiter d'instances aussi grandes.

Nom de l'instance	nombre d'emplacements	nombre de clients
cap71	16	50
cap72	16	50
cap73	16	50
cap74	16	50
cap101	25	50
cap102	25	50
cap103	25	50
cap104	25	50
cap131	50	50
cap132	50	50
cap133	50	50
cap134	50	50

Tableau 27. Les instances de problème de CFLP et UFLP

6.1.4. La représentation du domaine

Pour introduire le langage de modélisation des problèmes d'optimisation de *Alois*, nous nous sommes servis du problème de placement et d'allocation (chapitre 3, cf. p. 93). Nous ne reprenons pas ici sa modélisation. Nous appliquons notre technique de raisonnement à partir de cas à l'opérateur `closeWarehouse`. Sa définition est présentée dans la figure 11 (cf. p. 98), accompagnée d'un exemple de son contexte d'instanciation. La modélisation complète du domaine se trouve en annexe (cf. p. 215).

6.1.5. Les expérimentations

Les expériences conduites sur le problème de placement et d'allocation d'entités ont pour objet d'évaluer plusieurs points. Il s'agira tout d'abord d'établir l'impact de l'intégration de notre technique de raisonnement à partir de cas dans les quatre méthodes tabou décrites ci-dessus. Ensuite, nous évaluerons l'organisation de la mémoire de cas, sous forme d'arbre de prototypes dynamique, au regard des trois éléments clés que sont la qualité des solutions engendrées, l'efficacité lorsqu'il s'agit de manipuler de larges ensembles de cas et la précision de la récupération.

a) La qualité de la solution

Le but de ce paragraphe est d'évaluer les incidences de notre technique de raisonnement à partir de cas sur la qualité des solutions obtenues sur les deux problèmes UFLP et CFLP. Nous avons ainsi appliqué les quatre méthodes tabou, pour chaque problème, sur les 12 instances de problème cap71 - cap134. Pour chacune de ces configurations, nous avons d'abord appliqué la méthode tabou sans apprentissage, puis nous y avons ajouté le raisonnement à partir de cas. Nous avons en tout et pour tout effectué 192 optimisations différentes. Aucune n'a demandée plus de 150 itérations.

Les tableaux 28.a) pour UFLP, et 28.b) pour CFLP, présentent un sommaire des meilleures solutions obtenues, par méthode, lors de ces expériences. Les valeurs de la fonction objectif ont été normalisées de façon à ce que, pour chaque instance, la valeur de la solution optimale s^* connue soit $f(s^*) = 1$. Les deux colonnes '*Meilleure solution*' de chaque tableau présentent, pour chaque méthode tabou, la moyenne des valeurs de la fonction objectif de la meilleure solution trouvée pour chacune des 12 instances, à gauche sans apprentissage, et à droite en ajoutant l'apprentissage. Plus les valeurs de ces colonnes sont petites, meilleur est le résultat. La colonne tout à fait à droite indique le pourcentage d'amélioration de la fonction objectif résultant de l'intégration du raisonnement à partir de cas, par rapport à $f(s^*)$. La ligne '*Moyenne*' donne les moyennes sur les quatre méthodes tabou, tandis que la ligne '*Comparatif*' indique combien de fois une technique a produit une meilleure solution que l'autre.

On remarque tout d'abord que les solutions produites par toutes ces méthodes tabou ne sont pas de bonne qualité, l'écart avec les solutions optimales étant de 20% à 30% en moyenne. Ceci n'est pas vraiment étonnant dans la mesure où aucun effort n'avait été fait pour adapter ces méthodes aux problèmes traités. En revanche, l'ajout du raisonnement à partir de cas s'avère bénéfique. En effet, *Alois* produit les meilleures solutions dans 88 des 96 instances de problèmes, et réduit, en moyenne, l'écart avec la solution optimale de 33% (UFLP) à 41% (CFLP). De même, *Alois* améliore les performances des quatre méthodes tabou, et ceci pour les deux problèmes d'optimisation.

UFLP			
Méthode tabou	Meilleure solution		Amélioration
	Sans Appr.	Alois	
1-ouv./fixe	1.27	1.16	40.7%
2-ouv./var.	1.28	1.14	50.0%
3-ferm./fixe	1.34	1.27	20.6%
4-ferm./var	1.30	1.23	23.3%
Moyenne	1.30	1.20	33.7%
Comparatif	5/48	43/48	

a)

CFLP			
Méthode tabou	Meilleure solution		Amélioration
	Sans Appr.	Alois	
5-ouv./fixe	1.26	1.15	42.3%
6-ouv./var.	1.27	1.14	48.1%
7-ferm./fixe	1.34	1.22	35.3%
8-ferm./var	1.33	1.20	39.4%
Moyenne	1.30	1.18	41.3%
Comparatif	3/48	45/48	

b)

Tableau 28. Moyenne sur 12 instances des meilleures solutions obtenues par les diverses méthodes tabou

L'utilisation de la technique d'apprentissage a également des conséquences sur le temps d'exécution des méthodes tabou. Nous en traiterons ci-dessous lors de l'évaluation de l'organisation de la mémoire de cas.

b) L'organisation de la mémoire des cas

Une mémoire de cas doit satisfaire à deux buts. D'une part, elle doit fournir, lors de la récupération, le cas qui contient la situation la plus proche de l'opérateur applicable considéré. D'autre part, cette récupération doit être le plus efficace possible. Compte tenu de ce double objectif, nous avons introduit une organisation de la mémoire de cas basée sur les arbres de prototypes dynamiques (chapitre 4, cf. p. 121). En diminuant la précision de la récupération, ce type de mémoire devrait s'avérer bien plus efficace qu'une mémoire composée d'une liste séquentielle de cas, et qui constitue l'autre alternative.

L'évaluation de la mémoire de cas doit, en conséquence, porter sur trois éléments. En premier lieu, il s'agit d'analyser l'impact de l'organisation de la mémoire sur la qualité des solutions engendrées. Ensuite, il convient de considérer l'efficacité des opérations de la mémoire, et en dernier lieu, il faut établir la précision de la récupération. Pour ceci, nous allons comparer, entre elles, trois différentes organisations de la mémoire : une mémoire à liste plate, un arbre de prototypes binaire, et un arbre de prototypes dynamique.

Les deux tableaux 29 et 30 sont relatifs à l'influence de ces trois organisations de la mémoire de cas sur la qualité des solutions engendrées. Les expériences sont conduites sur 8 instances de problèmes de CFLP avec deux méthodes tabou, l'une avec une liste tabou fixe, l'autre avec une liste variable, et des conditions tabou du type fermeture. Les colonnes '*f(Best)*' indiquent, pour chaque instance et méthode, les valeurs de la fonction objectif de la meilleure solution trouvée, tandis que les colonnes '*Amélior.*' présentent l'amélioration de la fonction objectif, due à *Alois*, par rapport à la valeur de la solution optimale. A gauche sont présentés les résultats de la méthode tabou sans apprentissage, suivi de ceux de *Alois*, avec les trois types d'organisation de la mémoire qui sont évaluées.

Instance	Tabou	Alois + liste plate			Alois + arbre de prot.			Alois + arb. prot. dyn.		
	f(Best)	f(Best)	Amélior.	nb sim	f(Best)	Amélior.	nb sim	f(Best)	Amélior.	nb sim
cap71	1.08	1.03	63.4%	27800	1.07	13.8%	5731	1.05	37.1%	9339
cap72	1.10	1.06	41.2%	23756	1.06	42.9%	5700	1.06	41.3%	9558
cap73	1.10	1.05	48.5%	25412	1.07	31.5%	6029	1.04	63.7%	10458
cap74	1.14	1.10	29.0%	29080	1.08	39.6%	4828	1.09	34.8%	10519
cap101	1.41	1.08	81.5%	58425	1.05	88.4%	10811	1.07	84.0%	12645
cap102	1.44	1.06	86.9%	56984	1.28	35.9%	12207	1.08	82.3%	12377
cap103	1.51	1.17	66.0%	60763	1.32	36.8%	11039	1.32	36.8%	11582
cap104	1.57	1.24	58.3%	51521	1.41	28.1%	10319	1.21	62.6%	19224
Moyenne	1.29	1.10	66.8%	41718	1.17	43.0%	8333	1.11	61.2%	11963

Tableau 29. L'organisation de la mémoire de cas, évaluée sur CFLP avec la méthode tabou à liste fixe (fermeture)

Instance	Tabou	Alois + liste plate			Alois + arbre de prot.			Alois + arb. prot. dyn.		
	f(Best)	f(Best)	Amélior.	nb sim	f(Best)	Amélior.	nb sim	f(Best)	Amélior.	nb sim
cap71	1.06	1.02	67.9%	28747	1.05	23.0%	4819	1.04	33.3%	11588
cap72	1.05	1.04	15.5%	25367	1.05	4.8%	4220	1.02	55.0%	9357
cap73	1.07	1.05	31.5%	22749	1.04	49.9%	5016	1.05	35.8%	9356
cap74	1.10	1.09	8.8%	20023	1.09	14.9%	5743	1.09	9.9%	7495
cap101	1.41	1.02	95.1%	57252	1.06	84.4%	10532	1.06	84.4%	11967
cap102	1.39	1.10	74.5%	49702	1.06	85.2%	9622	1.07	81.1%	10933
cap103	1.46	1.16	65.3%	60391	1.24	48.3%	9686	1.24	49.0%	10378
cap104	1.54	1.24	55.7%	51468	1.31	42.9%	8562	1.13	75.2%	12930
Moyenne	1.26	1.09	65.4%	39462	1.11	57.5%	7275	1.09	66.0%	10501

Tableau 30. L'organisation de la mémoire de cas, évaluée sur CFLP avec la méthode tabou à liste variable (fermeture)

On peut constater que *Alois* améliore la qualité de la meilleure solution sur l'ensemble des optimisations conduites, et ceci quelle que soit celle des trois organisations de la mémoire employée. Toutefois, l'amélioration varie selon l'organisation choisie. Tout d'abord, la mémoire du type liste plate et la mémoire à base d'un arbre de prototypes dynamique présentent sensiblement les mêmes résultats. Ceci est a priori étonnant, puisque la récupération dans un arbre de prototypes dynamique est par définition moins précise que la récupération dans une mémoire liste plate. On peut en conclure qu'une mémoire basée sur un arbre de prototypes dynamique permet la récupération de cas qui sont aussi *utiles* que les cas récupérés dans une mémoire du type liste plate.

L'organisation de la mémoire basée sur l'arbre de prototypes classique, même si elle obtient des résultats à peu près équivalents à ceux des deux autres types d'organisation sur la méthode tabou à liste variable, se distingue en revanche de ces dernières lors de l'application dans la méthode tabou à liste fixe (tableau 29). En effet, dans cette dernière configuration, l'écart entre l'optimum et les solutions trouvées est en moyenne de 17%, tandis que pour les deux autres approches, celui-ci est respectivement de 10% et 11%. Nous allons reconsidérer ce point, lorsque nous mesurerons directement la précision de la récupération dans ces différentes mémoires.

c) Le temps d'exécution

La différence, au niveau du temps d'exécution, qui peut exister entre les méthodes utilisant différentes organisations de la mémoire de cas est essentiellement due au nombre de fois où la similarité entre deux contextes d'instanciation doit être calculée. Les deux tableaux 29 et 30 indiquent le nombre de calculs de la similarité, par instance et par méthode tabou, dans les colonnes '*nb sim*'. Notons que nous utilisons la fonction *Glouton-Similarité* pour le calcul de la similarité dans les problèmes UFLP et CFLP (chapitre 5, cf. p. 149). La figure 27 présente le temps de calcul, par itération, de chacune des quatre méthodes considérées.

Tout d'abord, la figure 27 confirme le ralentissement linéaire, proportionnel au nombre de cas en mémoire, auquel on pouvait s'attendre de la part de la mémoire plate. Les deux arbres de prototypes se révèlent bien plus adaptés aux mémoires de cas larges. Dans les deux tableaux 29 et 30, on remarque que le nombre de calculs de la similarité diminue d'un facteur 5 lors du passage d'une liste plate à un arbre de prototypes classique. Ceci est dû exclusivement à l'organisation structurée des cas. L'arbre de prototypes dynamique requiert légèrement plus de calcul de la similarité que ne le fait l'arbre classique, mais toujours beaucoup moins que la liste plate. La raison en est qu'en début de recherche, les premiers noeuds de l'arbre montrent, en raison du degré élevé qu'ils peuvent avoir, un comportement similaire à une liste plate. La figure 27 conforte cette analyse. On remarque que le temps d'exécution par itération de l'approche utilisant l'arbre de prototypes dynamique augmente, lors des quelques premières itérations, parallèlement à celui de la liste plate, alors que celui de l'approche utilisant l'arbre de prototypes classique s'aplatit plus rapidement.

Si l'effet de l'organisation de la mémoire de cas en arbre n'est pas surprenant, il convient quand même de souligner que l'accélération constatée ne se fait pas, au moins en ce qui concerne les arbres de prototypes dynamiques, au détriment de la qualité des solutions engendrées par *Alois*.

En comparant les temps d'exécution de la méthode tabou sans apprentissage avec ceux de la méthode tabou avec le raisonnement à partir de cas, il est apparent que la première approche est environ 2 fois plus rapide que la seconde. Le surcoût de l'apprentissage est donc non négligeable. Il faut, en conséquence, se poser la question de savoir si l'amélioration de la qualité des solutions vaut cette augmentation du temps d'exécution. Nous reviendrons ci-dessous sur cette question lors de l'analyse de *Alois* sur le problème de la coloration de graphes (cf. p. 189).

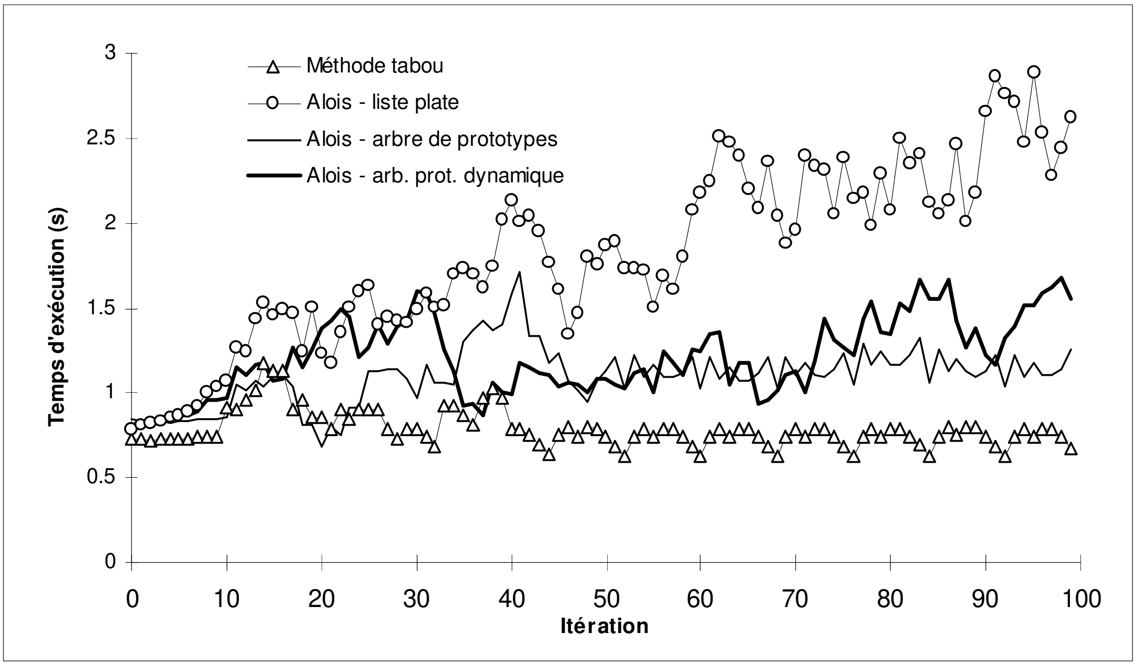


Figure 27. Temps d'exécution par itération sur cap101, méthodes tabou avec liste fixe (fermeture)[2]

---

[2] Notons que chaque point du graphique correspond à la moyenne des cinq points qui le précèdent, ce qui revient à appliquer un filtre aux courbes afin qu'elles deviennent plus lisses et que leur évolution soit plus apparente.

d) La précision de la récupération

Les résultats présentés dans le paragraphe b) confirment la qualité de la récupération de cas dans une mémoire de cas basée sur un arbre de prototypes dynamique. En effet, ces cas sont aussi utiles que ceux récupérés dans d'une mémoire à base de liste plate. Cette affirmation ne se fonde, cependant, que sur la qualité des solutions obtenues. Or, il est intéressant de mesurer directement la précision de la récupération dans ces diverses mémoires de cas.

Les cas n'appartenant pas à des classes, nous ne pouvons pas appliquer les méthodes d'évaluation classiques en catégorisation, qui ne portent que sur la précision de prédiction. Pour conduire notre évaluation, nous proposons donc une autre approche. Ainsi, nous avons donc modifié *Alois* de manière à ce que le système puisse travailler avec deux mémoires de cas différentes et indépendantes. Nous appelons l'une *maître*, et l'autre *esclave*. Les deux mémoires sont vides au début d'une optimisation. Les cas sont acquis dans ces deux mémoires, et la récupération y est effectuée. Cependant la mémoire maître guide l'exploration, tandis que le cas récupéré dans la mémoire esclave n'a pour objet que d'être comparé à celui récupéré dans la mémoire maître. Dans toutes les expérimentations décrites ici, la mémoire maître est toujours basée sur une liste plate, tandis que la mémoire esclave est soit basée sur un arbre de prototypes classique, soit sur un arbre de prototypes dynamique.

Rappelons que dans une mémoire plate, la récupération identifie toujours le cas contenant la situation `source` la plus proche de la situation `cible`, ce qui n'est pas nécessairement vrai dans une mémoire qui est structurée. C'est alors la variation de la similarité entre la situation récupérée `source` et la `cible` qui nous intéresse, et non pas la similarité entre les situations des cas récupérés dans les deux mémoires maître et esclave. Nous appelons $sim_M = similarit\acute{e}(\texttt{source}, \texttt{cible})$ la similarité entre la situation du cas récupéré dans la mémoire maître et la cible, et $sim_E = similarit\acute{e}(\texttt{source}, \texttt{cible})$ celle relative au cas récupéré dans la mémoire esclave. Définissons maintenant le *ratio de similarité r* par :

$$r = \frac{sim_E}{sim_M} \qquad (70)$$

Le ratio r est défini dans $[0\ldots1]$, puisque $sim_E \le sim_M$ quand la mémoire maître est une liste plate. Plus r est grand, plus les deux situations récupérées se ressemblent. Nous établissons un compteur pour chaque centième de l'intervalle de définition de r, donc un pour $r = 0.01$, un pour $r = 0.02$, … jusqu'au dernier compteur qui est relatif à $r = 1.00$.

Nous avons conduit les expérimentations sur les quatre instances de problèmes cap71 - cap74 de CFLP, en utilisant comme base la méthode tabou à liste fixe et la condition tabou sur la fermeture des entités. Chaque optimisation porte sur 500 itérations avec une sélection aléatoire des opérateurs. Chacune des deux mémoires acquiert donc, au fur et à mesure, 500 cas. Après chaque récupération, le ratio *r* est calculé, et le compteur correspondant à la valeur obtenue est incrémenté. A la fin, on fait la somme par compteur des quatre optimisations. On obtient alors une sorte de distribution de probabilité pour *r*.

Les résultats sont présentés dans la figure 28. L'abscisse porte sur l'intervalle de *r*, tandis que l'ordonnée indique la probabilité que *r* soit plus grand que la valeur correspondant à l'abscisse. La courbe en gras (A) est relative à la mémoire esclave basée sur un arbre de prototypes dynamique, tandis que la courbe fine porte sur une mémoire esclave basée sur un arbre de prototypes classique (B).

