Planification de coût optimal basée sur les CSP pondérés

Marie de Roquemaurel

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*
Discipline ou spécialité : *Informatique*

Présentée et soutenue par *Marie DE ROQUEMAUREL*
Le *12 Mars 2009*

Titre : *Planification de coût optimal basée sur les CSP pondérés*

JURY :

Mr Martin COOPER	Professeur - Université de Toulouse III	Président
Mr Eric JACOPIN	Maître de conférences HDR – Ecoles de Saint-Cyr	Rapporteur
Mr Philippe JEGOU	Professeur – Université d'Aix-Marseille	Membre
Mr Cyril PAIN-BARRE	Maître de conférences – Université de la Méditerranée	Membre
Mr Pierre REGNIER	Maître de conférences HDR – Université de Toulouse III	Membre
Mr Frank VELTMAN	Professeur – Université d'Amsterdam	Membre

Ecole doctorale : *MITT*
Unité de recherche : *Institut de Recherche en Informatique de Toulouse*
Directeur de Thèse : *Mr Pierre REGNIER*
Encadrant de Thèse : *Mr Martin COOPER*
Rapporteur : *Mme Sylvie THIEBAUX*

Marie de ROQUEMAUREL

Planification de coût optimal basée sur les CSP pondérés

Directeur de thèse : Pierre REGNIER, Maître de conférences, HdR
Lieu et date de soutenance : Université Toulouse III - 12 Mars 2009
Discipline Administrative : Informatique

Résumé

Un des challenges actuels de la planification est la résolution de problèmes pour lesquels on cherche à optimiser la qualité d'une solution telle que le coût d'un plan-solution. Dans cette thèse, nous développons une méthode originale pour la planification de coût optimal dans un cadre classique non temporel et avec des actions valuées.

Pour cela, nous utilisons une structure contenant tous les plans d'une longueur donnée, appelé graphe de planification. L'extraction d'une solution optimale, à partir d'un graphe de planification de longueur fixée, est codée comme un problème de satisfaction de contraintes pondérées (WCSP). La structure spécifique des WCSP obtenus permet aux solveurs actuels de trouver, pour une longueur donnée, une solution optimale dans un graphe de planification contenant plusieurs centaines de nœuds.

Nous présentons ensuite plusieurs méthodes pour déterminer la longueur maximale des graphes de planification nécessaire pour garantir l'obtention d'une solution de coût optimal. Ces méthodes incluent plusieurs notions universelles comme par exemple la notion d'ensembles d'actions indispensables pour lesquels toutes les solutions contiennent au moins une action de l'ensemble. Ces ensembles peuvent être rapidement détectés en résolvant une version relaxée du problème original. Les résultats expérimentaux effectués montrent que l'utilisation de ces méthodes permet une diminution de 60% en moyenne de la longueur requise pour garantir l'obtention d'une solution de coût optimal.

La comparaison expérimentale avec d'autres planificateurs montre que l'utilisation du graphe de planification et des CSP pondérés pour la planification optimale est possible en pratique même si elle n'est pas compétitive, en terme de temps de calcul, avec les planificateurs optimaux récents.

Mots-clés

Planification avec actions valuées, Optimalité en coût, Graphe de planification, CSP pondérés

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Toulouse III - Paul Sabatier, 118 route de Narbonne
31062 TOULOUSE cedex 9

Marie de ROQUEMAUREL

A weighted CSP approach to cost-optimal planning

Supervisor : Pierre REGNIER, Maître de conférences, HdR
Administrative discipline : Computer Science

Abstract

For planning to come of age, plans must be judged by a measure of quality, such as the total cost of actions. This thesis describes an optimal-cost planner in the classical planning framework except that each action has a cost.

We code the extraction of an optimal plan, from a planning graph with a fixed number k of levels, as a weighted constraint satisfaction problem (WCSP). The specific structure of the resulting WCSP means that a state-of-the-art exhaustive solver was able to find an optimal plan in planning graphs containing several thousand nodes.

We present several methods for determining a tight bound on the number of planning-graph levels required to ensure finding a globally optimal plan. These include universal notions such as indispensable sets S of actions : every valid plan contains at least one action in S. Different types of indispensable sets can be rapidly detected by solving relaxed planning problems related to the original problem. On extensive trials on benchmark problems, the bound on the number of planning-graph levels was reduced by an average of 60% allowing us to solve many instances to optimality.

Thorough experimental investigations demonstrated that using the planning graph in optimal planning is a practical possibility, although not competitive, in terms of computation time, with a recent state-of-the-art optimal planner.

Keywords

Optimal planning, planning graph, soft constraints

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Toulouse III - Paul Sabatier, 118 route de Narbonne. 31062 TOULOUSE cedex 9

Remerciements

Je tiens à remercier en tout premier lieu Mr Pierre REGNIER qui a dirigé cette thèse après m'avoir permis de découvrir la planification en encadrant mon stage de DEA. Tout au long de ces années, il a suivi mon travail de manière méticuleuse et a su orienter mes recherches aux bons moments. Pour sa disponibilité et ses précieux conseils, pour sa confiance et pour m'avoir permis d'en être là aujourd'hui, je le remercie du fond du coeur.

Je tiens de même à remercier Mr Martin COOPER qui a encadré cette thèse. En partageant les avancées des techniques des CSP pondérées et en cherchant à les utiliser pour résoudre des problèmes de planification, il a apporté de nombreuses discussions, de nouvelles directions de recherche qui ont initié puis enrichi cette thèse.

Un très grand merci aux rapporteurs Mme Sylvie THIEBAUX et Mr Eric JACOPIN pour l'intérêt qu'ils ont montré envers ces travaux. Leurs rapports très positifs et encourageants ainsi que leurs remarques toujours constructives m'ont permis d'améliorer grandement ce manuscrit et d'élargir mes recherches.

Je tiens également à remercier les membres du jury : Mr Philippe JEGOU, Mr Cyril PAIN-BARRE et Mr Frank VELTMAN d'avoir examiné ce manuscrit et pour l'honneur qu'ils m'ont faits en participant à ce jury.

Cette thèse n'aurait pas pu voir le jour sans l'aide de ceux qui m'ont fait confiance et qui m'ont aidé tout au long de cette thèse. Je remercie l'équipe RPDMP de m'avoir accueilli et de m'avoir proposé un cadre de travail très agréable. Je remercie également ceux qui m'ont permis d'obtenir une allocation MENRT puis un poste d'ATER. Je penses également aux aides reçues du personnel admnistratif, notamment Martine dont la gentillesse et la disponibilité permettent de venir à bout de toutes les démarches administratives.

Enfin, je remercie tous ceux qui, d'une manière ou d'une autre, ont permis l'aboutissement de cette thèse. Je pense ainsi à ma famille qui a toujours cherché à comprendre mon travail et à m'encourager dans les moments difficiles. Je remercie également mes amis qui m'ont permis d'oublier de temps en temps la thèse tout en acceptant mes absences régulières.

Sommaire

Chapitre 1

Introduction

1.1 Introduction au domaine

L'Intelligence Artificielle (IA) cherche à créer des machines autonomes et capables de simuler des comportements intelligents. Pour cela, l'approche la plus courante utilise une boucle Perception-Décision-Action. Dans ce cadre, de nombreux problèmes ont été étudiés comme la perception du monde extérieur, l'acquisition des connaissances, la représentation d'informations, la modélisation d'une réalité virtuelle et l'interaction sur l'environnement. Les chercheurs se sont également intéressés à formaliser et automatiser différents types de raisonnements ainsi qu'à résoudre de nombreux problèmes combinatoires.

Pour permettre à un système autonome d'agir efficacement dans son environnement, il est ainsi nécessaire de le doter d'une capacité de planification qui lui permette, à partir d'une situation initiale, de générer automatiquement un ensemble structuré d'actions permettant d'atteindre un but préalablement fixé (plan-solution). Comme le monde réel est bien trop complexe pour être pratiquement représenté (il est dynamique, partiellement observable, incertain, non déterministe, etc.), on se place généralement dans un cadre plus restrictif (dit cadre classique) pour aboutir à des représentations plus simple et à des traitements plus efficaces. On cherche ainsi à développer des techniques utilisables en pratique dans un environnement très contraint pour ensuite essayer de les étendre à un cadre plus proche de problèmes réels.

Dans cette thèse, nous nous plaçerons dans ce cadre classique en posant en particulier les hypothèses suivantes :

- l'environnement est statique : les seuls changements du monde sont ceux effectués par l'agent, qui évolue seul.
- L'agent est omniscient : il a une connaissance complète et instantanée de l'environnement dans lequel il évolue et de la nature de ses propres actions.
- Les actions sont atomiques : l'exécution d'une action par un agent est ininterruptible et indivisible. Elle est modélisée comme une transition atomique d'un état du monde vers un autre.
- Les actions sont déterministes : l'exécution d'une action par l'agent est une fonction déterministe de cette action et de l'état du monde.