On remarque que les valeurs obtenues pour *r* sont bien plus grandes pour (A) qu'elles ne le sont pour (B). En effet, pour ce qui est de (A), la mémoire basée sur un arbre de prototypes dynamique récupère dans 76.9% des hypothèses un cas avec une situation qui est aussi proche de la `cible` que l'est celle récupérée dans la liste plate ($r = 1$). De même, *r* est plus grand que 0.8 dans 87% des cas. La précision de la mémoire de cas structurée nous parait donc assez élevée. En ce qui concerne (B), on a $r = 1$ seulement en 57.7% des cas, et *r* est plus grand que 0.8 seulement dans 72% des récupérations.

Nous tirons trois conclusions de ces résultats. Premièrement, la récupération dans une mémoire basée sur un arbre de prototypes classique est bien moins précise que celle effectuée dans une mémoire basée sur un arbre de prototypes dynamique. Ceci nous semble expliquer la différence existante entre ces deux mémoires quant à la qualité des solutions qu'obtient *Alois*. Deuxièmement, le problème des objets écrans dans un arbre de prototypes dynamique, que nous avons mentionné au chapitre 4 (cf. p. 124), ne semble pas se produire, ou tout au moins, ne se manifeste pas de manière à dégrader la précision de la récupération. La récupération dans un arbre de prototypes classique, où le phénomène des objets écrans est a priori absent, est moins précise.

Troisièmement, on peut constater que la précision de la récupération a des conséquences directes sur la qualité des solutions produites par *Alois*. Si on réutilise des cas qui sont plus proches de la cible, la qualité des solutions s'améliore. Ceci n'est a priori pas étonnant de la part d'un système de raisonnement à partir de cas qui est bien conçu, et, par là, confirme l'approche entreprise dans *Alois*. En effet, on constate l'existence d'un lien entre la ressemblance de deux situations dans lesquelles un opérateur s'applique, et la validité de la réutilisation de la récompense. Plus les deux situations se ressemblent, plus est il intéressant de réutiliser la récompense. Toutefois, on constate aussi que sur ce point la méthode de *Alois* peut être perfectionnée. Bien que la récupération dans une liste plate soit par définition plus précise que la récupération dans un arbre de prototypes dynamique, la qualité des solutions ne s'en trouve pas améliorée.

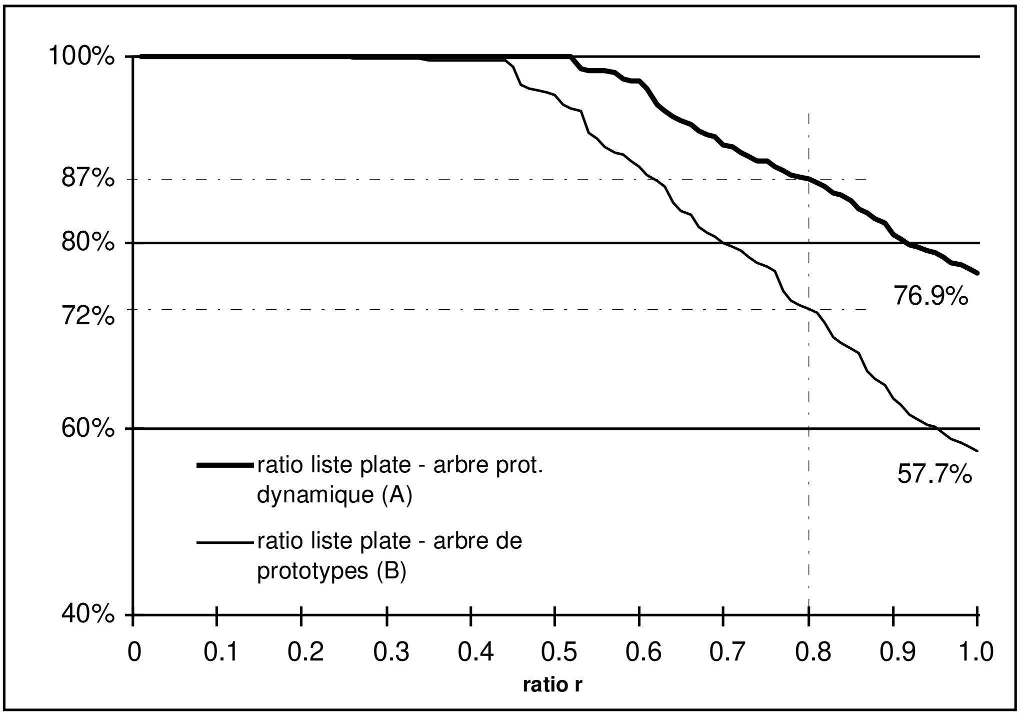


Figure 28. Evaluation de la précision de récupération

L'ensemble de ces résultats nous parait confirmer l'intérêt de l'utilisation d'arbres de prototypes dynamiques comme mémoires de cas. En effet, le gain obtenu au niveau de l'interaction avec une mémoire de cas de ce type est intéressant, surtout qu'il n'entraîne pas, en contrepartie, une diminution de la qualité des résultats. Tout au moins pour ce qui est de notre système, les cas récupérés dans une mémoire à base d'un arbre de prototypes dynamique sont aussi utiles que ceux récupérés dans une mémoire du type liste plate.

6.2. L'ordonnancement de production simple à une machine

L'ordonnancement de production est l'un des domaines d'application les plus importants de la recherche opérationnelle. Son impact économique en est pour une bonne partie responsable. Cependant, le caractère particulièrement dur de la résolution des problèmes d'ordonnancement de production n'y est pas non plus étranger. En effet, non seulement ils sont souvent NP-difficiles, mais en outre, pour la plupart d'entre eux, il n'existe aujourd'hui pas de technique, exacte ou heuristique, qui permette de résoudre des instances de grande taille[3]. Pour le problème simple d'ordonnancement à une machine, que nous considérons pour l'évaluation expérimentale de *Alois*, des solutions optimales exactes sont connues pour des instances allant jusqu'à 20 tâches [Barnes & Laguna 1993].

Un nombre important de travaux traite des différents aspects des problèmes d'ordonnancement. La plupart sont issus de la recherche opérationnelle. Ils étudient les différents aspects de ces problèmes et proposent un vaste éventail de techniques de résolution. Parmi les travaux récents, on peut citer [Kassou 1992, Chrétienne & al. 1995, Pinedo 1995, Blazewicz & al. 1996]. Les approches utilisant des techniques d'intelligence artificielle sont plus récentes. Elles se consacrent souvent à l'amélioration d'ordonnancements sous-optimaux ou à la gestion des aléas inhérents à l'exécution d'un ordonnancement [Miyashita & Sycara 1995]. Actuellement, les techniques issues de ces deux disciplines se rapprochent, et de leur combinaison découlent des résultats intéressants [Ghedjati - Guessoum 1994, Zweben & Fox 1994, Brown & Scherrer 1995]. Selon Barnes & Laguna (1993), l'application de la méthode tabou aux problèmes d'ordonnancement a été l'une de ses premières réussites. Depuis, des méthodes tabou ont été construites pour des problèmes d'ordonnancement variés [Barnes & al. 1995]. C'est pourquoi nous avons choisi d'évaluer *Alois* sur un problème de cette famille.

6.2.1. Description formelle du problème

Le *problème d'ordonnancement simple à une machine* est un problème simple de la famille des problèmes d'ordonnancement. Il s'agit de déterminer la séquence de n tâches, toutes indépendantes les unes des autres, sur une seule machine. Toutes les tâches sont disponibles dès le temps 0. Chaque tâche i a un temps de production p_i, et il existe un temps de changement d'outils c_{ij} lorsque les tâches i et j se suivent. Dans ce problème

---

[3] Ce qui est pourtant le cas pour les deux autres problèmes, NP-difficiles eux aussi, sur lesquels nous évaluons notre système.

l'optimisation peut avoir différents buts. La fonction objectif la plus apparente, et à laquelle nous nous tenons dans ce qui suit, consiste à minimiser la durée totale de la séquence des tâches, c'est à dire la durée d'occupation de la machine (figure 29). Un autre objectif peut être la minimisation de la durée d'attente de l'ensemble des tâches. Si les tâches ont des délais de finition, on peut aussi vouloir minimiser le retard total pour l'ensemble des tâches. Ce problème d'ordonnancement à *n!* solutions possibles. Il équivaut au problème du voyageur de commerce asymétrique. Malgré la simplicité de sa formulation, il est donc NP-difficile.

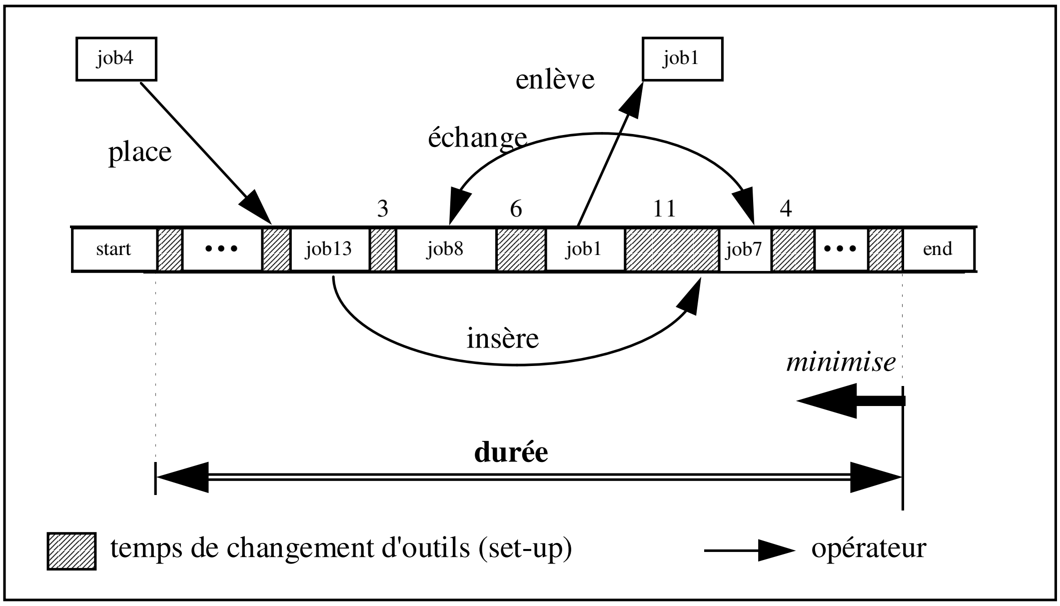


Figure 29. Le problème d'ordonnancement simple à une seule machine

6.2.2. La représentation du domaine

Le tableau 31 indique les attributs du domaine. Les tâches sont les attributs objet. Ils sont du type `Job`. Chaque tâche a un état d'ordonnancement, une position dans la séquence des tâches ordonnées, un temps de production et une durée de changement d'outils par rapport à la tâche qui la précède. Ceux sont les attributs valeur. Ils ont chacun leur type, `State`, `Position`, `p-Time` et `s-Time`.

La définition des prédicats est donnée au tableau 32. Le prédicat `state` décrit la tâche. Il indique son état (`free` ou `scheduled`) ainsi que son temps de production. Le prédicat `before` décrit la liaison entre deux tâches. Il contient les deux tâches ainsi que le temps de changement d'outils nécessaire pour passer de l'une à l'autre. Un littéral typique de ce domaine est `(before job5 job14 34)` qui indique que la tâche `job5` est le prédécesseur de la tâche `job14`, avec un temps de changement d'outils de

`34`. Le tableau 23 (cf. p. 152) présente deux contextes d'instanciation relatifs à l'opérateur `enlève`.

Attribut	Classe	Type	Exemple
Tâche	Objet	`Job`	`job23`
Etat	Valeur	`State`	`scheduled`
Position	Valeur	`Position`	`16`
Temps production	Valeur	`p-Time`	`13`
Durée set-up	Valeur	`s-Time`	`34`

Tableau 31. Les attributs du domaine d'ordonnancement

Prédicat	Attributs			
`(state`	`Job`	`Position`	`State`	`p-Time)`
`(before`	`Job`	`Job`	`s-Time)`	

Tableau 32. La définition des prédicats des domaines UFLP et CFLP

6.2.3. La méthode tabou dans *Alois*

Nous avons implanté deux techniques différentes de recherche tabou pour ce problème. La première, que nous appellerons désormais *A*, est celle esquissée dans Barnes & Laguna (1993). L'espace de recherche y est exploré par deux schémas d'opérateur, `échange` et `insère` (figure 29). Le schéma d'opérateur `insère` enlève une tâche de la séquence et la réinsère dans un autre endroit, tandis que `échange` échange les places de deux tâches. Chaque voisinage des deux schémas d'opérateur est de taille $O(n^2)$. A chaque schéma d'opérateur est associée une liste tabou différente d'une taille fixe de 7, dans laquelle sont inscrites, et sont donc taboues, les tâches déplacées.

Une deuxième méthode tabou, *B*, est implantée pour l'évaluation expérimentale de la décomposition des transitions (cf. p. 31). Elle utilise les schémas d'opérateur `place` et `enlève` (figure 29). Si `enlève` sort une tâche de l'ordonnancement, `place` l'insère dans un autre endroit. En les enchaînant comme cela est indiqué à la figure 3 (cf. p. 33), il est possible de réaliser les transitions `insère` et `échange` de Barnes & Laguna (1993). La taille des voisinages de ces deux opérateurs est ramenée à $O(n)$. Dans cette méthode tabou, nous utilisons, comme dans la première méthode, une liste tabou par schéma d'opérateur.

Notons que la fonction objectif que nous employons dans la modélisation de ces deux approches diffère légèrement de celle de Barnes & Laguna (1993). En effet, notre critère d'optimisation ne minimise que la durée totale de l'ordonnancement, ce qui revient à minimiser les temps de changement d'outils lors du passage d'une tâche à une autre. En outre, dans les expérimentations, nous n'avons employé que l'opérateur `insère`. La raison de ces deux limitations est simple. Les résultats des expériences conduites sur cette version simplifiée du problème d'optimisation, que nous verrons ci-dessous, ne nous ont pas encouragé à poursuivre l'implantation des deux méthodes.

6.2.4. Les expérimentations

L'évaluation expérimentale de *Alois* est conduite sur des instances de problème comprenant 15, 25, 40, 50 et 60 tâches. Pour chacune de ces tailles, nous avons construit de manière aléatoire 100 instances de problème différentes. A chaque approche *A* ou *B*, nous avons ajouté notre technique de raisonnement à partir de cas avec une mémoire de type liste plate. Les quatre approches sont évaluées sur chacune des 500 instances de problème. Chaque recherche est arrêtée lorsqu'elle n'a pas produit une amélioration de la qualité de la solution au bout de 50 itérations.

L'évaluation porte sur deux points : a) le changement de la qualité de la meilleure solution, qu'identifie chaque approche avec et sans raisonnement à partir de cas, et b) la différence de qualité des solutions et du temps d'exécution entre les deux approches *A* et *B*.

a) La qualité de la solution

Il ne nous parait pas utile de revenir sur l'ensemble des solutions des 2000 optimisations que nous avons effectuées. Le tableau 33 s'en tient donc à la présentation des moyennes des meilleures solutions obtenues, par chacune des quatre approches, sur l'ensemble des instances d'une même taille. La colonne “*Différence*” de ce tableau indique la proportion dans laquelle l'ajout du raisonnement à partir de cas a affecté la qualité de la meilleure solution. On peut alors constater que, contrairement à ce que nous espérions, l'ajout du raisonnement à partir de cas n'améliore *pas* la qualité des solutions identifiées, mais la dégrade plutôt, même si ce n'est qu'infiniment. En outre, ceci est vrai pour les deux approches, *A* et *B*, même si en ce qui concerne *A*, la dégradation est moindre.

Il s'agit maintenant de comprendre le pourquoi de ces résultats décevants. Examinons à cette fin la description des situations contenues dans les cas. Deux de celles relatives à l'opérateur `enlève` sont présentées dans le tableau 23 (chapitre 5, cf. p. 152). Elles sont reprises de manière graphique dans la figure 30. On constate qu'elles se ressemblent fortement. Elles comportent la même structure et ne se distinguent l'une de l'autre qu'au niveau des valeurs numériques décrivant les tâches. Les contextes d'instanciation sont donc extrêmement homogènes. De plus, ceci est vrai pour *chaque* contexte d'instanciation relatif à `enlève`, et il en va de même pour les autres schémas d'opérateur des deux méthodes tabou considérées.

nb tâches	Approche A	A + Alois	Différence	Approche B	B + Alois	Différence
15	212.20	212.87	-0.31%	229.37	230.87	-0.65%
25	319.33	319.33	0.00%	340.20	342.60	-0.71%
40	472.43	472.97	-0.11%	493.50	496.87	-0.68%
50	580.00	575.00	0.86%	601.80	607.60	-0.96%
60	676.80	680.40	-0.53%	709.60	709.80	-0.03%
Moyenne	452.15	452.11	-0.02%	474.89	477.55	-0.61%

Tableau 33. Qualité des solutions du problème d'ordonnancement simple à une machine

Sur ce point, ils se distinguent des contextes d'instanciations de cas des méthodes tabou des deux autres problèmes d'optimisation que nous considérons dans cette thèse. La description des situations des cas est définie par le contexte d'instanciation, qui lui tend à receler une description des entités, ici les tâches, aussi complète que possible. Toutefois, cette description est par trop homogène et ne nous parait pas contenir des informations pertinentes pour la réutilisation. On peut en conclure que la définition du contexte d'instanciation telle que nous l'employons dans *Alois* n'est pas adaptée à ce problème d'ordonnancement.

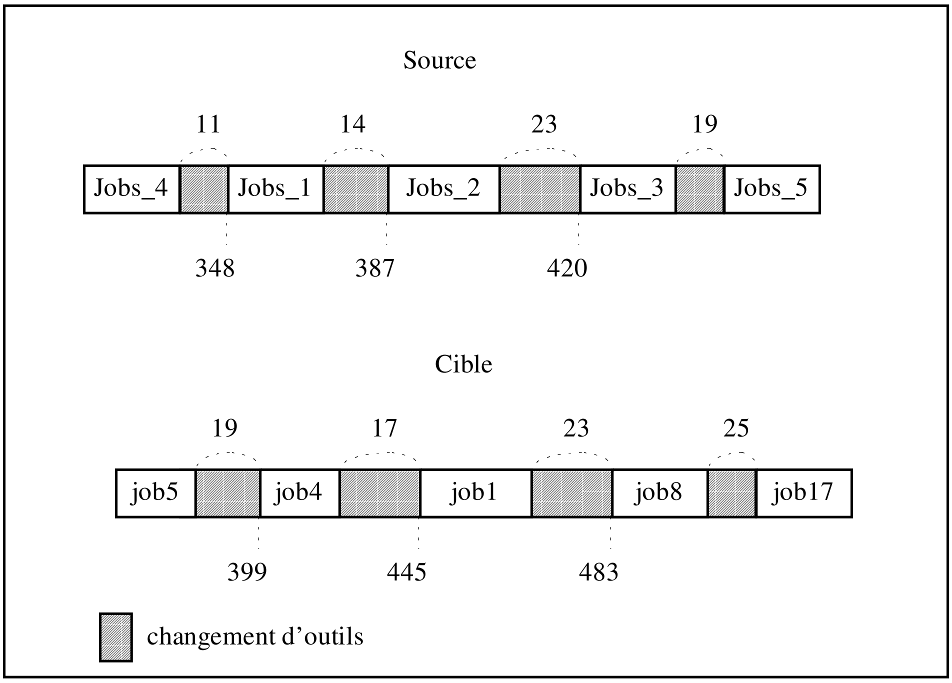


Figure 30. Représentation graphique des situations décrites au tableau 23 (cf. p. 152)

b) La décomposition des transitions

Comparons maintenant les deux approches *A* et *B* entre elles. En ce qui concerne la qualité des solutions, on remarque que l'approche *B* identifie des solutions qui sont en moyenne 5% moins bonnes que celles de *A* (tableau 33 ci-dessus). La décomposition des transitions peut donc conduire à une diminution de la qualité des solutions. Cependant, la vitesse d'exécution de l'approche *B* est bien plus élevée que celle de *A* (figure 31). Cette accélération de la recherche est due à la diminution de la taille des voisinages des opérateurs, les implantations étant identiques à ce point près. Ces résultats ne sont pas surprenant. En effet, ils ne font que confirmer les avantages et les inconvénients prévisibles de la décomposition des transitions.