Dans ce cadre, on cherche à développer un système, appelé planificateur, qui prenne en entrée

un problème de planification comprenant la description :

- des différentes actions possibles,
- de l'état initial du monde considéré,
- et du but à obtenir.

Avec ces données en entrée, le planificateur doit synthétiser un plan-solution.

Pour pouvoir représenter ces problèmes d'une manière indépendante du domaine, plusieurs langages ont été successivement développés. Le planificateur STRIPS (STanford Research Institute Problem Solver) [Fikes et Nilsson, 1971] a été le premier à proposer un tel langage. Dans celui-ci :

- un état du monde est défini comme une conjonction de littéraux propositionnels (appelés fluents). Les seuls fluents représentés dans un état sont ceux qui sont vrais et Fikes et Nilsson font également l'hypothèse du monde clos : tous les fluents qui ne sont pas vrais dans un état sont considérés comme faux.
- Un but est un état (généralement partiel) composé d'une conjonction de fluents. Un état satisfait un but ssi tous les fluents du but appartiennent à cet état.
- Une action est représentée par des préconditions (ensemble de fluents devant être satisfaits dans l'état dans lequel l'action doit être appliquée), des ajouts (ensemble de fluents ajoutés à l'état résultant de l'application de l'action) et des retraits (ensemble de fluents retirés de l'état résultant de l'application de l'action). Pour éviter d'avoir à représenter explicitement tous les fluents non modifiés lors de l'exécution d'une action (problème du décor, frame problem), Fikes et Nilsson imposent que chaque fluent non mentionné dans les effets de l'action demeure inchangé par son exécution.

Le langage ADL (Action Description Language) [Pednault, 1989] a ensuite étendu celui du planificateur STRIPS en autorisant la prise en compte des négations, des disjonctions ou de quantificateurs existentiels et universels.

Enfin, pour homogénéiser les différents langages existants, le langage PDDL (Planning Domain Definition Language) a été adopté par la communauté. Depuis sa première version [McDermott *et al.*, 1998], il a évolué et propose maintenant différents niveaux d'expressivité qui permettent de représenter le temps, des variables numériques ou des préférences [Gerevini et Long, 2006]. Ce langage permet de représenter des domaines très différents comme des problèmes de logistique, de transport, de gestion d'aéroport, ou de commande de satellites, de pipelines, etc.

Pour évaluer le plus objectivement possible les progrès effectués en planification, une compétition internationale bi-annuelle (IPC[1]) est organisée dans le cadre de la conférence ICAPS (International Conference on Planning and Scheduling). Elle permet d'analyser et de comparer les approches actuelles, de faire évoluer les différentes techniques et de mettre en avant de nouveaux axes de recherches. Elle permet également de mettre au point de nouveaux jeux de tests (benchmarks) qui se rapprochent de plus en plus de problèmes réels.

[1] http ://ipc.informatik.uni-freiburg.de

1.2 Cadre de travail

A l'heure actuelle les algorithmes qui ont été développés dans le cadre classique permettent d'obtenir rapidement (moins de 30 minutes pour les dernières compétitions IPC) un plan-solution pour des problèmes de taille importante. Lors de la compétition IPC de 2006, plusieurs planificateurs ont ainsi réussi à résoudre le plus gros problème proposé qui modélise des déplacements de marchandises et leurs achats dans différents marchés (domaine "Travelling and Purchase"). Ce problème se compose de 8 marchés et 3 dépots reliés de manière aléatoire. Huit camions peuvent se déplacer, charger ou décharger simultanément un exemplaire d'un produit parmi vingt qui sont répartis aléatoirement dans les marchés. Le but est d'acheter un certain nombre de ces produits. Le planificateur SGPLAN [Chen *et al.*, 2004], vainqueur de cette compétition, trouve ainsi en moins de 17 minutes un plan-solution contenant près de 400 actions.

Les progrès qui sont donc réalisés dans le cadre classique ainsi que les performances de plus en plus élevées des machines, permettent maintenant d'envisager le traitement de problèmes de planification plus complexes. Il devient par exemple possible d'imposer un critère de sélection sur les plans-solutions pour chercher à obtenir non plus un plan-solution quelconque mais un plan optimal selon un critère donné. Différents critères d'optimalité peuvent ainsi être considérés et éventuellement combinés :

- longueur des plans-solutions en nombre d'actions,
- longueur des plans-solutions en nombre d'étapes (actions "simultanées"),
- temps total d'exécution du plan-solution,
- coût total des actions du plan-solution.