A partir de ces résultats, il eut maintenant fallu explorer différentes configurations des deux méthodes tabou A et B afin de plus précisément pouvoir établir l'impact de la décomposition sur la qualité des solutions. En effet, une expérience intéressante aurait portée sur la meilleure solution atteinte au bout d'un temps de calcul égal pour les deux approches. Cependant, nous n'avons pas poursuivi les expériences dans cette direction, les résultats relatifs à l'apprentissage, le but principal de cette thèse, ne nous y ayant pas encouragés.

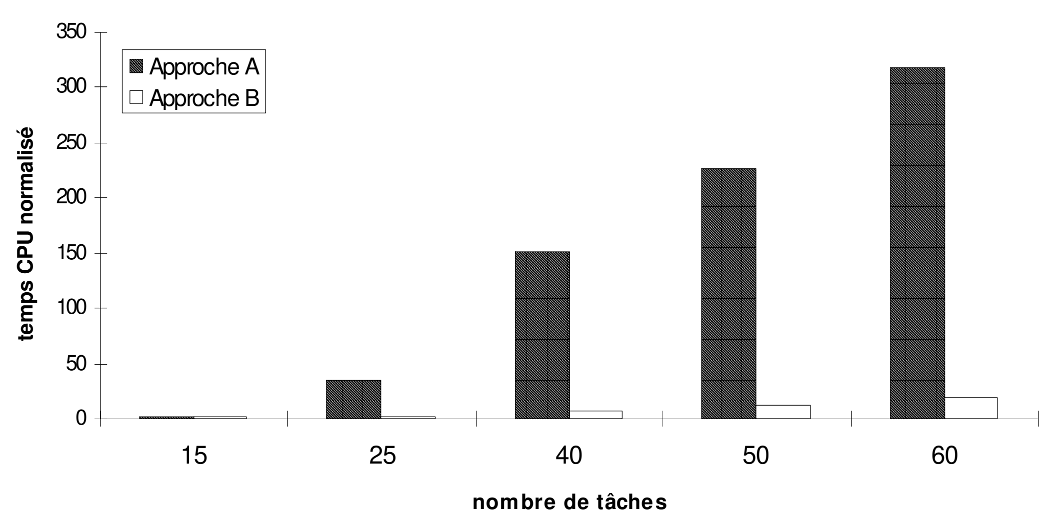


Figure 31. Temps d'exécution pour les deux méthodes tabou du problème d'ordonnancement simple, normalisé pour les instances à 15 tâches

6.3. La coloration minimale de graphes

Le *problème de la coloration minimale de graphes* est un problème classique en optimisation combinatoire. Il comporte d'importantes applications dans divers domaines, tels que la création d'emploi du temps ou l'allocation de fréquences radios aux émetteurs. Nous avons choisi ce problème pour évaluer *Alois* car il constitue l'une des premières applications de la méthode tabou. Nous avons réimplanté la technique développée par [Hertz & De Werra 1987] et testé l'impact de *Alois* sur les résultats produits par la méthode.

6.3.1. Formulation du problème

Pour un graphe simple non-orienté $G = (N, A)$, composé des noeuds $n_i \in N$ et des arcs $a_{ij} \in A$, $A \subseteq N \times N$, une coloration de G consiste en une affectation d'étiquettes, appelées *couleur*, aux noeuds de G. Toute coloration *valide* de G satisfait à la contrainte $\forall(n_i, n_j) \in A$, $\text{couleur}(n_i) \neq \text{couleur}(n_j)$. La *coloration valide minimale* de G est une coloration qui utilise le plus petit nombre de couleurs possible [Christophides 1975]. Désignons par k le nombre de couleurs utilisées par une coloration. La figure 32 présente un exemple de coloration valide minimale, avec $k = 3$, pour un graphe à 6 noeuds. Le problème de la coloration minimale d'un graphe est NP-difficile [Garey & Johnson 1979]. Cependant, il existe des heuristiques très efficaces pour sa résolution, notamment celle que nous avons réimplantée dans *Alois*.

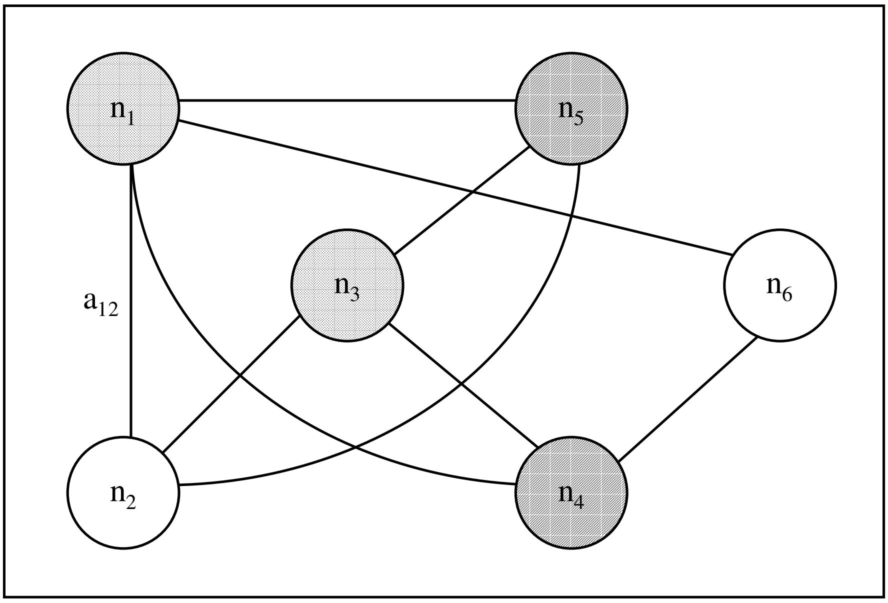


Figure 32. Exemple d'une coloration valide d'un graphe à 6 noeuds

6.3.2. La méthode tabou Tabucol

L'heuristique de coloration de graphes de Hertz & de Werra (1987), appelée *Tabucol*, cherche, pour un graphe donné, à identifier une coloration valide à pas plus de k couleurs. Si *Tabucol* parvient à identifier une coloration valide, le nombre maximal de couleurs est diminué de 1, $k' = k - 1$, et à nouveau *Tabucol* est exécutée. On continue à diminuer k, de manière itérative, jusqu'à ce que *Tabucol* ne réussisse plus à construire une coloration valide.

Tabucol est une méthode tabou employant un seul schéma d'opérateur, `recolore`, lequel change la couleur d'un noeud du graphe. L'état initial de la recherche est une coloration à k couleurs, générée de manière aléatoire. Cette coloration initiale n'est en général pas valide, c'est à dire qu'elle contient des arcs dont les deux extrémités ont la même couleur. Un tel arc constitue une *erreur de coloration*. L'ensemble des états de l'espace de recherche de *Tabucol* sont des colorations comportant des erreurs. La recherche s'arrête dès lors qu'une coloration valide est identifiée. L'ensemble candidat $V(s)$ d'un état s est composé par *rep* opérateurs `recolore`, sélectionnés de manière aléatoire dans le voisinage $N(s)$. La fonction d'évaluation $p(\cdot)$ de `recolore` calcule la différence entre le nombre d'erreurs de coloration avant et après l'application de l'opérateur. A chaque itération est sélectionné, parmi ceux qui sont membres de l'ensemble candidat, l'opérateur dont l'évaluation $p(\cdot)$ est maximale.

Le contrôle tactique de cette méthode tabou comporte les deux éléments de base que sont la liste tabou et le critère d'aspiration (chapitre 1, cf. p. 35). Une liste tabou est définie pour l'opérateur `recolore`. Elle est de taille fixe 7. Y sont inscrits les noeuds recolorés et les couleurs qu'ils avaient avant le recoloriage. Tant qu'un couple (noeud, couleur) est membre de la liste tabou, tout opérateur qui affecterait cette même couleur au même noeud est déclaré tabou et n'est pas autorisé à s'appliquer. Le critère d'aspiration, quant à lui, est défini de manière classique.

Nous avons implanté l'heuristique *Tabucol* décrite, sans reprendre deux modifications apportées par Hertz & de Werra (1987). Celles-ci consistent d'une part, à sélectionner le premier opérateur avec une évaluation positive, en évitant ainsi l'évaluation de $p(\cdot)$ pour les opérateurs restant de l'ensemble candidat. Reprendre cette extension dans *Alois* annihilerait l'intérêt de la technique elle-même, puisqu'il n'y aurait qu'un seul opérateur applicable, et donc aucun besoin de connaissances de contrôle. D'autre part, les auteurs ont introduit une technique visant à améliorer les dernières étapes de la recherche. En effet, lorsque l'ensemble des arcs en erreur possèdent un seul noeud en commun, ils enchaînent une technique de recherche exhaustive qui vérifie si une coloration valide peut être identifiée en recolorant simultanément trois noeuds au plus. Cette extension ne nous parait pas aisée à implanter dans le formalisme de représentation qu'emploie *Alois*.

Nous l'avons remplacé par un mécanisme plus simple qui arrête la recherche dès que le nombre d'erreurs de coloration est inférieur ou égal à 2.

6.3.3. La représentation du domaine

Le tableau 34 représente les deux attributs du domaine. Les noeuds du graphe sont les seuls attributs objet, et le degré des noeuds constitue le seul attribut valeur. Le domaine est composé de deux prédicats (tableau 35). Le littéral `node` décrit pour chaque noeud sa couleur et son degré, et le littéral `neighbours` indique pour chaque noeud quels sont ses voisins qui sont de la même couleur que lui, et qui constituent donc une erreur de coloration. Deux contextes d'instanciation pour l'opérateur `recolore` ont été présentés auparavant (tableau 24, cf. p. 154).

Attribut	Classe	Type	Exemple
Noeud	Objet	`Node`	`n86`
Degree	Valeur	`Degree`	`49`

Tableau 34. Les attributs du problème de la coloration de graphes

Prédicat	Attributs		
`(node`	`Node`	`Color`	`Degree)`
`(neighbours`	`Node`	`Node`	`...)`

Tableau 35. La définition des prédicats du problème de la coloration de graphes

6.3.4. Les expérimentations

L'évaluation expérimentale initiale de *Tabucol* est conduite sur des graphes, construits de manière aléatoire, d'une densité d'arcs de 0.5 et composés de 100 à 1000 noeuds [Hertz & de Werra 1987]. Nous avons fait de même. Cependant, en raison du caractère prototypique de l'implantation de *Alois*, il ne nous a pas été possible de considérer des graphes plus grands que 100 noeuds. Nous avons donc créé de manière aléatoire 20 graphes de 100 noeuds chacun. Ces graphes ont été colorés, initialement, de manière aléatoire. Nous leurs avons d'abord appliqué *Tabucol*, à laquelle nous avons ensuite ajouté *Alois*. Pour ces expériences *Alois* utilise une mémoire de cas basée sur les arbres de prototypes ainsi que la fonction d'évaluation de la similarité *RegretMatch*.

L'évaluation expérimentale porte sur le nombre d'itérations nécessaires à la construction d'une coloration valide, ainsi que sur le temps d'exécution utilisé. Sachant que Tabucol arrive à identifier, pour les graphes considérés, des colorations valides à 16 couleurs[4], nous avons appliqué les deux méthodes pour les colorations à $k = 18$, 17 et 16 couleurs.

a) Le nombre d'itérations

Le tableau 36 indique le nombre d'itérations effectuées par les deux méthodes sur chacune des 20 instances de problème. Les colonnes "*TC*" correspondent à *Tabucol*, les colonnes "*Alois*" à la combinaison de *Tabucol* avec *Alois*, et "*Diff.*" à la différence en nombre d'itérations entre les deux méthodes. Les résultats sont donnés pour $k = 18$, 17 et 16 couleurs. Les trois colonnes à droite correspondent à la somme des itérations, par instance de problème, effectuées pour obtenir une coloration valide des trois valeurs de *k*. La dernière ligne présente le total des itérations qu'a effectué une méthode pour obtenir une coloration valide sur l'ensemble des instances pour un même *k*. Les valeurs en gras indiquent les cas où *Alois* a nécessité moins d'itérations que *Tabucol*.

Nr Pb	**18 couleurs**			**17 couleurs**			**16 couleurs**			**Total**		
	TC	**Alois**	**Diff.**	**TC**	**Alois**	**Diff.**	**TC**	**Alois**	**Diff.**	**TC**	**Alois**	**Diff.**
1	189	144	**-45**	209	140	**-69**	278	290	12	676	574	**-102**
2	122	86	**-36**	143	338	195	281	207	**-74**	546	631	85
3	98	101	3	160	145	**-15**	166	397	231	424	643	219
4	131	124	**-7**	326	201	**-125**	294	298	4	751	623	**-128**
5	119	135	16	192	276	84	508	163	**-345**	819	574	**-245**
6	108	137	29	181	134	**-47**	219	215	**-4**	508	486	**-22**
7	121	116	**-5**	194	158	**-36**	312	317	5	627	591	**-36**
8	117	192	75	266	145	**-121**	388	303	**-85**	771	640	**-131**
9	121	114	**-7**	204	147	**-57**	406	159	**-247**	731	420	**-311**
10	105	164	59	138	139	1	327	248	**-79**	570	551	**-19**
11	95	104	9	194	214	20	188	347	159	477	665	188
12	131	137	6	142	160	18	593	387	**-206**	866	684	**-182**
13	104	115	11	176	148	**-28**	531	497	**-34**	811	760	**-51**
14	119	119	0	125	284	159	214	166	**-48**	458	569	111
15	111	129	18	207	168	**-39**	231	209	**-22**	549	506	**-43**
16	120	160	40	216	147	**-69**	437	258	**-179**	773	565	**-208**
17	166	104	**-62**	130	218	88	374	139	**-235**	670	461	**-209**
18	116	131	15	152	199	47	388	227	**-161**	656	557	**-99**
19	139	132	**-7**	154	291	137	484	224	**-260**	777	647	**-130**
20	106	166	60	180	230	50	238	266	28	524	662	138
Total	2438	2610	7.1%	3689	3882	5.2%	6857	5317	**-22.5%**	12984	11809	**-9.0%**

Tableau 36. Le nombre d'itérations pour la coloration de graphes à 100 noeuds

Tout d'abord on remarque que, pour les deux méthodes, le nombre d'itérations effectuées augmente lorsque *k* diminue. Ceci concorde avec l'intuition que le problème

---

[4] Ceci correspond au nombre chromatique de ces graphes [Hertz & de Werra 1987].

devient plus difficile lorsque *k* diminue. En ce qui concerne les colorations à 18 couleurs, sur 12 des 20 instances, *Tabucol* a effectué moins d'itérations que *Alois*, ce qui correspond à une diminution du nombre d'itérations de 7.1% sur l'ensemble des 20 instances. Il en va de même pour $k = 17$, où *Tabucol* requiert dans l'ensemble 5.2% d'itérations de moins que *Alois*. Cependant, chacune de ces deux méthodes s'avère plus efficace que l'autre dans 10 instances.

S'agissant des colorations à 16 couleurs, qui sont les plus difficiles, l'image change. En effet, sur 14 des 20 instances *Alois* effectue moins d'itérations que *Tabucol*, et *Alois* diminue le nombre d'itérations de 22.5% par rapport à *Tabucol*. Examinons à présent le nombre total d'itérations qu'a effectué chacune des deux méthodes pour les trois valeurs de *k*. On remarque tout d'abord que *Alois* a nécessité 9% d'itérations de moins que *Tabucol*. En outre, en nombre d'instances, sur 15 des 20, *Alois* effectue moins d'itérations que *Tabucol*.

Nous concluons de ces résultats que la technique de raisonnement à partir de cas, telle que nous l'avons conçue pour *Alois*, acquiert et réutilise des connaissances de contrôle qui sont intéressantes pour le problème de la coloration de graphes. En outre, *plus* le problème devient difficile (*k* diminue), plus la recherche doit d'être guidée afin de trouver une coloration valide, et plus l'impact des connaissances de contrôle engendrées par *Alois* augmente.

b) Le temps d'exécution

Après ces conclusions positives, quant à l'utilisabilité des connaissances de contrôle de *Alois*, étudions maintenant le temps d'exécution[5] des deux méthodes (tableau 37). Les données sont présentées de la même manière que dans le tableau précédant, à ceci prêt que les colonnes “*Ratio*” indiquent le rapport entre les temps d'exécution des deux méthodes. La dernière ligne du tableau indique, en outre, les moyennes obtenues par une méthode, sur les instances de même taille. Tout d'abord, on peut constater l'influence qu'a l'implantation sur le temps d'exécution. En effet, l'implantation originale de Hertz & de Werra (1987) a nécessité, pour la coloration à 16 couleurs de ces graphes, pas plus de 0.5s sur un ordinateur CDC Cyber 170/855, alors que notre implantation requiert aux environs de 1000s. Même si les deux temps d'exécution ne sont pas directement comparables, leur différence est frappante. Le langage de représentation basée sur la logique d'ordre 1 en est en majeure partie responsable.

---

[5] Rappelons que les expériences sont conduites sur une Sun Sparc 10 et que nous ne mesurons que le temps d'exécution, et non pas le temps CPU. S'agissant d'une station de travail à utilisateurs multiples, ces deux ne sont pas nécessairement comparables, même si les expériences ont été conduites pendant des heures creuses.

Pb Nr	18 couleurs			17 couleurs			16 couleurs			Totale		
	TC	Alois	Ratio	TC	Alois	Ratio	TC	Alois	Ratio	TC	Alois	Ratio
1	472	825	1.7	505	934	1.8	689	1918	2.8	1666	3677	2.2
2	314	651	2.1	352	2465	7	640	1090	1.7	1306	4206	3.2
3	250	656	2.6	387	736	1.9	401	2152	5.4	1038	3544	3.4
4	347	1081	3.1	758	1119	1.5	1385	3418	2.5	2490	5618	2.3
5	312	983	3.2	494	1979	4	1281	1635	1.3	2087	4597	2.2
6	488	675	1.4	455	1207	2.7	562	1740	3.1	1505	3622	2.4
7	318	895	2.8	527	979	1.9	1360	2267	1.7	2205	4141	1.9
8	391	1252	3.2	649	1548	2.4	1436	2189	1.5	2476	4989	2
9	313	703	2.2	493	1062	2.2	963	1393	1.4	1769	3158	1.8
10	267	967	3.6	345	1043	3	1356	1584	1.2	1968	3594	1.8
11	320	581	1.8	485	1166	2.4	477	2235	4.7	1282	3982	3.1
12	343	857	2.5	362	1215	3.4	1542	2838	1.8	2247	4910	2.2
13	439	873	2	424	781	1.8	1202	2693	2.2	2065	4347	2.1
14	319	1064	3.3	339	2053	6.1	535	1777	3.3	1193	4894	4.1
15	289	633	2.2	497	1085	2.2	551	1333	2.4	1337	3051	2.3
16	322	1001	3.1	547	1098	2	1039	1710	1.6	1908	3809	2
17	434	820	1.9	341	1468	4.3	864	1104	1.3	1639	3392	2.1
18	313	770	2.5	477	951	2	893	1464	1.6	1683	3185	1.9
19	357	841	2.4	413	1912	4.6	1196	3839	3.2	1966	6592	3.4
20	293	1133	3.9	476	1716	3.6	595	1654	2.8	1364	4503	3.3
Moyenne	**345**	**863**	**2.5**	**466**	**1326**	**2.8**	**948**	**2002**	**2.1**	**1760**	**4191**	**2.4**

Tableau 37. Le temps d'exécution (en s) pour le problème de la coloration de graphes

Si l'on compare maintenant le temps d'exécution des deux méthodes, on remarque sur l'ensemble des instances, que *Tabucol* est plus rapide que *Alois*. La différence est de 20% (graphe 10, $k = 16$) à 700% (graphe 2, $k = 17$), mais en moyenne elle est d'un facteur 2.4. On remarque que même pour les instances à $k = 16$ *Alois* est plus lent que *Tabucol*, malgré la diminution de plus de 20% du nombre d'itérations.

Les connaissances de contrôle qu'apprend *Alois* demandent donc davantage de temps d'exécution, bien qu'elles diminuent le nombre d'itérations. Nous assistons, en conséquence, à ce que l'on appelle en apprentissage analytique le *problème d'utilité*, où le coût de la réutilisation des connaissances apprises contrebalance le gain de celle-ci (cf. p. 53). Le remède préconisée pour faire face à ce problème est l'augmentation de la sélectivité de l'acquisition des connaissances, et l'oubli de connaissances qui sont dépassées [Smyth & Keane 1995b]. Nous pensons cependant qu'avant de s'engager dans une méthode de ce type, il faudrait d'abord procédé à une implantation de *Alois* propre au problème de la coloration de graphes. Ceci permettrait, déjà, de clairement établir l'impact sur la performance qu'a la représentation en ordre 1, et les conséquences qui découlent de l'adoption de ce type de langage (notamment s'agissant de la mesure de similarité qui nécessite, actuellement, une phase d'appariement plutôt lourde).

Conclusion
et
Perspectives

Nos travaux ont porté sur l'intégration d'une technique d'apprentissage de connaissances de contrôle, basée sur le raisonnement à partir de cas, dans la méthode tabou, qui elle est une technique heuristique de résolution de problèmes d'optimisation combinatoire. Le travail présenté a donc comporté deux volets, l'un consacré à l'utilisation du raisonnement à partir de cas à l'intérieur de la méthode tabou et l'autre à la technique du raisonnement à partir de cas en tant que telle. Tout deux méritent que l'on y revienne.

Méthode Tabou

En ce qui concerne la méthode tabou, cette thèse s'inscrit dans une ligne de recherche qui tend à mettre en oeuvre des stratégies du contrôle de l'exploration de l'espace de recherche qui restent valables à travers divers problèmes d'optimisation et diverses réalisations de méthodes tabou. Notre contribution est double. Premièrement, nous proposons le recours à une technique d'intensification basée sur la situation dans laquelle un opérateur s'applique. Nous sommes en effet partis de l'hypothèse selon laquelle l'intérêt de l'application d'un opérateur dépend de la situation locale de l'opérateur. Comme nous l'avons vu, cette démarche n'est pas celle habituellement suivie par les techniques d'intensification employées dans la méthode tabou. En principe, celles-ci emploient des critères de sélection d'opérateurs basés sur des analyses de nature statistique de la recherche antérieure.

Or, notre hypothèse est des plus ordinaires dans le domaine de l'apprentissage analytique, tel qu'il se présente dans les systèmes de génération de plans en intelligence artificielle, et de plus encore pour les systèmes de raisonnement à partir de cas. Dans ces derniers, un cas peut-être vu comme un couple situation - solution, la solution étant relative au problème décrit par la situation. L'utilisation du raisonnement à partir de cas parait, en conséquence, assez naturel dans le cadre de l'intensification telle que nous la concevons.

Ceci nous mène à ce que nous considérons comme le second apport de ce travail. En effet, l'algorithme de construction de la situation propre à un opérateur, et avec lui la technique d'intensification elle-même, est indépendant du problème d'optimisation. Sur ce point, notre approche ressemble aux autres techniques d'apprentissage développées en intelligence artificielle pour la génération de plans. Ces techniques sont, en effet, applicables à des classes entières de problèmes de génération de plans. Nous avons souligné la polyvalence de notre approche en l'appliquant à trois problèmes d'optimisation sensiblement différents.

Nous espérons par ce travail contribuer à une compréhension approfondie du processus général de l'intensification en méthode tabou. Il est, en effet, souvent difficile de transposer une technique d'intensification, conçue pour une méthode tabou et pour un problème d'optimisation, dans un autre problème d'optimisation. En définissant une technique d'intensification de manière indépendante du problème d'optimisation, un premier obstacle à son application à d'autres problèmes est franchi. Il reste alors a étudier la nature des problèmes pour lesquels la technique s'avère fructueuse.

Ce dernier constat nous conduit à évoquer les *performances* de l'approche proposée. De manière succincte, on remarque qu'*Alois* améliore les performances de la méthode tabou sur les problèmes de placement et d'allocation d'entités. Elle diminue, les concernant, l'écart entre la solution optimale et la solution identifiée par une méthode tabou simple. En outre, pour le problème de la coloration minimale de graphes *Alois* diminue le nombre d'itérations, nécessaires à l'obtention d'une coloration valide, de 20% en moyenne par rapport à une méthode tabou de qualité. En revanche, pour le problème d'ordonnancement simple aucune amélioration n'a pu être observée ni au niveau de la qualité des solutions, ni pour ce qui est du nombre des itérations.

Si on examine à présent les cas qui sont acquis lors des optimisations de ces trois problèmes, nous constatons que, pour ce qui est des deux premiers, ils sont d'une diversité qui révèle que les opérateurs sont employés dans des situations très variées. Pour ce qui est par contre des cas engendrés lors de l'optimisation d'instances du problème d'ordonnancement, il n'en va pas de même. Ces cas sont très homogènes. Ils ont tous la même structure et ne se distinguent qu'au niveau des valeurs des attributs numériques. Il n'est donc pas étonnant du tout que l'intensification ne produise guère de résultats intéressants.

Par ailleurs, la mise en oeuvre de notre technique présente, pour l'instant, l'inconvénient d'être d'un coût non négligeable en termes de temps de calcul. Les expériences réalisées sur le problème de la coloration de graphe sont de ce point de vue significatives. Ainsi, bien que le nombre d'itérations soit réduit, le temps d'exécution est lui, en revanche, le double de celui requis lors de l'exécution de notre réimplantation de la méthode tabou. On est donc face à ce que l'on appelle, en apprentissage analytique, le *problème d'utilité.* Le surcoût résultant de la manipulation des connaissances apprises contrebalance le gain obtenu grâce à leur réutilisation.

L'origine de ce problème se situe, au moins pour partie, dans l'utilisation d'un langage de représentation des problèmes qui est basé sur la logique des prédicats. Ce type de langages, dont l'utilisation est classique par les techniques d'apprentissages en intelligence artificielle pour son expressivité, entraîne une perte d'efficacité du système lors de son exécution[1]. Si, en revanche, on crée un système qui, destiné à résoudre un seul problème d'optimisation, ne requiert plus de représentation dans ce type de langages, les résultats ne pourront être que meilleurs.

Un autre point nous parait, également, pouvoir s'inscrire dans notre ligne de recherche. Dans nos travaux nous n'avons considéré que l'intensification. Or, la méthode tabou comprend une deuxième stratégie de contrôle : la diversification. Orthogonale à l'intensification, la diversification souffre peu ou prou des mêmes limites que celle-ci, et pourrait en conséquence tout autant tirer profit d'une technique de raisonnement à partir de cas. Une première approche consisterait, à partir du même mécanisme de raisonnement à partir de cas, à favoriser l'application d'opérateurs non par rapport à la similarité entre les situations, mais par rapport à leur dissimilarité. Cette stratégie de diversification encouragerait des opérateurs menant vers des situations originales.

Raisonnement à partir de cas

Dans ce travail, nous explorons l'utilisation du raisonnement à partir de cas pour l'acquisition de connaissances de contrôle à l'intérieur de la méthode tabou. Une analyse des connaissances, employées par la méthode pour la stratégie de contrôle du type intensification, nous a montré qu'elles sont de nature expérimentale et opérationnelle. Expérimentales, car elles sont construites, de manière non-supervisée, au cours même de l'optimisation ; et opérationnelles, car les techniques communes

---

[1] Ce qui, comme nous l'avons dit ailleurs, nous paraît acceptable dans le cadre de notre étude.

d'intensification n'envisagent ni leur généralisation, ni même leur réutilisation lors d'autres optimisations.

Quand bien même *Alois* ne fait pas partie du courant principal du raisonnement à partir de cas, mais seulement d'une niche qui est encore petite, il nous parait bien illustrer la flexibilité et la généralité même de celui-ci. En effet, le nombre de domaines d'application du raisonnement à partir de cas ne cesse de croître, et les méthodes élaborées assemblent une grande variété de techniques empruntées à des domaines voisins. Il en va de même dans ce travail. L'application du raisonnement à partir de cas à la résolution de problèmes d'optimisation n'est en tant que telle pas nouvelle, trois systèmes l'ont déjà fait, comme nous l'avons vu. La coopération étroite que nous proposons entre une méthode d'optimisation et le raisonnement à partir de cas est cependant originale de par son ampleur.

Nous avons présenté les *arbres de prototypes dynamiques* comme alternative aux mémoires de cas classiques. L'évaluation expérimentale de cette forme d'organisation de la mémoire de cas s'avère positive. En effet, les arbres de prototypes dynamiques cumulent les atouts des deux approches prépondérantes en matière de mémoire de cas que sont les mémoires basées sur la représentation des cas et les mémoires du type liste plate. Ils permettent d'une part une organisation structurée de la mémoire, ce qui accélère tant la récupération que la mémorisation des cas. D'autre part, ils rendent possible l'utilisation de mesures de similarité explicites, qui, parce qu'indépendantes de la représentation des entités qu'elles comparent, sont connues pour être plus facilement adaptables au domaine. Outre ces deux avantages, les arbres de prototypes dynamiques sont plutôt simples d'implantation et de configuration. Ceci devrait les rendre utilisables dans bien d'autres systèmes de raisonnement à partir de cas.

Comme nous l'avons remarqué ci-dessus, notre technique de raisonnement souffre du problème d'utilité. Or, des techniques pour y remédier existent, mais elles ne sont pas intégralement insérées dans *Alois* actuellement. L'adoption de techniques telles que l'oubli de cas ou l'acquisition très sélective de cas, pourrait constituer un remède à ce problème. Ceci d'autant plus que l'outil principal servant à distinguer les cas utiles des cas inutiles, la rétribution, n'est pas du tout utilisé dans *Alois*.

Aux termes de ces travaux, s'il s'avère que nous avons obtenu des résultats mesurés, et ceci dans des domaines précis. Il n'en demeure pas moins que nous avons explorés là une ligne de recherche nouvelle, qui a permis de mettre en évidence une proximité, entre deux techniques, jusque là ignorée. Cette voie mérite sûrement que lui soit consacrée de futures recherches.

Références bibliographiques

[Aamodt 1995] Aamodt A., Knowledge acquisition and learning by experience - the role of case-specific knowledge, in: Tecuci G., Kodratoff Y. (eds.), *Machine Learning and Knowledge Acquisition*, Academic Press, 1995, p. 197-245

[Aamodt & Plaza 1994] Aamodt A., Plaza E., Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, *AI Communications* 7 (i), 1994, p. 39-59

[Aha & Bankert 1995] Aha D., Bankert R., A comparative evaluation of sequential feature selection algorithms, International Workshop on Artificial Intelligence and Statistics, 1995, p. 1-7

[Aha & al. 1991] Aha D., Kibler D., Albert M., Instance-based learning algorithms, *Machine Learning* 6, 1991, p. 37-66

[Aikens 1985] Aikens C., Facility location models for distribution planning, *European Journal of Operational Research* 22, 1985, p. 263-79

[Alterman 1986] Alterman R., An adaptive Planner, *National Conference on Artificial Intelligence*, 1986

[Ashley 1988] Ashley K., Arguing by Analogy in Law: A Case-Based Model, in: Helman (ed.), *Analogical Reasoning: Perspectives of Artificial Intelligence, Cognitive Science, and Philosophy*, Kluwer Academic, 1988, p. 205-24

[Aykin 1995] Aykin T., The hub location and routing problem, *European Journal of Operational Research* 83 (1), 1995, p. 200-19

[Barletta & Hennessy 1989] Barletta R., Hennessy D., Case adaptation in autoclave design in: Hammond K. (ed.), *Case Based Reasoning Workshop*, Morgan Kaufmann, 1989, p. 203-7

[Barletta & Mark 1988] Barletta R., Mark W., Explanation-based indexing of cases *National Conference on Artificial Intelligence (AAAI)*, 1988, p. 541-6

[Barnes & Laguna 1993] Barnes J., Laguna M., A tabu search experience in production scheduling, in: Glover F., Laguna M., Taillard E., de Werra D. (eds.), Tabu Search, *Annals of Operations Research* 42 (1-4), Baltzer Science Publishers, Basel, Switzerland, 1993, p. 141-56

[Barnes & al. 1995] Barnes J., Laguna M., Glover F., An overview of tabu search approaches to production scheduling, in: Brown D., Scherrer W., *Intelligent Scheduling Systems*, Kluwer Academic Pub., 1995, p. 101-27

[Battiti & Tecchiolli 1994] Battiti R., Tecchiolli G., The reactive tabu search, *ORSA Journal on Computing* 6 (2), 1994, p. 126-40

[Beale 1988] Beale E., *Introduction to Optimization*, Wiley, 1988

[Beasley 1990] Beasley J.E., OR-library: distributing test problems by electronic mail, *Journal of the Operations Research Society* 41, 1990, p. 1069-72

[Beasley 1993] Beasley J.E., Lagrangien heuristics for location problems, *European Journal of Operational Research* 65, 1993, p. 383-99

[Bentley 1975] Bentley J., Multidimensional binary search trees used for associative searching, *Communications of the ACM* 18 (9), 1975, p. 509-17

[Bergadano & Gunetti 1996] Bergadano F., Gunetti D., *Inductive Logic Programming*, MIT Press, 1996

[Berge 1970] Berge C., *Graphes et Hypergraphes*, 3me éd., Bordas, Paris, 1970

[Bertsekas 1995] Bertsekas D., *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, 1995

[Biberman 1994] Biberman Y., A context similarity measure, *European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence, n°784, Springer Verlag, 1994, p. 49-63

[Bisson 1992] Bisson G., Learning in FOL with a similarity measure, *National Conference on Artificial Intelligence (AAAI)*, 1992, p. 82-7

[Bisson 1993] Bisson G., *KBG : Induction de Bases de Connaissances en Logique des Prédicats*, Ph. D. thesis, Université de Paris-Sud, Orsay, 1993

[Blazewicz & al. 1996] Blazewicz J., Domschke W., Pesch E., The job shop scheduling problem: Conventional and new solution techniques, *European Journal of Operational Research* 93 (1), 1996, p. 1-33

[Bournaud 1996] Bournaud I., *Regroupement Conceptuel pour l'Organisation de connaissances*, Ph.D. thesis, Université Paris 6, 1996

[Bournaud & Ganascia 1995] Bournaud I., Ganascia J.G., Conceptual clustering of complex objects: a generalization space based approach, *Proceedings of the International Conference on Conceptual Structures*, Springer Verlag, 1995

[Brachman & Anand 1994] Brachman R., Anand T., The process of knowledge discovery in databases, *AAAI-Workshop of Knowledge Discovery in Databases*, 1994, p. 1-12

[Brandeau & Chiu 1989] Brandeau M., Chiu S., An overview of representative problems in location research, *Management Science* 35, 1989, p. 645-74

[Brown & Scherrer 1995] Brown D., Scherrer W. (eds.), *Intelligent Scheduling Systems*, Kluwer Academic Pub., 1995, p. 101-27

[Brown 1995] Brown R., Accelerated template matching using template trees grown by condensation, *IEEE Transactions on Systems, Man and Cybernetics* 25 (3), 1995, p. 523-8

[Brown & al. 1995] Brown M., Watson I., Filer N., Separating the cases from the data: Towards more flexible case-based reasoning, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 157-68

[Bunke & Messmer 1993] Bunke H., Messmer B., Similarity Measures for Structured Representations, *European Workshop on Case-Based Reasoning*, Lecture Notes in Artificial Intelligence 837, Springer Verlag, Berlin, 1993, p. 106-18

[Bunke & Sanfeliu 1990] Bunke H., Sanfeliu A. (eds.), *Syntactic and Structural Pattern Recognition: Theory and Applications*, World Scientific Pub., Singapore, 1990

[Cardie 1993] Cardie C., Using decision trees to improve case-based learning, *International Conference on Machine Learning*, 1993, p. 25-32

[Charniak & al. 1987] Charniak F., Riesbeck K., McDermott D., Meehan J., *Artificial Intelligence Programming Techniques*, Lawrence Erlbaum Ass., Hillsdale, NJ, 1987

[Chrétienne & al. 1995] Chrétienne P., Coffman E., Lenstra J., Liu Z. (eds.),*Scheduling theory and its applications*, Wiley, 1995

[Christophides 1975] Christophides N., *Graph Theory: An Algorithmic Approach*, Academic Press, 1995

[Clark & Holte 1992] Clark P., Holte R., Lazy Partial Evaluation: An Integration of Explanation-Based Generalization and Partial Evaluation, *International Conference on Machine Learning*, 1992, p. 82-91

[Console & Torasso 1992] Console L., Torasso P., An approach to the compilation of operational knowledge from causal models, *IEEE Transactions on Systems, Man and Cybernetics* 22, 1992, p. 772-89

[Crainic & al. 1993a] Crainic T., Delorme L., Dejax P., A branch-and-bound method for multicommodity location with balancing requirements, *European Journal of Operational Research* 65, 1993, p. 368-82

[Crainic & al. 1993b] Crainic T., Gendreau M., Soriano P., Toulouse M., A tabu search procedure for multicommodity location/allocation with balancing requirements, *Annals of Operations Research* 42 (1-4), 1993, p. 359-83

[Cunningham & al. 1993] Cunningham P, Smyth B., Fin D., Cahill E., Retrieval issues in real-world CBR applications: How far can we go with discrimination nets?, *International Joint Conference on Artificial Intelligence, Workshop on Re-Use of Designs*, 1993

[Cunningham & al. 1995] Cunningham P, Smyth B., Hurley N., On the use of CBR in optimisation problems such as the TSP, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 401-10

[Daengdej & al. 1996] Daengdej J., Lukose D., Tsui E., Beinat P., Prophet L., Dynamically creating indices for 2 million cases: a real world problem, *European Workshop on Case-Based Reasoning*, 1996

[Danzig 1963] Danzig G., *Linear Programming and Extensions*, Princeton University Press, 1963

[Dasarathy 1991] Dasarathy B. (ed.), *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, 1991

[Daskin 1995] Daskin M., *Networks and Discrete Location: Models, Algorithms and Applications*, Wiley Interscience, New York, 1995

[Davis 1991] Davis L. (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991

[DeJong & Mooney 1986] DeJong G., Mooney R., Explanation-based learning: an alternative view, *Machine Learning* 1 (2), 1986, p. 145-76

[Ellis 1993] Ellis G., Efficient retrieval from hierarchies of objects using lattice operations, *International Conference on Conceptual Structures*, Lecture Notes in Artificial Intelligence, n°699, Springer Verlag, 1993, p. 274-93