Selon le point de vue considéré, l'un ou l'autre de ces critères peut alors s'avérer plus pertinent. Dans l'exemple précédent, si toutes les actions ont un coût et un temps d'exécution identiques, un logisticien préférera une solution qui minimise le nombre d'actions alors qu'un acheteur qui doit respecter un délai de livraison préfèrera une optimisation de la longueur des solutions. Si les actions ont maintenant des durées différentes, l'acheteur peut préférer une solution de temps total d'exécution minimal. Si le coût des actions représente un coût monétaire ou une consommation de ressources, l'acheteur peut préférer une solution qui minimise le coût total de la solution. On peut enfin vouloir combiner ces critères d'optimalité pour, par exemple, chercher une solution de coût optimal parmi les solutions de longueur minimale.

Pour optimiser le coût d'un plan-solution, deux types essentiels d'approches sont actuellement développées. Les premières cherchent à minimiser le coût des solutions parmi celles de longueur optimale. Les autres essayent de produire une solution de coût global sans fixer de borne à la longueur des plans. Dans cette thèse, nous présentons des algorithmes de planification originaux pour ces deux types d'approches.

Planificateurs de coût optimal pour un nombre d'étapes donné

Dans le cadre atemporel où toutes les actions ont une durée d'exécution unitaire, on appelle étape un ensemble d'actions susceptibles de s'exécuter "simultanément" ou dans n'importe quel ordre sans que cela change le résultat de l'exécution du plan.

Pour obtenir une solution optimale en nombre d'étapes, certaines approches (dites "par compilation") compilent le problème de planification en un problème SAT (satisfiabilité d'une base de

clauses), un programme linéaire sur les entiers (IP) ou un problème de satisfaction de contraintes (CSP). Elles procèdent généralement de manière incrémentale en cherchant une solution pour un nombre d'étapes fixé et en incrémentant ce nombre jusqu'à l'obtention d'un plan-solution. Les méthodes d'extraction propres à ces techniques (SAT, IP, CSP...) permettent souvent d'optimiser un second critère pour obtenir une solution de coût optimal pour une nombre d'étapes donné. Pour des coûts uniformes, on peut ainsi minimiser le nombre d'actions contenues dans des solutions optimales en nombre d'étapes [Rintanen, 1998 ; Bylander, 1997]. Des approches similaires [Chen *et al.*, 2008b] considèrent des actions valuées et permettent de minimiser le coût du plan-solution. [Miguel *et al.*, 2000 ; Brafman et Chernyavsky, 2005] associent des préférences ou des degrés de satisfaction aux buts et aux actions pour obtenir une solution optimale en terme de gain ou de satisfaction. [van den Briel *et al.*, 2004b ; Do *et al.*, 2007 ; Giunchiglia et Maratea, 2007b] associent des gains à la réalisation de buts tout en conservant des coûts sur les actions et montrent comment obtenir une solution optimale en terme de bénéfices.

Planificateurs de coût optimal

Pour obtenir une solution de coût optimal, les approches par recherche heuristique apportent, à l'heure actuelle, les techniques les plus efficaces. Lors de la dernière compétition IPC'2008, huit planificateurs ont ainsi concouru dans la catégorie "sequential optimization track". Ils sont, en général, basés sur un algorithme de recherche optimal (A* ou IDA*) guidé par une heuristique admissible. Pour des coûts uniformes, [Haslum et Geffner, 2000 ; Zhou et Hansen, 2006 ; Helmert *et al.*, 2007 ; van den Briel *et al.*, 2007] montrent comment obtenir une solution optimale en nombres d'actions. Des approches similaires [Haslum, 2006 ; Coles *et al.*, 2008 ; Edelkamp, 2003 ; Haslum *et al.*, 2005 ; Helmert *et al.*, 2008 ; Katz et Domshlak, 2008b ; Robinson *et al.*, 2008a] considèrent des actions valuées et permettent de minimiser le coût des solutions. [Benton *et al.*, 2007a,b ; Edelkamp et Kissmann, 2008b] associent des gains à la réalisation de buts tout en conservant des coûts sur les actions et montrent comment obtenir une solution optimale en terme de bénéfices.

1.3 Objectifs

Dans cette thèse, nous avons développé une méthode originale pour la planification de coût optimal dans un cadre classique non temporel et avec des actions valuées. Pour cela, nous avons d'abord cherché à obtenir une solution de coût optimal parmi les solutions optimales en nombre d'étapes. Pour augmenter l'efficacité de la résolution de ces problèmes, nous avons ensuite étudié différentes méthodes d'analyse des domaines et des problèmes. Nous avons enfin étendu nos méthodes pour obtenir une solution de coût optimal (indépendamment du nombre d'étapes).