[Emde & Wettschereck 1996] Emde W., Wettschereck D., Relational instance-based learning, *International Conference on Machine Learning*, 1996, p. 122-30

[Erlenkotter 1978] Erlenkotter D., A dual-based procedure for the uncapacitated facility location, *Operations Research* 26, 1978, p. 992-1002

[Esposito & al. 1991] Esposito F., Malerba D., Semeraro G., Flexible matching for noisy structural descriptions, *International Joint Conference on Artificial Intelligence* (IJCAI), 1991, p. 658-64

[Etzioni 1993] Etzioni O., A structural theory of explanation-based learning, *AI* 60, 1993, p. 93-139

[Faigle & Kern 1992] Faigle U., Kern W., Some convergence results for probabilistic tabu search, *ORSA Journal on Computing* 4 (1), 1992, p. 32-7

[Falkenhainer et al. 1989] Falkenhainer B., Forbus K., Gentner D., The Structure-Mapping Engine: Algorithm and Examples, *Artificial Intelligence* 41, 1989, p. 1-63

[Fayyad & Irani 1993] Fayyad U., Irani K., Multi-interval discretization of continuos-valued attributes for classification learning, *International Joint Conference on Artificial Intelligence* (IJCAI), 1993, p. 1022-9

[Feigenbaum 1963] Feigenbaum A., The simulation of natural learning behavior, in: Feigenbaum A., Feldman J. (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963, p. 297-309

[Fikes & Nilsson 1971] Fikes R., Nilsson N., STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* 2, 1971, p. 189-208

[Fischer 1981] Fischer M., The lagrangian method for solving integer programming problems, *Management Science* 27 (1), 1981, p. 1-19

[Fisher 1987] Fisher D., Knowledge acquisition via incremental conceptual clustering, *Machine Learning* 2, 1987, p. 139-72

[Fix & Hodges 1951] Fix E., Hodges J., *Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties*, USAF School of Medicine, Project 21-49-004, 1951, p. 261-79, in: [Dasarathy 1991]

[Forgy 1982] Forgy C., Rete: a fast algorithm for the many pattern/many object pattern matching problem, *Artificial Intelligence* 19 (1), 1982, p. 17-37

[Fox 1994] Fox M., ISIS: A Retrospective, in: Zweben M., Fox M. (eds.), *Intelligent Scheduling*, Morgan Kaufmann Publishers, San Francisco, CA, 1994, p. 3-28

[Francis & Ram 1993] Francis A., Ram A., The Utility Problem in Case-Based Reasoning, *Workshop on Case-Based Reasoning* (Abstract), Technical Report WS-93-1, 1993

[Fu 1995] Fu J., Pattern matching in directed graphs, in: Galil Z., Ukkonen F. (eds.) *Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 937, Springer, 1995, p. 64-77

[Ganascia 1987] Ganascia J.G., *Agape et Charade : deux techniques d'apprentissage symbolique appliquées à la construction de bases de connaissances*, Thèse d'Etat, Université de Paris-Sud, 1987

[Ganascia 1990] Ganascia J.G., Commentary on: Schank R., Kass A., Explanations, machine learning and creativity, in: Kodratoff Y., Michalski R. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. 3, Morgan Kaufmann, 1990, p. 49-59

[Garey & Johnson 1979] Garey M., Johnson D., *Computers and Intractability: A Guide to the Theory on NP-Completeness*, W. H. Freemann and Co., 1979

[Gates 1972] Gates G., The reduced nearest neighbor rule, *IEEE Transactions on Information Theory* 18 (3), 1972, p. 431-3

[Geibel & Wysotzki 1996] Geibel P., Wysotzki F., Learning relational concepts with decision trees, *International Conference on Machine Learning*, 1996, p. 166-74

[Gentner 1983] Gentner D., Structure Mapping: A theoretical framework for analogy, *Cognitive Science* 7, 1983, p. 155-70

[Georgeff 1987] Georgeff M., Planning, *Ann. Rev. Comput. Sci. 2*, 1987, p. 359-400

[Ghedjati - Guessoum 1994] Ghedjati - Guessoum F., *Résolution par des heuristiques dynamiques et des algorithmes génétiques du problème d'ordonnancement de type job - shop généralisé*, Thèse, Université Paris 6, 1994

[Giarrantano & Riley 1994] Giarrantano J., Riley G., *Expert Systems: Principles and Programming*, International Thompson Pub., 1994

[Glover 1986] Glover F., Future paths for integer programming and links to AI, *Computers and Operations Research* 13, 1986, p. 533-49

[Glover 1989] Glover F., Tabu Search - Part 1, *ORSA Journal on Computing* 1 (3), 1989, p. 190-206

[Glover 1990] Glover F., Tabu Search - Part 2, *ORSA Journal on Computing* 2 (1), 1990, p. 4-32

[Glover 1995] Glover F., *Tabu Search Fundamentals and Uses*, Technical Report, University of Colorado, Boulder, April 1995

[Glover & Greenberg 1989] Glover F., Greenberg H., New approaches for heuristic search: a bilateral linkage with artificial intelligence, *European Journal of Operational Research* 39 (2), 1989, p. 119-30

[Glover & Laguna 1991] Glover F., Laguna M., Target Analysis to improve a tabu search method for machine scheduling, *Arabian Journal for Science and Engineering* 16 (2/3), 1991, p. 239-53

[Goldberg 1989] Goldberg D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989

[Goodmann 1989] Goodmann M., CBR in battle planning, in: Hammond K. (ed.), *Case Based Reasoning Workshop*, Morgan Kaufmann, 1989, p. 264-9

[Grolimund 1996] Grolimund S., Accélération des mémoires de cas du type plus proches voisins, *Journées Françaises d'Apprentissage*, 1996

[Grolimund & Ganascia 1994] Grolimund S., Ganascia J.G., *A Case-Based Reasoning approach to knowledge transfer in Tabu Search*, Technical Report 94/06, Université Paris VI, Laforia, Paris, France, 1994

[Grolimund & Ganascia 1995a] Grolimund S, Ganascia J.G., Integrating Case-Based Reasoning and Tabu Search for Solving Optimisation Problems, *International Conference on Case-Based Reasoning*, Lecture Notes in Artificial Intelligence 1010, Springer-Verlag, Berlin, 1995, p. 451-60

[Grolimund & Ganascia 1995b] Grolimund S, Ganascia J.G., *Case-Based Reasoning and Tabu Search for the Single Machine Scheduling Problem: First Results*, Technical Report 95/24, Université Paris VI, Laforia, Paris, France, 1995

[Grolimund & Ganascia 1996] Grolimund S., Ganascia J.G., Speeding-up nearest neighbor memories: The template tree case memory organization, in: Saitta L. (ed.), *International Conference on Machine Learning*, Morgan Kaufmann Pub., 1996, p. 225-33

[Grolimund & Ganascia 1997] Grolimund S., Ganascia J.G., Driving Tabu Search with Case-Based Reasoning, *European Journal of Operational Research*, 1997 (to appear)

[Guedes & al. 1995] Guedes P., Saw R., Waller A., Logistic Strategy Planning: Modeling and Decision Support for the 1990s, *International Journal of Logistics Management* 6 (1), 1995, p. 37-50

[Haigh & Veloso 1995] Haigh K., Veloso M., Route planning by analogy, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, 1995, p. 169-80

[Hakimi & Kuo 1991] Hakimi S., Kuo C., On a general location - production - allocation problem, *European Journal of Operational Research* 55, 1991, p. 31-45

[Hammond 1986] Hammond K., CHEF: A model of case-based planning, *National Conference on Artificial Intelligence (AAAI)*, 1986

[Hanks & Weld 1995] Hanks S., Weld D., A domain-independent algorithm for plan adaptation, *Journal of Artificial Intelligence Research* 2, 1995, p. 319-60

[Hansen & al. 1992] Hansen P., Pedrosa E., Ribeiro C., Location and sizing of offshore platforms for oil exploration, *European Journal of Operational Research* 58, 1992, p. 202-14

[Hart 1968] Hart P., The condensed nearest neighbor rule, *IEEE Trans. on Information Theory* 1968, p. 515-6

[Haussler 1989] Haussler D., Learning conjunctive concepts in structural domains, *Machine Learning* 4, 1989, p. 7-46

[Hayes-Roth & McDermott 1977] Hayes-Roth F., McDermott J., Knowledge acquisition from structural descriptions, *International Joint Conference on Artificial Intelligence* (IJCAI), 1977,p. 356-62

[Hayes-Roth & al. 1978] Hayes-Roth F., Waterman D., Lenat D., Principles of pattern-directed inference systems, in: Waterman D., Hayes-Roth F., *Pattern-Directed Inference Systems*, Academic Press, 1978

[Hertz & De Werra 1987] Hertz A., de Werra D., Using tabu search techniques for graph coloring, *Computing* 39, 1987, p. 345-51

[Hindrichs 1988] Hindrichs T., Towards an architecture for open world problem solving, in: Kolodner J. (ed.), *Workshop in Case Based Reasoning*, Morgan Kaufmann, San Mateo, 1988, p. 182-9

[Holder 1992] Holder L., Empirical analysis of the general utility problem in machine learning, *National Conference on Artificial Intelligence (AAAI)*, 1992, p. 249-54

[Holland 1975] Holland J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975, (MIT Press, 1992)

[Holyoak & Thagard 1989] Holyoak K., Thagard P., Analogical Mapping, *Cognitive Science* 13, 1989, p. 295-355

[Hopcroft & Karp 1973] Hopcroft J., Karp R., An $n^{5/2}$ algorithm for maximal matchings in bipartite graphs, *SIAM Journal on Computing* 2, 1973, p. 225-31

[Hurley 1995] Hurley N., Evaluating the application of CBR in mesh design for simulation problems, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 205-17

[Johnson & al. 1988] Johnson D., Papadimitriou C., Yannakakis M., How easy is Local Search?, *Journal of Computers and Systems Sciences 37*, 1988, p. 79-100

[Johnson & al. 1989] Johnson D., Aragon C., McGeoch L., Schevon C., Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning, *Operations Research* 37, 1989, p. 865-92

[Johnson & al. 1991] Johnson D., Aragon C., McGeoch L., Schevon C., Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning, *Operations Research* 39, 1991, p. 378-406

[Kassou 1992] Kassou I., *Amélioration d'ordonnancements par des méthodes de voisinage*, Thèse d'Université, Université de Rouen, 1992

[Keller 1988] Keller R., Defining operationality for EBL, *Artificial Intelligence* 35, 1988, p. 227-41

[Kirkpatrick & al. 1983] Kirkpatrick S., Gelatt J., Vecchi M., Optimization by Simulated Annealing, *Science 220* (4598), 1983, p. 671-80

[Klincewicz & al. 1986] Klincewicz J., Luss H., Rosenberg E., Optimal and heuristic algorithms for multiproduct uncapacitated facility location, *Europ. Journal of Operational Research* 12, 1986, p. 251-8

[Köbler & al. 1993] Köbler J., Schöning U., Toran J., *The graph isomorphism problem: its structural complexity*, Birkhäuser, 1993

[Kodratoff & Ganascia 1986] Kodratoff Y., Ganascia J.G., Improving the generalization step in learning, in: Michalski R., Carbonell J., Mitchell T. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Volume 2, Morgan Kaufmann Pub., 1986

[Kolodner 1993] Kolodner J., *Case-Based Reasoning*, Morgan Kaufmann, 1993

[Kolodner & Simpson 1989] Kolodner J., Simpson R., The Mediator: analysis of an early case-based problem solver, *Cognitive Science* 13 (4), 1989, p. 507-49

[Korf 1987] Korf R., Planning as search: a quantitative approach, *Artificial Intelligence* 33, 1987, p. 65-88

[Koton 1988] Koton P., Reasoning about evidence in causal explanations, *National Conference on Artificial Intelligence (AAAI)*, 1988, p. 256-61

[Labbé & al. 1994] Labbé M., Schmeichel E., Hakimi S., Approximation algorithms for the capacitated plant allocation problem, *Operations Research Letters* 15 (1994), p. 115-26

[Laguna & Glover 1993] Laguna M., Glover F., Integrating target analysis and tabu search for improved scheduling systems, *Expert Systems with Applications* 6, 1993, p.287-97

[Laird & al. 1986] Laird J., Rosenbloom P., Newell A., Chunking in SOAR: The Anatomy of a General Learning Mechanism, *Machine Learning* 1, 1986, p. 11-46

[Lawler & Wood 1966] Lawler E., Wood D., Brach-and-Bound methods: a survey, *Operations Research* 14, 1966, p. 699-719

[Lawler & al. 1985] Lawler E., Lenstra J., Rinnoody Kan A. (eds.), The traveling salesman problem, North-Holland, Amsterdam, 1985

[Leake 1996] Leake D., CBR in context: the present and future, in: Leake D. (ed.), *Case-Based Reasoning: Experiences, Lessons and Future Directions*, AAAI Press, 1996

[Leake & al. 1995] Leake D., Kinley A., Wilson D., Learning to improve case adaptation by introspective reasoning and CBR, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 229-40

[Levinson 1984] Levinson R., A self-organizing retrieval system for graphs, *National Conference on Artificial Intelligence (AAAI)*, 1984, p. 203-6

[Lieber & Napoli 1996] Lieber J., Napoli A., Using Classification in Case-Based Planning, *ECAI 96*

[Lin 1965] Lin S., Computer solutions of the traveling salesman problem, *Bell Systems Technical Journal* 44, 1965, p. 2245-69

[Lloyd 1987] Lloyd J., *Foundations of Logic Programming*, Springer Verlag, 1987

[Malek 1995] Malek M. A connectionist indexing approach for CBR systems, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, 1995, p. 520-7

[Marir & Watson 1994] Marir F., Watson I., Case-based reasoning: a categorized bibliography, *The Knowledge Engineering Review* 9 (4), 1994

[Magnati & Wong 1994] Magnati T., Wong R., Network design and transportation planning: Models and algorithms, *Transportation Science* 18 (1), 1994

[McCarthy 1956] McCarthy J., The inversion of functions defined by Turing machines, *Annals of Mathematics Studies* 34, 1956, p. 177-81

[McDermott & Forgy 1978] McDermott J., Forgy C., Production system conflict resolution strategies, in: Waterman D., Hayes-Roth F. (eds.), *Pattern-Directed Inference Systems*, Academic Press, NY, 1978

[Messmer & Bunke 1993] Messmer B., Bunke H., *A network based approach to exact and inexact graph matching*, Technical report, IAM-93-021, University of Bern, Switzerland, 1993

[Michalski 1983] Michalski R., A Theory and Methodology of Inductive Learning, in: Michalski R., Carbonell J., Mitchell T. (eds.), *Machine Learning: An artificial Intelligence Approach*, Volume 1, Springer Verlag, 1983, p. 83-134

[Michalski & al. 1981] Michalski R., Stepp R., Diday E., A recent advance in data analysis: clustering objects into classes characterized by conjunctive concepts, in: Kanal L., Rosenfeld A. (eds.), *Progress in Pattern Recognition*, North-Holland Pub., 1981, p. 33-56

[Miller 1956] Miller G., The magic number seven, plus or minus two, *Psycho. Review* 63, 1956, p. 81-97

[Minoux 1987] Minoux M., *Programmation mathématique : théorie et algorithmes*, Dunod, 1987

[Minski 1963] Minski M., Steps towards artificial intelligence, in: Feigenbaum A., Feldman J. (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963, p. 406-50

[Minton 1989] Minton S., *Learning Search Control Knowledge: An Explanation-Based Approach*, Kluwer Academic Press, Boston, 1989

[Minton 1990] Minton S. Qualitative results concerning the utility of EBL, *AI* 42, 1990, p. 363-92

[Minton & al. 1994] Minton S., Bresina J., Drummond M., Total-order and partial-order planning: A comparative analysis, *Journal of Artificial Intelligence Research* 2, 1994, p. 227-62

[Mirchandani & Francis 1990] Mirchandani P., Francis R. (eds.), *Discrete Location Theory*, John Wiley & Sons, New York, 1990

[Mitchell 1983] Mitchell T., Learning and problem solving, *International Joint Conference on Artificial Intelligence* (IJCAI), 1983, p. 1140-51

[Mitchell & al. 1986] Mitchell T., Keller R., Kedar-Cabelli S., Explanation-Based Generalization: A Unifying View, *Machine Learning 1*, 1986, p. 47-80

[Miyashita & Sycara 1995] Miyashita K., Sycara K., CABINS: a framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair, *Artificial Intelligence* 76, 1995, p. 377-426

[Monticone & Funk 1994] Monticone L., Funk G., Application of the facility location problem to the problem of locating concentrators on a FAA microwave system, *An. of Operations Research* 50, 1994, p. 437-54

[Moulinier & al. 1996] Moulinier I., Raskinis G., Ganascia J.G., Text categorization: a symbolic approach, *Symposium on Document Analysis and Information Retrieval*, 1996

[Muñoz-Avila & Huellen 1995] Muñoz-Avila H., Huellen J., Retrieving cases in structured domains using goal dependencies, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 241-52

[Myles & Hand 1990] Myles J., Hand D., The multi-class metric problem in nearest neighbor discrimination rules, *Pattern Recognition* 23 (11), 1990, p. 1291-7

[Napoli 1992] Napoli A., Subsumption and classification-based reasoning in object-based representations, *European Conference on Artificial Intelligence*, 1992, p. 425-9

[Nau 1984] Nau D., Kumar V., Kanal L., General branch and bound and its relation to A* and AO*, *Artificial Intelligence* 23, 1984, p. 29-58

[Nemhauser & Wolsey 1988] Nemhauser G., Wolsey L., *Integer and Combinatorial Optimization*, Wiley Interscience, New York, 1988

[Nemhauser & al. 1989] Nemhauser G., Rinooy Kan A., Todd M. (eds.), *Optimisation*, Handbooks in Operations Research and Management Science, Volume 1, Elsevier, Amsterdam, NL, 1989

[Newell & Simon 1963] Newell A., Simon H., GPS, a program that simulates human thought, in: Feigenbaum A., Feldman J. (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963

[Newell & Simon 1976] Newell A., Simon H., Computer science as empirical inquiry - symbols and search, *Communications of the ACM* 19, 1976, p. 119-26

[Nilsson 1980] Nilsson N., *Principles of Artificial Intelligence*, Tioga Pub. Co., 1980 (trad. fr : *Principes d'intelligence artificielle*, Cepadues Editions, Toulouse, France, 1988)

[O'Leary 1993] O'Leary D., Verification and validation of case-based systems, *Expert Systems with Applications* 6, 1993, p. 57-66

[Padberg 1979] Padberg M., Covering, packing and knapsack problems, *Annals of Discrete Mathematics* 4, 1979, p. 265-87

[Papadimitriou & Steiglitz 1982] Papadimitriou C., Steiglitz K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, New Jersey, Englewood Cliffs, 1982

[Pearce & al. 1992] Pearce M., Goel A., Kolodner J., Zimring C., Sentosa L., Billington R., Case-based design support: a case study in architectural design, *IEEE Expert* 7 (5), 1992, p. 14-20

[Pearl 1984] Pearl J., *Heuristics*, Addison-Wesley, 1984

[Pérez 1996] Pérez A., Representing and learning quality-improving search control knowledge, *International Conference on Machine Learning*, 1996, p. 382-90

[Pinedo 1995] Pinedo M., *Scheduling: Theory and Algorithms*, Prentice-Hall, 1995

[Pirlot 1992] Pirlot M., General local search heuristics in combinatorial optimization: a tutorial, *Belgian Journal of Operations Research, Statistics and Computer Science* 32, 1992, p. 7-67

[Plaza 1995] Plaza E., Cases as terms: a feature term approach to the structured representation of cases, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 265-76

[Porter 1989] Porter, B., Similarity assessment: computation vs. representation, *Case Based Reasoning Workshop*, Morgan Kaufmann, 1989, p. 82-4

|Porter & al. 1990] Porter B., Bareiss R., Holte R., Concept learning and heuristic classification in weak-theory domains, *Artificial Intelligence* 45, 1990, p. 229-63

[Py 1994] Py M., Un modèle conceptuel de raisonnement par analogie, *Revue d'Intelligence Artificielle* 8 (1), 1994, p. 63-99

[Quinlan 1986] Quinlan J., Induction of decision trees, *Machine Learning* 1 (1), 1986, p. 81-106

[Quinlan 1990] Quinlan J., Learning logical definitions from relations, *Machine Learning* 5, 1990, p. 239-66

[Ramalho & Ganascia 1994] Ramalho G., Ganascia J.G., Simulating Creativity in Jazz Performance, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1994, p. 108-13

[Ramalho 1996] Ramalho G., *Construction d'un agent rationnel qui joue du jazz*, Thèse de doctorat, Université Paris 6,1996

[Ramapriyan 1976] Ramapriyan H., A multilevel approach to the detection of pictorial features, *IEEE Transactions on Computers* 25 (1), 1976, p. 66-78

[Redmond 1990] Redmond M., Distributed cases for case-based reasoning: facilitating use of multiple cases, *National Conference on Artificial Intelligence (AAAI)*, 1990, p. 304-9

[Reeves 1993] Reeves C., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Pub., London, 1993

[Rego & Roucairol 1995] Rego C., Roucairol C., Using tabu search for solving a dynamic multi-terminal truck dispatching problem, *European Journal of Operational Research* 83, 1995, p. 411-29

[Reinelt 1994] Reinelt G., *The Traveling Salesman: Computational Solutions for TSP Applications*, Lecture Notes in Computer Science, n° 840, Springer Verlag, 1994

[Ricci & Avesani 1996] Ricci F., Avesani P., Exact learning and data compression with a local asymmetrically weighted metric, *ICML-96 Workshop on Learning in Context-Sensitive Domains*, 1996

[Richter 1992] Richter M., Classification and Learning of Similarity Measures, *Jahrestagung der Gesellschaft für Klassifikation*, in: Opitz, Lausen, Klar (eds.), *Studies in Classification, Data Analysis and Knowledge Organization*, Springer Verlag, 1992

[Rifqi 1996] Rifqi M., *Mesures de comparaison, typicalité et classification d'objets flous : théorie et pratique*, Thèse de doctorat, Université Paris 6, 1996

[Riley 1991] Riley G., *Clips 5.1 Reference Manual*, Software Technology Branch, L. Johnson Space Center, Nasa, 1991

[Robert & Flores 1966] Robert S., Flores B., An engineering approach to the traveling salesman problem, *Management Science* 13, 1966, p. 269-88

[Rochat & Taillard 1995] Rochat Y., Taillard E., *Probabilistic Diversification and Intensification in Local Search for Vehicle Routing*, ORWP 95/03, Ecole Polytechnique Lausanne, 1995, (to appear in *Heuristics*)

[Rougegrez-Loriette 1994] Rougegrez-Loriette S., *Prédiction de processus à partir de comportements observés : le système Rebecas*, Thèse d'Université, Université Paris 6, 1994

[Roux 1985] Roux M., *Algorithmes de classification*, Masson, Paris, 1985

[Ruby & Kibler 1992] Ruby D., Kibler D., Learning Episodes for Optimization, *International Conference on Machine Learning*, 1992, p. 279-84

[Sacerdoti 1977] Sacerdoti E., *A Structure for Plans and Behavior*, North-Holland, 1977

[Salzberg 1991] Salzberg S., A nearest hyperrectangle learning method, *Machine Learning* 6, 1991, p. 277-309

[Schaeffer 1993] Schaeffer V., *Spécification et Optimisation de mission pour un robot mobile autonome*, Ph.D. thesis, Université Pierre et Marie Curie - Paris 6, Paris, 1993

[Schank 1982] Schank R., *Dynamic Memory: A theory of learning in computers and people*, Cambridge Univ. Press, New York, 1982

[Shapiro & Haralick 1990] Shapiro L., Haralick R., Matching relational structures using discrete relaxation, in: Bunke H., Sanfeliu A., (eds.) *Syntactic and Structural Pattern Recognition: Theory and Applications*, World Scientific Pub., Singapore, 1990, p. 179-95

[Shasha & al. 1994] Shasha D., Wang J., Zhang K., Shih F., Exact and approximate algorithms for unordered tree matching, *IEEE Transactions on Systems, Man and Cybernetics* 24 (4), 1994, p. 668-678

[Shimazu & al. 1993] Shimazu H., Kitano H., Shibata A., Retrieving cases from relational data-bases: another stride towards corporate-wide case-based systems, *International Joint Conference on Artificial Intelligence* (IJCAI), 1993, p. 909-14

[Simon 1973] Simon H., The structure of ill-structured problems, *Artificial Intelligence* 4, 1973, p. 181-201

[Skala 1994] Skala D., Prototype and feature selection by sampling and random mutation hill climbing algorithms, *International Conference on Machine Learning*, 1994, p. 293-301

[Skorin-Kapov 1990] Skorin-Kapov J., Tabu search applied to the quadratic assignment problem, *ORSA Journal on Computing* 2 (1), 1990, p. 33-45

[Smyth & Keane 1995a] Smyth B., Keane M., Experiments on adaptation-guided retrieval in case-based design, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 313-24

[Smyth & Keane 1995b] Smyth B., Keane M., Remembering to forget: a competence-preserving case deletion policy for case-based reasoning systems, *International Joint Conference on Artificial Intelligence* (IJCAI), 1995, p. 377-82

[Stanfill & Waltz 1986] Stanfill C., Waltz D., Towards memory-based reasoning, *Communications of the ACM* 29 (12), 1986, p. 1213-28

[Steiglitz & Weiner 1968] Steiglitz K., Weiner P., Some improved algorithms for computer solutions of the TSP, *Allerton Conference on Circuit and Systems Theory*, 1968, p. 814-21

[Stepp & Michalski 1986] Stepp R., Michalski R.S., Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects, in: Michalski R., Carbonell J., Mitchell T. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Volume 2, Morgan Kaufmann Pub., 1986

[Strube & al. 1995] Strube G., Enzinger A., Janetzko D., Knauff M., Knowledge engineering for CBR systems from a cognitive science perspective, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 548-58

[Subramanian & Hunter 1992] Subramanian D., Hunter S., Measuring utility and the design of provably good EBL algorithms, *International Conference on Machine Learning*, 1992, p. 426-35

[Surma & Vanhoof 1995] Surma J., Vanhoof K., Integrating rules and cases for the classification task, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, Berlin, 1995, p. 325-34

[Sutton1988] Sutton, Learning to predict by the method of temporal differences, *Machine Learning 3*, 1988, p. 9-44

[Sycara & Miyashita 1994] Sycara K., Miyashita K., Case-based acquisition of User Preferences for Solution Improvement in ill-structured Domains, *National Conference on Artificial Intelligence*, 1994, p. 44-9

[Taillard 1991] Taillard E., Robust taboo search for the quadratic assignment problem, *Parallel Computing 17*, 1991, p. 443-55

[Tambe & al. 1990] Tambe M., Newell A., Rosenbloom P., The problem of expensive junks and its solution by restricting expressiveness, *Machine Learning* 5, 1990, p. 299-349

[Tambe & Rosenbloom 1993] Tambe M., Rosenbloom P., On the Masking Effect, *National Conference on Artificial Intelligence (AAAI)*, 1993, p. 526-33

[Tate & al. 1990] Tate A., Hendler J., Drummond M., A review of AI planning techniques, in: Allen J., Hendler J., Tate A. (eds.), *Readings in Planning*, Morgan Kaufmann, San Mateo, 1990, p. 26-49

[Thomason 1990] Thomason M., Introduction and Overview, in: Bunke H., Sanfeliu A., (eds.) *Syntactic and Structural Pattern Recognition: Theory and Applications*, World Scientific Pub., Singapore, 1990, p. 3-25

[Tversky 1977] Tversky A., Features of similarity, *Psychological Review* 84, 1977, p. 327-52

[Van Harmelen & Bundy 1988] Van Harmelen F., Bundy A., EBG = partial evaluation, *Artificial Intelligence* 36, 1988, p. 401-12

[Van Hentenryck 1989] Van Hentenryck P., *Constraint Satisfaction in Logic Programming*, MIT Press, 1989

[Van Laarhoven & Aarts 1988] Van Laarhoven P., Aarts E., *Simulated Annealing: Theory and Applications*, Kluwer, Dordrecht, 1988

[Veloso 1994] Veloso, M., *Planning and learning by analogical reasoning*, Lecture Notes in Computer Science 886, Springer-Verlag, Berlin, 1994

[Vidal 1986] Vidal E., An algorithm for finding nearest neighbors in (approximately) constant average time, *Pattern Recognition Letter* 4, 1986, p. 145-57

[Vidal 1994] Vidal E., New formulation and improvements of the nearest-neighbor approximating and eliminating search algorithm (AESA), *Pattern Recognition Letter* 15, 1994, p. 1-7

[Voisin & Devijver 1987] Voisin J., Devijver P., An application of the multi-edit condensation technique to the reference selection problem in a print recognition system, *Pattern Recognition* 20 (5), 1987, p. 465-74

[Vosselmann 1992] Vosselmann G., *Relational Matching*, Springer Verlag, 1992

[Watson & Marir 1994] Watson I., Marir F., Case-based reasoning: a review, *The Knowledge Engineering Review* 9 (4), 1994

[Wess & al. 1993] Wess S., Althoff K., Derwand G., Using k-d trees to improve the retrieval step in case-based reasoning, *European Workshop on Case-Based Reasoning*, Lecture Notes in Artificial Intelligence 837, Springer Verlag, Berlin, 1993, p. 167-81

[Wettschereck & Aha 1995] Wettschereck D., Aha D., Weighting features, *International Conference on Case-Based Reasoning*, 1995, Lecture Notes in Computer Science 1010, Springer-Verlag, 1995, p. 346-58

[Wettschereck & al. 1996] Wettschereck D., Aha D., Mohri T., A review of feature weighting algorithms, *Artificial Intelligence Review Journal*, 1996 (to appear)

[Wielinga & al. 1992] Wielinga B., Schreiber T., Breuker J., KADS: a modeling approach to knowledge engineering, *Knowledge Acquisition* 4 (1), 1992, p. 5-53

[Williams 1993a] Williams H.P., *Model building for mathematical programming*, John Wiley, 3rd ed., 1993

[Williams 1993b] Williams H.P., *Model Solving in Mathematical Programming*, John Wiley, 1993

[Winston 1975] Winston P., Learning structural descriptions from examples, in: Winston P., *The Psychology of Computer Vision*, McGraw-Hill, 1975

[Wirth 1975] Wirth N., *Algorithmen und Datenstrukturen*, Teubner Verlag, Stuttgart, 1975

[Woodruff 1994] Woodruff D., *Proposal for Chunking and Tabu Search*, Technical Report, University of California Davis, Dec. 1994

[Zanakakis & Evans 1989] Zanakakis S., Evans J., Vazacopoulos A., Heuristic methods and applications: a categorized survey, *European Journal of Operational Research* 43, 1989, p. 88-110

[Zerr 1992] Zerr F., *Ariane : Intégration d'un Mécanisme d'Apprentissage par Explications dans un Générateur de Systèmes Experts Industriel*, Thèse, Université Paris-Sud, Orsay, 1992

[Zerr & Ganascia 1990] Zerr F., Ganascia J.G., Comparaison du chunking avec l'EBG implémenté sur SOAR, *Journées Francophones d'Apprentissage*, 1990

[Zucker 1996] Zucker J.D., *Appariement et changements de représentation pour l'apprentissage symbolique*, Thèse de doctorat, Université Paris 6, 1996

[Zucker & Ganascia 1996] Zucker J., Ganascia J.G., Representation changes for efficient learning in structural domains, *International Conference on Machine Learning*, 1996, p. 543-51

[Zweben & Fox 1994] Zweben M., Fox M. (eds.), *Intelligent Scheduling*, Morgan Kaufmann Publishers, San Francisco, CA, 1994

Annexe A. Implantation du problème de placement et d'allocation

Cette annexe contient la représentation du problème de placement et d'allocation d'entités avec restriction de capacité telle que employée par *Alois* au cours des expérimentations présentées. La modélisation comporte plusieurs fichiers, dont le premier est, dans la terminologie de *Clips*, un fichier batch. Il contrôle l'exécution de *Alois* du début à la fin. Les autres fichiers décrivent, dans l'ordre que nous les présentons ci-dessous, le contrôle de l'enchaînement des opérateurs lors de la recherche, la construction de l'état initial, l'opérateur qui ferme les entrepôts, et en dernier lieu les autres opérateurs de la théorie du domaine.

Le fichier batch

```
; Fichier TRSPN.opl : lancement de Alois par "alois -f TRSPN.opl"
; capacitated transportation problem
;
(defglobal
        ?*printFlag* = 0                ;O= no trace, 1 = trace
        ?*tabuWareHouse* = (mv-append 0 0 0 0 0 0 0)
                                        ;tabu-list for closed warehouses
                                        ;initial size = 7
        ?*tabuWareMin* = 6
        ?*tabuWareMax* = 10
        ?*variableTabuList* = 0         ;0 = fixed size, 1 = variable
)

;case library options
;====================
(caselib none)

(interface minimal)
(reward best)

(caselibtype templatetree)

; Domain definitions
(defliteral warehouse 1 5)
(defliteral client 1 1)
(defliteral clientArc 2 2)

(defmvliteral clientList 2 0)


(defattrclass Warehouse Object)
(defattrclass Client    Object)
(defattrclass Quantity  Integer)
(defattrclass Cost      Integer)


(defposition client Client    1)
(defposition client Quantity  2)

(defposition warehouse Warehouse 1)
(defposition warehouse Cost      2)
(defposition warehouse Quantity  3)
(defposition warehouse Quantity  4 GT0)
(defposition warehouse Quantity  5)
(defposition warehouse Cost      6)

(defposition clientArc Warehouse 1)
(defposition clientArc Client    2)
(defposition clientArc Quantity  3 GT0)
```

```
(defposition clientArc Cost      4)

(defposition clientList Warehouse 1)
(defposition clientList Client    2)

(defattr Client client1 client2 client3 client4 client5 client6 client7 client8 client9
client10 client11 client12 client13 client14 client15 client16 client17client18 client19
client20 client21 client22 client23 client24 client25 client26client27 client28 client29
client30 client31 client32 client33 client34 client35client36 client37 client38 client39
client40 client41 client42 client43 client44client45 client46 client47 client48 client49
client50)

(defattr Warehouse warehouse1 warehouse2 warehouse3 warehouse4 warehouse5warehouse6 warehouse7
warehouse8 warehouse9 warehouse10 warehouse11 warehouse12warehouse13 warehouse14 warehouse15
warehouse16 warehouse17 warehouse18 warehouse19 warehouse20 warehouse21 warehouse22
warehouse23 warehouse24 warehouse25 warehouse26 warehouse27 warehouse28 warehouse29
warehouse30 warehouse31 warehouse32 warehouse33 warehouse34 warehouse35 warehouse36
warehouse37 warehouse38 warehouse39 warehouse40 warehouse41 warehouse42 warehouse43
warehouse44 warehouse45 warehouse46 warehouse47 warehouse48 warehouse49 warehouse50)

(defopschema CloseWarehouse)

(verifydomain)

; load construct files
(load "trspctrl.clp")        ;control structures
(load "trspinit.clp")        ;initial feasibel state generation
(load "trspware.clp")        ;decide on warehouses to be closed
(load "trspclnt.clp")        ;client management

; execute
(reset)
(assert (control initialAffectClients))
(unwatch all)

(run)

; close Alois
(quit)
```

Contrôle de l'enchaînement des opérateurs

```
; Fichier trspctrl.opl
; capacitated transportation problem
;
; control structures
;
;-----------------------------------------------------------------------------
; global varaibles
;
(defglobal
        ?*x*             = 0               ;salience holder
        ?*y*             = 0               ;internal variable
        ?*Qaffect*       = 0               ;holds temporary affect quantities
        ?*countBest*     = 0               ;solution improvement counter
        ?*countNoBest*   = 0               ;non-improvement counter
        ?*bestSolution*  = 100000          ;quality of the best solution
        ?*bestIteration* = 0               ;the iteration where the best sol found
)

;-----------------------------------------------------------------------------
; functions
;
(defgeneric transportCost "determine transportation cost")

(defmethod transportCost  "add fixed cost"
        ((?Quantity INTEGER)
         (?Distance FLOAT)
         (?FixedCost INTEGER))
        (if (> ?Quantity 0) then
                (min (div (+ ?FixedCost (* ?Quantity ?Distance)) 100) 10000)
         else
                (max (div (+ ?FixedCost (* ?Quantity ?Distance)) 100) 10000)
        )
)

;-----------------------------------------------------------------------------
```

```
; tabu list management
;
(deffunction updateTabuList "variable lenght tabu list"
        (?newElement $?oldList)
        (if (eq ?*variableTabuList* 1) then
                (progn (bind ?rand (/ (random) 32767))      ;set a random variable
                  ;if <0.2 and size > minSize -> reduce size
                  (if (and (< ?rand 0.2) (> (length ?oldList) ?*tabuWareMin*))
                    then
                         (bind ?oldList (mv-delete 1 ?oldList)))
                  ;if >0.8 and size < maxsize -> enlarge
                  (if (and (< 0.8 ?rand) (< (length ?oldList) ?*tabuWareMax*))
                    then
                         (bind ?oldList (mv-append 0 ?oldList)))
                  ;else let size invariant
                  ;update list
                  (mv-append (mv-delete 1 ?oldList) ?newElement)
                )
         else
                (mv-append (mv-delete 1 ?oldList) ?newElement)
        )
)

;-----------------------------------------------------------------------------
; initial state management
;
(defrule AllClientsSatisfied
 ?f1 <- (control initialAffectClients)
        (not (client ? ?Qdemand&:(> ?Qdemand 0)))
=>
(retract ?f1)
(printout t "All clients affected" crlf)
(timer restart)
(assert (control feasibleState))
)


;-----------------------------------------------------------------------------
; local optima + new best solutions
;
(defrule BestFeasibleState
 ?f0 <- (control feasibleState)
        (test (< ?*objective* ?*bestSolution*))
=>
(bind ?*bestSolution* ?*objective*)
(bind ?*countBest* (+ ?*countBest* 1))
(bind ?*bestIteration* (+ ?*countBest* ?*countNoBest*))
(printout t crlf "new best solution: " ?*objective* " (" ?*countBest* ")" crlf)

(if (eq ?*printFlag* 1) then
  (printout t "===== start next iteration =====" crlf)
)
(retract ?f0)
(assert (control startNextIteration))
)


;-----------------------------------------------------------------------------
; local optima + no solution improvement
;
(defrule BullshitFeasibleState
 ?f0 <- (control feasibleState)
        (test (>= ?*objective* ?*bestSolution*))
=>
(bind ?*countNoBest* (+ ?*countNoBest* 1))
(printout t crlf "solution: " ?*objective* " (" ?*countNoBest* ")" crlf)

(if (eq ?*printFlag* 1) then
  (printout t "===== start next iteration =====" crlf)
)
(retract ?f0)
(assert (control startNextIteration))
)


;-----------------------------------------------------------------------------
; state transition diagramm
;
;-----------------------------------------------------------------------------
; 'startNextIteration' --> 'separateClient'
;
```

```
(defrule StartNextIteration_SeperateClient
 ?f1 <- (closeWarehouse ?W)
=>
(retract ?f1)
(if (eq ?*printFlag* 1) then
  (printout t "'startNextIteration' --> 'separateClient'" crlf)
)
(assert (control separateClient ?W))
)


;---------------------------------------------------------------------------
; 'separateClient' --> 'affectClient'
;
(defrule SeparateClient_AffectClient
 ?f0 <- (control separateClient ?W)
        (not (clientArc ?W ? ?Quan&:(> ?Quan 0) ?))
=>
(retract ?f0)
(if (eq ?*printFlag* 1) then
  (printout t "'separateClient' --> 'affectClient'" crlf)
)
(assert (control affectClient))
)


;---------------------------------------------------------------------------
; 'affectClient' --> 'feasibleState'
;
(defrule AffectClient_FeasibleState
 ?f0 <- (control affectClient)
        (not (client ? ?Qdemand&:(> ?Qdemand 0)))
=>
(retract ?f0)
(timer getelapsed)
(if (eq ?*printFlag* 1) then
  (printout t "'affectClient' --> 'feasibleState'" crlf)
)
(assert (control feasibleState))
)
```

Construction de l'état initial

```
; Fichier trspinit.opl
; capacitated location allocation problem
;
;---------------------------------------------------------------------------
; initial state generation rules
;
(defrule InitialSatisfyClient
        (declare (salience ?*x*))
        (control initialAffectClients)
 ?f1 <- (client    ?C ?Qdemand&:(> ?Qdemand 0))
 ?f2 <- (warehouse ?W ?Fixed ?Capa ?Qin ?Qout&:(> ?Capa ?Qout) ?OldTotalCost)
 ?f3 <- (clientArc ?W ?C   ?QuanOld ?CostOld)
 ?f4 <- (clientList ?W $?CL)
        (cnr ?C ?cnrC)
        (cnr ?W ?cnrW)
        (or (distance ?cnrC ?cnrW ?distance)
            (distance ?cnrW ?cnrC ?distance))
        (test (progn (bind ?*Qaffect* (min ?Qdemand (- ?Capa ?Qout)))
                     (bind ?*x* (- 0 (div
                     (transportCost ?*Qaffect* ?distance) ?*Qaffect*)))
               TRUE))
=>
(bind ?Qaffect (min ?Qdemand (- ?Capa ?Qout)))

(if (eq ?Qout 0) then
  (bind ?*y* (transportCost ?Qaffect ?distance ?Fixed))
else
  (bind ?*y* (transportCost ?Qaffect ?distance))
)

(bind ?*objective* (+ ?*objective* ?*y*))

(if (eq ?*printFlag* 1) then
  (printout t "warehouse " ?W " -> client " ?C " quan: " ?Qaffect crlf)
```

```
)

(retract ?f1 ?f2 ?f3)
(assert (client ?C =(- ?Qdemand ?Qaffect))
        (warehouse  ?W ?Fixed ?Capa =(+ ?Qin ?Qaffect) =(+ ?Qout ?Qaffect)
                                =(+ ?OldTotalCost ?*y*))
        (clientArc  ?W ?C   =(+ ?QuanOld ?Qaffect)
                                =(+ ?CostOld (transportCost ?Qaffect ?distance)) )
        )
(if (not (member ?C ?CL)) then
  (progn (retract ?f4)
   (assert (clientList ?W =(mv-append ?CL ?C))) )
  )
)
```

Fermeture d'un entrepôt

```
; Fichier trspware.opl
; capacitated location allocation problem
;

; Close a warehouse
;
(defrule CloseWarehouse
        (declare (salience ?*x*))
 ?f0 <- (control startNextIteration)
        (warehouse ?W ? ? ? ?Qout&:(> ?Qout 0) ?TotalCost)
        (test (not (member ?W ?*tabuWareHouse*)))
        (test (progn (bind ?*x* (min 10000 (div (* ?TotalCost 100) ?Qout)))
               TRUE))
=>
(retract ?f0)

(bind ?*tabuWareHouse* (updateTabuList ?W ?*tabuWareHouse*))
(printout t "close " ?W " (" (length ?*tabuWareHouse*) ")" crlf)
(assert (closeWarehouse ?W))
)
```

La théorie du domaine

```
; Fichier trspclnt.opl
; capacitated transportation problem
;

;----------------------------------------------------------------------------
; Client management
;  - separate a client from its warehouses
;  - affect a client to a warehouse

;----------------------------------------------------------------------------
; separate one client from one warehouse
;
(defrule separateOneClient
        (declare (salience ?*x*))
        (control separateClient ?W)
 ?f0 <- (warehouse  ?W ?Fixed ?Capa ?Qin ?Qout ?TotalCost)
 ?f1 <- (clientArc  ?W ?C ?Quan ?Cost)
 ?f2 <- (client     ?C ?QuanOld)
 ?f3 <- (clientList ?W $?CL)
        (test (member ?C ?CL))
        (test (progn (bind ?*x* (- 0 ?Cost))
               TRUE))
=>
(if (eq ?Qout ?Quan) then
  (bind ?*y* ?TotalCost)
else
  (bind ?*y* ?Cost)
)

(bind ?*objective* (- ?*objective* ?*y* ))

(if (eq ?*printFlag* 1) then
  (printout t "separate: warehouse " ?W " -> client " ?C " quan: " ?Quan crlf)
)
```

```
(retract ?f0 ?f1 ?f2 ?f3)
(assert (client ?C =(+ ?QuanOld ?Quan))
        (clientArc ?W ?C 0 0)
        (clientList ?W =(mv-delete (member ?C ?CL) ?CL))
        (warehouse ?W ?Fixed ?Capa =(- ?Qout ?Quan) =(- ?Qout ?Quan)
                                                    =(- ?TotalCost ?*y*))
        (lastServedBy ?C ?W)
        )
)

;-----------------------------------------------------------------------------
; affect a client to any warehouse
;
(defrule affectClient
        (declare (salience ?*x*))
        (control affectClient)
 ?f0 <- (client    ?C ?Qdemand&:(> ?Qdemand 0))
 ?f1 <- (lastServedBy ?C ?Wold)
 ?f3 <- (warehouse ?W&:(neq ?W ?Wold) ?Fixed ?Capa ?Qin ?Qout&:(> ?Capa ?Qout)
                                 ?TotalCost)
 ?f2 <- (clientArc ?W ?C   ?QuanOld ?CostOld)
 ?f4 <- (clientList ?W $?CL)
        (cnr ?C ?cnrC)
        (cnr ?W ?cnrW)
        (or (distance ?cnrC ?cnrW ?distance)
            (distance ?cnrW ?cnrC ?distance))
        (test (not (member ?W ?*tabuWareHouse*)))
        ; restriction de la capacité de l'entrepôt
        (test (progn (bind ?*Qaffect* (min ?Qdemand (- ?Capa ?Qout)))
                     (bind ?*x* (- 0 (transportCost ?*Qaffect* ?distance)))
               TRUE))
=>
(bind ?Qaffect (min ?Qdemand (- ?Capa ?Qout)))

(if (eq ?Qin 0) then
  (bind ?*y* (transportCost ?Qaffect ?distance ?Fixed))
else
  (bind ?*y* (transportCost ?Qaffect ?distance))
)

(bind ?*objective* (+ ?*objective* ?*y*))

(if (eq ?*printFlag* 1) then
  (printout t "warehouse " ?W " -> client " ?C " quan: " ?Qaffect crlf)
)

(retract ?f0 ?f1 ?f2 ?f3)
(assert (client ?C =(- ?Qdemand ?Qaffect))
        (warehouse ?W ?Fixed ?Capa  =(+ ?Qin    ?Qaffect)
                        =(+ ?Qout    ?Qaffect) =(+ ?TotalCost ?*y*))
        (clientArc ?W ?C   =(+ ?QuanOld ?Qaffect)
                        =(+ ?CostOld (transportCost ?Qaffect ?distance)))
        )
(if (not (member ?C ?CL)) then
  (progn (retract ?f4)
         (assert (clientList ?W =(mv-append ?CL ?C))) )
  )
)
```

Annexe B. Un exemple de la fonction *RegretMatch*

Cette annexe présente le fonctionnement de l'heuristique d'appariement *RegretMatch* introduite au tableau 25 (cf. p. 163). L'exemple porte sur les deux situations présentées au tableau 24 (cf. p. 154). Le texte en italique est un commentaire de la trace d'exécution de *Alois*.

Description de `Source`

Strate 1:

```
         role(Node_1)
```

Strate 2:

```
Literal: node(Node_1 Color_1 49)
Literal: neighbours(Node_1 Node_2 Node_3 Node_4 Node_5)
```

Strate 3:

```
Literal: node(Node_3 Color_1 49)
Literal: node(Node_2 Color_1 49)
Literal: node(Node_4 Color_1 43)
Literal: node(Node_5 Color_1 47)
Literal: neighbours(Node_2 Node_3 Node_6 Node_1)
Literal: neighbours(Node_3 Node_2 Node_1 Node_4)
Literal: neighbours(Node_4 Node_1 Node_3)
Literal: neighbours(Node_6 Node_2 Node_5)
Literal: neighbours(Node_5 Node_1 Node_6)
```

Construction de la partie correspondant à* `Source` *du graphe bipartite. *Chaque ligne correspond à un noeud du graphe bipartite. Les lignes commençant par 'Node' correspondent aux attributs de* `Source`*, et les lignes 'Arc' aux liens. 'Lx' indique le numéro 'x' du littéral dans lequel est référencé pour la première fois le noeud, et 'Py' indique sa position 'y' dans ce littéral. après 'Var' sont indiqués les numéros des variables d'appariements qui sont présentes dans le graphe bipartite.*

Attributs

```
Node Node_1, L0 P1 Var 1
Node Color_1, L1 P2 Var 2
Node 49, L1 P3 Var 3
Node Node_2, L1 P2 Var 4 5 6
Node Node_3, L1 P2 Var 7 8 9
Node Node_4, L1 P2 Var 10 11 12
Node Node_5, L1 P2 Var 13 14 15
Node 43, L2 P3 Var 16 17 18
Node 47, L2 P3 Var 19 20 21
Node Node_6, L2 P2
```

Liens

```
Arc from Node_1 to Color_1, L1 P1 Va
Arc from Node_1 to 49, L1 P2 Var 23
Arc from Node_1 to Node_2, L1 P1 Var
Arc from Node_1 to Node_3, L1 P1 Var
Arc from Node_1 to Node_4, L1 P1 Var
Arc from Node_1 to Node_5, L1 P1 Var
Arc from Node_3 to Color_1, L2 P1 Va
Arc from Node_3 to 49, L2 P2 Var 39
Arc from Node_2 to Color_1, L2 P1 Va
Arc from Node_2 to 49, L2 P2 Var 45
Arc from Node_4 to Color_1, L2 P1 Va
Arc from Node_4 to 43, L2 P2 Var 51
Arc from Node_5 to Color_1, L2 P1 Va
Arc from Node_5 to 47, L2 P2 Var 57
Arc from Node_2 to Node_3, L2 P1 Var
Arc from Node_2 to Node_6, L2 P1 Var
Arc from Node_2 to Node_1, L2 P1 Var
Arc from Node_3 to Node_2, L2 P1 Var
Arc from Node_3 to Node_1, L2 P1 Var
Arc from Node_3 to Node_4, L2 P1 Var
Arc from Node_4 to Node_1, L2 P1 Var
Arc from Node_4 to Node_3, L2 P1 Var
Arc from Node_6 to Node_2, L2 P1 Var
Arc from Node_6 to Node_5, L2 P1 Var
Arc from Node_5 to Node_1, L2 P1 Var
Arc from Node_5 to Node_6, L2 P1 Var
```

Description de `Cible`

Strate 1:

```
         role(n_86)
```

Strate 2:

```
Literal: node(n_86 col_6 48)
Literal: neighbours(n_86 n_27 n_26 n
```

Strate 3:

```
Literal: node(n_27 col_6 57)
Literal: node(n_26 col_6 52)
Literal: node(n_2 col_6 51)
Literal: neighbours(n_73 n_26)
Literal: neighbours(n_26 n_73 n_86 n
Literal: neighbours(n_27 n_86 n_2)
```

```
Literal: neighbours(n_2 n_26 n_86 n_27)
```

Construction de la partie correspondant à `Cible` *du graphe bipartie.*

Attributs

```
Node n_86, L0 P1 Var 1
Node col_6, L1 P2 Var 2
Node 48, L1 P3 Var 3
Node n_27, L1 P2 Var 4 7 10 13
Node n_26, L1 P2 Var 5 8 11 14
Node n_2, L1 P2 Var 6 9 12 15
Node 57, L2 P3 Var 16 19
Node 52, L2 P3 Var 17 20
Node 51, L2 P3 Var 18 21
Node n_73, L2 P1
```

Liens

```
Arc from n_86 to col_6, L1 P1 Var 22
Arc from n_86 to 48, L1 P2 Var 23
Arc from n_86 to n_27, L1 P1 Var 24 27 30 33
Arc from n_86 to n_26, L1 P1 Var 25 28 31 34
Arc from n_86 to n_2, L1 P1 Var 26 29 32 35
Arc from n_27 to col_6, L2 P1 Var 36 42 48 54
Arc from n_27 to 57, L2 P2 Var 39 45 51 57
Arc from n_26 to col_6, L2 P1 Var 37 43 49 55
Arc from n_26 to 52, L2 P2 Var 40 46 52 58
Arc from n_2 to col_6, L2 P1 Var 38 44 50 56
Arc from n_2 to 51, L2 P2 Var 41 47 53 59
Arc from n_73 to n_26, L2 P1 Var 60 69 78 87 96 105 114 123 132 141 150 159
Arc from n_26 to n_73, L2 P1 Var 61 70 79 88 97 106 115 124 133 142 151 160
Arc from n_26 to n_86, L2 P1 Var 62 71 80 89 98 107 116 125 134 143 152 161
Arc from n_26 to n_2, L2 P1 Var 63 72 81 90 99 108 117 126 135 144 153 162
Arc from n_27 to n_86, L2 P1 Var 64 73 82 91 100 109 118 127 136 145 154163
Arc from n_27 to n_2, L2 P1 Var 65 74 83 92 101 110 119 128 137 146 155 164
Arc from n_2 to n_26, L2 P1 Var 66 75 84 93 102 111 120 129 138 147 156 165
Arc from n_2 to n_86, L2 P1 Var 67 76 85 94 103 112 121 130 139 148 157 166
Arc from n_2 to n_27, L2 P1 Var 68 77 86 95 104 113 122 131 140 149 158 167
```

Ici commence l'heuristique.

```
Start regret heuristic
Entities to match :
```

Dès le début, les attributs rôle sont mis en correspondance entre eux. Le `1` *à la fin de la ligne ci-dessous indique le numéro de la variable fixée à 1. Ce numéro peut être retrouvé dans la description des noeuds du graphe bipartite ci-dessus.*

```
Role matched: 1
```

Ci-dessous suit la liste `L` *contenant l'ensemble des noeuds qui ne sont pas encore appariés. Après le ':' est indiqué le degré de chaque noeud.*

```
   Node Color_1 : 1
   Node 49 : 1
   Node Node_2 : 3
   Node Node_3 : 3
   Node Node_4 : 3
   Node Node_5 : 3
   Node 43 : 3
   Node 47 : 3
   Arc from Node_1 to Color_1 : 1
   Arc from Node_1 to 49 : 1
   Arc from Node_1 to Node_2 : 3
   Arc from Node_1 to Node_3 : 3
   Arc from Node_1 to Node_4 : 3
   Arc from Node_1 to Node_5 : 3
   Arc from Node_3 to Color_1 : 3
   Arc from Node_3 to 49 : 3
   Arc from Node_2 to Color_1 : 3
   Arc from Node_2 to 49 : 3
   Arc from Node_4 to Color_1 : 3
   Arc from Node_4 to 43 : 3
   Arc from Node_5 to Color_1 : 3
   Arc from Node_5 to 47 : 3
   Arc from Node_2 to Node_3 : 6
   Arc from Node_2 to Node_6 : 6
   Arc from Node_2 to Node_1 : 3
   Arc from Node_3 to Node_2 : 6
   Arc from Node_3 to Node_1 : 3
   Arc from Node_3 to Node_4 : 6
   Arc from Node_4 to Node_1 : 3
   Arc from Node_4 to Node_3 : 6
   Arc from Node_6 to Node_2 : 6
   Arc from Node_6 to Node_5 : 6
   Arc from Node_5 to Node_1 : 3
   Arc from Node_5 to Node_6 : 6
   Node col_6 : 1
   Node 48 : 1
   Node n_27 : 4
   Node n_26 : 4
   Node n_2 : 4
   Node 57 : 2
   Node 52 : 2
   Node 51 : 2
   Arc from n_86 to col_6 : 1
   Arc from n_86 to 48 : 1
   Arc from n_86 to n_27 : 4
   Arc from n_86 to n_26 : 4
   Arc from n_86 to n_2 : 4
   Arc from n_27 to col_6 : 4
   Arc from n_27 to 57 : 4
   Arc from n_26 to col_6 : 4
   Arc from n_26 to 52 : 4
   Arc from n_2 to col_6 : 4
   Arc from n_2 to 51 : 4
   Arc from n_73 to n_26 : 8
```

```
   Arc from n_26 to n_73 : 8
   Arc from n_26 to n_86 : 4
   Arc from n_26 to n_2 : 8
   Arc from n_27 to n_86 : 4
   Arc from n_27 to n_2 : 8
   Arc from n_2 to n_26 : 8
   Arc from n_2 to n_86 : 4
   Arc from n_2 to n_27 : 8
```

Ici sont choisies les mises en correspondance. Le `(1.00)` *indique qu'il s'agit de noeuds à degré 1.*

```
Select a match (1.00) :choose between value nodes
   There are 2 choices
    Regret for 2 is 2.00
   Generate match : 2
    Regret for 3 is 8.17
```

La variable d'appariement 3 est sélectionnée.

```
   Generate match : 3
```

Après avoir choisi un appariement entre attributs, il est aussi possible, sans commettre d'erreur, d'apparier des liens du même degré

```
  choose between value vertices
   There are 2 choices
    Regret for 23 is 2.04
    Regret for 22 is 2.00
   Generate match : 22
   Generate match : 23
```

La nouvelle liste L.

```
Remaining entities to match :
   Arc from n_73 to n_26 : 8
   Arc from n_26 to n_73 : 8
   Arc from n_26 to n_2 : 8
   Arc from n_27 to n_2 : 8
   Arc from n_2 to n_26 : 8
   Arc from n_2 to n_27 : 8
   Arc from Node_2 to Node_3 : 6
   Arc from Node_2 to Node_6 : 6
   Arc from Node_3 to Node_2 : 6
   Arc from Node_3 to Node_4 : 6
   Arc from Node_4 to Node_3 : 6
   Arc from Node_6 to Node_2 : 6
   Arc from Node_6 to Node_5 : 6
   Arc from Node_5 to Node_6 : 6
   Node n_27 : 4
   Node n_26 : 4
   Node n_2 : 4
   Arc from n_86 to n_27 : 4
   Arc from n_86 to n_26 : 4
   Arc from n_86 to n_2 : 4
   Arc from n_27 to col_6 : 4
   Arc from n_27 to 57 : 2
   Arc from n_26 to col_6 : 4
   Arc from n_26 to 52 : 2
   Arc from n_2 to col_6 : 4
   Arc from n_2 to 51 : 2
   Arc from n_26 to n_86 : 4
   Arc from n_27 to n_86 : 4
   Arc from n_2 to n_86 : 4
   Node Node_2 : 3
   Node Node_3 : 3
   Node Node_4 : 3
   Node Node_5 : 3
   Node 43 : 3
   Node 47 : 3
   Arc from Node_1 to Node_2 : 3
   Arc from Node_1 to Node_3 : 3
   Arc from Node_1 to Node_4 : 3
   Arc from Node_1 to Node_5 : 3
   Arc from Node_3 to Color_1 : 3
   Arc from Node_2 to Color_1 : 3
   Arc from Node_4 to Color_1 : 3
   Arc from Node_4 to 43 : 3
   Arc from Node_5 to Color_1 : 3
   Arc from Node_5 to 47 : 3
   Arc from Node_2 to Node_1 : 3
   Arc from Node_3 to Node_1 : 3
   Arc from Node_4 to Node_1 : 3
   Arc from Node_5 to Node_1 : 3
   Node 57 : 2
   Node 52 : 2
   Node 51 : 2
```

Choix d'une mise en correspondance

```
Select a match (2.00) :choose betwee
   There are 6 choices
    Regret for 16 is 10.60
    Regret for 17 is 9.67
    Regret for 18 is 9.49
    Regret for 19 is 9.70
    Regret for 20 is 8.85
    Regret for 21 is 8.68
   Generate match : 21
  choose between value vertices
   There are 3 choices
    Regret for 51 is 11.93
    Regret for 52 is 10.88
    Regret for 59 is 7.60
   Generate match : 59
```

La nouvelle liste L.

```
Remaining entities to match :
```

```
   Arc from n_73 to n_26 : 8
   Arc from n_26 to n_73 : 8
   Arc from n_26 to n_2 : 8
   Arc from n_27 to n_2 : 8
   Arc from n_2 to n_26 : 8
   Arc from n_2 to n_27 : 8
   Arc from Node_2 to Node_3 : 6
   Arc from Node_2 to Node_6 : 6
   Arc from Node_3 to Node_2 : 6
   Arc from Node_3 to Node_4 : 6
   Arc from Node_4 to Node_3 : 6
   Arc from Node_6 to Node_2 : 6
   Arc from Node_6 to Node_5 : 6
   Arc from Node_5 to Node_6 : 6
   Node n_27 : 3
   Node n_26 : 3
   Node n_2 : 1
   Arc from n_86 to n_27 : 4
   Arc from n_86 to n_26 : 4
   Arc from n_86 to n_2 : 4
   Arc from n_27 to col_6 : 4
   Arc from n_26 to col_6 : 4
   Arc from n_2 to col_6 : 4
   Arc from n_26 to n_86 : 4
   Arc from n_27 to n_86 : 4
   Arc from n_2 to n_86 : 4
   Node Node_2 : 2
   Node Node_3 : 2
   Node Node_4 : 2
   Node Node_5 : 1
   Node 43 : 2
   Arc from Node_1 to Node_2 : 3
   Arc from Node_1 to Node_3 : 3
   Arc from Node_1 to Node_4 : 3
   Arc from Node_1 to Node_5 : 3
   Arc from Node_3 to Color_1 : 3
   Arc from Node_2 to Color_1 : 3
   Arc from Node_4 to Color_1 : 3
   Arc from Node_4 to 43 : 2
   Arc from Node_5 to Color_1 : 3
   Arc from Node_2 to Node_1 : 3
   Arc from Node_3 to Node_1 : 3
   Arc from Node_4 to Node_1 : 3
   Arc from Node_5 to Node_1 : 3
   Arc from n_27 to 57 : 1
   Arc from n_26 to 52 : 1
   Node 57 : 1
   Node 52 : 1
```

Choix d'une mise en correspondance

```
Select a match (1.00) :choose between object nodes
   There are 1 choices
   Generate match : 15
  choose between value nodes
   There are 2 choices
    Regret for 16 is 5.30
    Regret for 17 is 4.84
   Generate match : 17
```

La nouvelle liste L.

```
Remaining entities to match :
   Arc from n_73 to n_26 : 6
   Arc from n_26 to n_73 : 6
   Arc from n_26 to n_2 : 1
   Arc from n_27 to n_2 : 1
   Arc from n_2 to n_26 : 1
   Arc from n_2 to n_27 : 1
   Arc from Node_2 to Node_3 : 2
   Arc from Node_2 to Node_6 : 2
   Arc from Node_3 to Node_2 : 2
   Arc from Node_3 to Node_4 : 2
   Arc from Node_4 to Node_3 : 2
   Arc from Node_6 to Node_2 : 2
   Arc from Node_6 to Node_5 : 2
   Arc from Node_5 to Node_6 : 2
   Arc from n_86 to n_27 : 3
   Arc from n_86 to n_26 : 3
   Arc from n_86 to n_2 : 1
   Arc from n_27 to col_6 : 3
   Arc from n_26 to col_6 : 3
   Arc from n_2 to col_6 : 1
   Arc from n_26 to n_86 : 3
   Arc from n_27 to n_86 : 3
   Arc from n_2 to n_86 : 1
   Node n_27 : 3
   Node n_26 : 3
   Arc from Node_1 to Node_2 : 2
   Arc from Node_1 to Node_3 : 2
   Arc from Node_1 to Node_4 : 2
   Arc from Node_1 to Node_5 : 1
   Arc from Node_3 to Color_1 : 2
   Arc from Node_2 to Color_1 : 2
   Arc from Node_4 to Color_1 : 2
   Arc from Node_5 to Color_1 : 1
   Arc from Node_2 to Node_1 : 2
   Arc from Node_3 to Node_1 : 2
   Arc from Node_4 to Node_1 : 2
   Arc from Node_5 to Node_1 : 1
   Node Node_2 : 2
   Node Node_3 : 2
   Node Node_4 : 2
   Arc from Node_4 to 43 : 1
   Arc from n_26 to 52 : 1
```

Choix d'une mise en correspondance

```
Select a match (1.00) :choose between object vertices
   There are 6 choices
    Regret for 165 is 5.00
    Regret for 167 is 5.00
    Regret for 144 is 5.00
    Regret for 146 is 5.00
    Regret for 35 is 2.00
   Generate match : 35
    Regret for 157 is 2.00
   Generate match : 157
   Generate match : 165
```

La nouvelle liste L.

```
Remaining entities to match :
   Arc from n_73 to n_26 : 6
   Arc from n_26 to n_73 : 6
   Arc from n_86 to n_27 : 3
   Arc from n_86 to n_26 : 3
   Arc from n_27 to col_6 : 3
   Arc from n_26 to col_6 : 3
   Arc from n_26 to n_86 : 3
   Arc from n_27 to n_86 : 3
   Node n_27 : 3
   Arc from Node_2 to Node_3 : 2
   Arc from Node_2 to Node_6 : 2
   Arc from Node_3 to Node_2 : 2
   Arc from Node_3 to Node_4 : 2
   Arc from Node_4 to Node_3 : 2
   Arc from Node_6 to Node_2 : 2
   Arc from Node_6 to Node_5 : 2
   Arc from Node_1 to Node_2 : 2
   Arc from Node_1 to Node_3 : 2
   Arc from Node_1 to Node_4 : 2
   Arc from Node_3 to Color_1 : 2
   Arc from Node_2 to Color_1 : 2
   Arc from Node_4 to Color_1 : 2
   Arc from Node_2 to Node_1 : 2
   Arc from Node_3 to Node_1 : 2
   Arc from Node_4 to Node_1 : 2
   Node Node_2 : 1
   Node Node_3 : 1
   Node Node_4 : 1
   Arc from n_26 to n_2 : 1
   Arc from n_27 to n_2 : 1
   Arc from n_2 to col_6 : 1
   Arc from Node_5 to Color_1 : 1
   Arc from Node_4 to 43 : 1
   Arc from n_26 to 52 : 1
```

Choix d'une mise en correspondance

```
Select a match (1.00) :choose between object nodes
   There are 3 choices
    Regret for 4 is 22.00
    Regret for 7 is 22.00
    Regret for 10 is 19.00
   Generate match : 10
  choose between object vertices
   There are 1 choices
   Generate match : 144
```

La nouvelle liste L.

```
Remaining entities to match :
   Arc from n_73 to n_26 : 4
   Arc from n_26 to n_73 : 4
   Arc from n_86 to n_27 : 1
   Arc from n_86 to n_26 : 2
   Arc from n_27 to col_6 : 1
   Arc from n_26 to col_6 : 2
   Arc from n_26 to n_86 : 2
   Arc from n_27 to n_86 : 1
   Arc from Node_2 to Node_3 : 2
   Arc from Node_2 to Node_6 : 2
   Arc from Node_3 to Node_2 : 2
   Arc from Node_6 to Node_2 : 2
   Arc from Node_1 to Node_2 : 1
   Arc from Node_1 to Node_3 : 1
   Arc from Node_1 to Node_4 : 1
   Arc from Node_3 to Color_1 : 1
   Arc from Node_2 to Color_1 : 1
   Arc from Node_4 to Color_1 : 1
   Arc from Node_2 to Node_1 : 1
   Arc from Node_3 to Node_1 : 1
   Arc from Node_4 to Node_1 : 1
   Arc from n_2 to col_6 : 1
   Arc from Node_5 to Color_1 : 1
```

A ce stade, l'algorithme s'arrête, car il n'y des liens. Continuer l'appariement n'appo fonction GloutonSimilarité.

Le retour de la fonction RegretMatch

```
Matchings : 1, 2, 3, 22, 23, 21, 59,
```

Annexe C. Guide de l'utilisateur de *Alois*

Le système *Alois* se repose sur le moteur d'inférence *Clips 5.1* [Riley 1991, Giarrantano & Riley 1994] pour tout ce qui relève de la manipulation des schémas d'opérateur, de la détermination des opérateurs applicables et de la représentation et modification des états de recherche. Le langage de représentation des problèmes est une extension du langage employé par *Clips*. *Clips* est librement disponible[1]. Ce *guide de l'utilisateur* décrit les fonctionnalités complémentaires de *Alois* par rapport à la version 5.1, avec lequel l'utilisateur doit être familier.

Options de la ligne de commande de Alois

```
Chaque options ne peut être présent qu'une seule fois. Leur ordre est libre

-f 'filename'
Similaire à Clips 5.1. Charge est exécute le fichier batct 'filename' avant d'accepter
des commandes de l'utilisateur.

-ident 'AnyString'
'AnyString' est un identificateur de l'instance de problème que l'on optimise. Cette
chaine de caractères libre va être affiché au début de la trace.
La variable '?*IdentString*' = 'AnyString' pendant l'exécution.

-imp 'number'
Lorsque Alois a effectué 'number' d'itération sans amélioration de la foction objectif,
l'exécution est arrrêté.

-match [heuristic|bipartite|regret]
Selectionne la technique d'appariement
   'heuristic" >> GloutonSimilarité (par défaut)
   'bipartite' >> appariement bi-partite (requiert LP_Solve!)
   'regret'    >> RegretMatch

-p 'integer'
Affecte '?*ProblemInstanceSize*' = 'integer'.
Utilise la commande Clips 'defglobal'. L'affectation est fait avant l'exécution d'un
fichier batch.

-pbinstance 'filename'
Charge le fichier 'filename' qui doit contenir exclusivement des comamndes Clips .
Equivalent à la fonction LoadConstructs() ou '(Load )' de Clips.

-r 'number'
'number' est le nombre mamaximal d'itération après une commande (run). La valeur par
défault est 20000.

-seed 'integer'
Initialise le générateur de nombre aléatoires avec 'integer'. Clips untilise un
générateur conforme à la norme ANSI. L'initialisation par défault est 1.

-time
Indique a Alois d'initialiser l'horloge intèrne afin de pouvoir mesurer (en ms) le temps
ecoulé. Attention, Alois mesure le temps réel, et non pas le temps CPU.

-trace 'filename'
Après une commande (quit) la trace va être sauvegardée dans le fichier 'filename'. Ce
fichier ne doit pas exister avant l'exécution de Alois.
```

Définition des schémas d'opérateur, des littéraux et des attributs

```
(defliteral name #objects #values [poids])
```

---

[1] http://www.jac.nasa.gov/~clips/CLIPS.html

Définition d'un littéral
name : le nom du littéral
#objects : le nombre d'attributs objet
#values : le nombre d'attributs valeur
poids : le poids du littéral dans la mesure de similarité

`(defmvliteral name #objects #values [poids])`
Définition d'un littéral multivalué
name : le nom du littéral
#objects : le nombre d'attributs objet
#values : le nombre d'attributs valeur
poids : le poids du littéral dans la mesure de similarité

`(defattrclass name type)`
Définition d'une classe d'attributs
name : le nom de la classe des attributs
type : le type des attributs de cette classe (Object, Set, Real ou Integer)

`(defattr attributeClassName attr [attr])`
Définition des attributs d'une classe (pas nécessaire pour les attributs valeur)
attributeClassName : nom de la classe
attr : identificateur d'un attribut

`(defposition literalName attributeClassName position [filter])`
Définition des types par position dans un littéral
literalName :nom du littéral
attributeClassName : nom de la classe de l'attribut
position : numéro de la position de l'attribut
filter : condition que doit satisfaire l'attribut en question pour que le littéral soit retenu dans un contexte d'instanciation (voir exemple UFLP)

`(defopschema schema [schema])`
Définition des schémas d'opérateur pour lesquelles le système apprends. Le schéma lui-même doit être défini par la commande Clips defrule.
schéma : nom du schéma

`(functionalrole oneObjectAttribute)`
Défini les attributs qui peuvent être des rôles dans le contexte d'instanciation
oneObjectAttribute : identificateur de l'attribut

`(verifydomain)`
Procède à un teste de complétude afin de vérifier si des atrtibuts sont définis pour l'ensemble des positions des littéraux.

Comportement standard de Clips

L'exploration de l'espace de recherche, par le moteur d'inférence, est figé de sorte à ce qu'elle porduit le comportement établi par les quatre commandes suivantes :

```
   SetStrategy(BREADTH_STRATEGY);
   SetFactDuplication(CLIPS_FALSE);
   SetSalienceEvaluation(WHEN_ACTIVATED);
   SetConserveMemory(CLIPS_FALSE);
```

En outre, des variables globales sont par défault ajoutés a Clips. Il s'agit de :
?*objective* : valeur de la fonction objective
?*ProblemInstanceSize* : taille de l'insatnce de problème
?*IdentString* : chaine de caractères permettant l'identification de l'instance
?*SeedGrain* : grain d'initialisation du générateur aléatoire

Dans chaque état, le système doit déterminer la valeur de la fonction objectif, et doit affecter cette valuer à la variable ?*objective*.
Lorsque la recherche atteint un état réalisable, le système doit ensuite construire le littéral (control feasibleState) afin de dire à *Alois* qu'un état réalisable soit atteint.

Commandes de Alois

`(run [n])`
Démarre l'optimisation, 'n' indique le nombre d'itérations à faire.

`(quit)`
Quitte Alois, arrête l'horloge et sauvegarde la trace si fichier avait été spécifié auparavent.

```
(aloiswatch or
(aloisunwatch errors|literals|cases|operators|matching|similarities|statistics|
              opschema|iterations|adaptation|library|outcome|masterslave|
              regretmatch)
```

Permets d'observer divers niveau de trace. L'option 'error' doit être avtivé afin que les autres options soyent disponibles.

```
(caselib acquire|reuse|none|acquire_and_reuse)
Permets de choisir le mode dans lequel le CBR travaille.

(interface minimal)
Réduit l'interface au strcte minimum. Permets une accélération considérable au cas où
l'on utilise l'interface graphique. Sans effet pour la version sans interface. La
commande doit être donné avant toute commande 'defrule', 'reset', ou 'assert'.

(reward best|brute)
Selectione le type de récompense :
   best  >> pondère la récompense par la similarité
   brute >> prends la valeur maximale de la récompense (2 pour bonus, 0.5 pour malus)

(timer start|stop|restart|getelapsed)
Commandes relatives à l'horloge
   start      >> démarre
   stop       >> arrête
   restart    >> arrête puis redémarre
   getelapsed >> indique le temps écoulé depuis le dernier start


(caselibio print|printstructure)
Affiche le contenu de la mémoire de cas
   print          >> affiche tout le contenu
   printstructure >> affiche l'arbre de prototypes dynamique


(caselibtype 'name' [type])
Spécifie la mémoire de cas
   name = list, templatetree
          liste plate ou arbre de prototypes
   type = standard, variable
          arbre de prototypes classique ou dynamique

(ttconf retrieveRedundancy insertRedundancy intraDivergence maxDegree)
Pour un arbre de prototypes dynamiques, permets de spécifier les paramètres auxiliairs.

(masterslavevalidate on|off random|guided)
Démarre la valdation de la récupération du type "maître / esclave" La mémoire mastère
fonctionne comme d'habitude, et la mémoire slave est testé par rapport au résultat de la
récupération dans le master.
   random >> la résolution de conflicts selectionne les opérateurs de manière aléatoire
   guided >> fonctionnement normal de la résolution de conflicts

(slavelibtype 'name' [type])
Spécifie la mémoire de cas esclave, corresponds à caselibtype

(slavettconf retrieveRedundancy insertRedundancy intraDivergence maxDegree
Spécifie les paramètres de la mémoire esclave, corresponds à ttconf

(agendasize)
Indique le nombre d'opérateurs présent dans l'agenda

(aloisrandom)
Fourni une valeur aléatoire

(aloisrandommaximum)
Indique le valeur maximale du générateur de nombre aléatoires. Cette valeur peut varier
selon le compilateur et selon le système d'exploitation.

(aloishalt)
Arrête l'exécution de Alois depuis une règle ou une fonction. Ne jamais utiliser (halt)
dde Clips.

(stringtoindex 'offset' 'string')
Transforme une chaine de caractères en un index. En lève les premiers 'offset'
caractères de la chaine 'string'
   Exemple : (stringtoindex 2 "n_167") donne 167
```

Index

Résumé

Sous le nom de méthode tabou, on regroupe une famille d'algorithmes qui ont pour but de résoudre des problèmes d'optimisation combinatoire, donc des problèmes NP-difficiles. Basées sur la recherche locale, ces techniques construisent des solutions sous-optimales, mais qui sont en général de bonne qualité. La méthode tabou a fait appel à des techniques d'apprentissage pour améliorer le processus de construction et d'identification de solutions, afin de parvenir aux bonnes solutions de manière plus efficace, et afin de montrer des performances plus robustes. Or, ces techniques d'apprentissage doivent être mises au point de manière spécifique à chaque problème d'optimisation, nécessitant chaque fois une bonne compréhension de la résolution de ce dernier.

Cette thèse s'inscrit dans la voie de recherche qui étudie la construction, pour la méthode tabou, de techniques d'apprentissage qui sont indépendant du problème d'optimisation auquel elles s'appliquent. Nous proposons une technique d'apprentissage qui est basée sur le raisonnement à partir de cas. Un cas étant relatif à l'application d'une transformation d'une solution, il est composé de : a) le contexte dans lequel la transformation a été appliquée, et b) la récompense qualitative attribuée aux effets de cette transformation. Les futurs transformations sont alors examiné de manière approfondie en prenons en considération les récompenses contenues dans les cas antérieurs. L'approche est évaluée sur trois des problèmes classiques d'optimisation combinatoire.

Abstract

Tabu search is a meta-heuristic that is able to solve various NP-hard combinatorial optimisation problems. A variant of classical local search, tabu search constructs high quality, near to optimal solutions. In order to speed up the solution generation process, as well as to make its results more robust, tabu search employs particular learning techniques whilst optimisation is in progress. However, these learning schemes have to be specially adapted to each optimisation problem at hand.

This thesis studies the construction of a learning technique for tabu search which is problem independent. We introduce an algorithm that uses case-based reasoning. A case is constructed from the application of a transformation to a solution state. It has two parts, a description of the context in which the transformation applies, and a qualitative reward credited to it with regard to the results of the transformations application. Using this information available in the cases, the ongoing transformations are more deeply analysed with respect to available reward. The fully implemented system is evaluated on three classical combinatorial optimisation problems.