Pour atteindre ces objectifs, nos algorithmes utilisent le graphe de planification introduit par le planificateur GRAPHPLAN [Blum et Furst, 1997]. Cette structure possède l'avantage de contenir tous les plans d'une longueur donnée qui peuvent y être représentés de manière très efficace grâce aux améliorations proposées dans STAN [Long et Fox, 1999].

Pour extraire efficacement une solution de ce graphe, nous avons montré comment le coder en un WCSP (problème de satisfaction de contraintes pondérés). Dans le cadre atemporel classique,

l'utilisation de techniques CSP pour obtenir une première solution a déjà montré de bonnes performances [Do et Kambhampati, 2001] mais les extensions de ces travaux se limitent à l'emploi de valeurs numériques uniquement pour les conditions ou les effets des actions [Lopez et Bacchus, 2003 ; Laborie, 2003 ; Srivastava, 2000 ; Srivastava et Kambhampati, 1999]. De plus, elles ne considèrent que l'optimalité en nombre d'étapes. Grâce à l'utilisation de contraintes valuées, les WCSP offrent un cadre rigoureux et des algorithmes performants pour rechercher des solutions de coût optimal dans des problèmes de planification avec des actions valuées.

Par ailleurs, bien qu'il existe de nombreux travaux proposant des méthodes pour utiliser des informations contenues de manière implicite dans les domaines et les problèmes de planification, nous avons mis en évidence le rôle de certaines actions. Cette approche originale améliore notre méthode et reste suffisamment générale pour être utilisée dans d'autres types d'algorithmes.

1.4 Structure du manuscrit

Nous présentons le contexte de cette thèse dans les deux chapitres suivants. Le chapitre 2 présente un état de l'art des différentes approches de planification qui ont été développées pour obtenir une solution de coût optimal. Nous verrons ainsi qu'il existe essentiellement deux types de méthodes qui permettent, soit d'obtenir une solution de coût optimal pour un nombre d'étapes donné, soit d'obtenir une solution de coût optimal indépendamment du nombre d'étapes. L'approche que nous proposons utilise le cadre des WCSP pour résoudre des problèmes de planification dans un cadre classique avec actions valuées. Ce formalisme offre un cadre rigoureux et des algorithmes performants pour résoudre efficacement des problèmes d'optimisation combinés à des fonctions de coûts locales. Nous le présentons dans le chapitre 3.

Les autres chapitres présentent le travail effectué au cours de cette thèse. Le chapitre 4 décrit une méthode originale pour garantir l'obtention d'une solution de coût optimal pour un nombre d'étapes minimum. Cette méthode, implémentée dans le planificateur GP-WCSP, est basée sur le codage d'un graphe de planification en un WCSP. La résolution optimale de ce WCSP est rendue plus efficace grâce à l'utilisation d'algorithmes de résolution performants dans le cadre des WCSP aléatoires. Les résultats expérimentaux obtenus montrent que cette approche est utilisable pratiquement pour obtenir une solution de coût optimal pour un nombre d'étapes fixé. Son extension pour l'obtention d'une solution de coût optimal nécessite, pour être utilisée en pratique, l'exploitation de connaissances issues d'une analyse des domaines et des problèmes.

Le chapitre 5 présente ainsi des techniques basées sur une analyse des domaines et des problèmes qui permettent d'obtenir des informations plus précises que les méthodes couramment utilisées :

- Construction d'ensembles d'actions qui sont indispensables à la résolution du problème.
- Construction d'ensembles d'inégalités linéaires à partir du nombre d'occurences des actions et des fluents contenus dans les solutions pour détecter des problèmes de planification sans solution.
- Transformation d'un problème de planification en un problème plus facile à résoudre pour maximiser une borne inférieure au coût d'une solution optimale.

Le chapitre 6 décrit ensuite une méthode originale qui permet de garantir l'obtention d'une solution de coût optimal. Cette méthode, implémentée dans le planificateur GP-WCSP*, est

basée sur l'algorithme de GP-WCSP. Elle poursuit le développement du graphe dans les étapes suivantes après l'apparition du premier plan-solution et son efficacité dépend essentiellement du nombre d'étapes à développer pour garantir l'optimalité. Une borne supérieure à ce nombre peut-être calculée à partir d'un premier plan-solution, puis largement améliorée en utilisant les connaissance extraites grâce à l'analyse des domaines et des problèmes. Nous présentons enfin une comparaison expérimentale de notre approche avec des planificateurs non-optimaux, puis optimaux.

Chapitre 2

Planification optimale

Ce chapitre présentera un état de l'art des différentes approches de planification développées pour obtenir une solution de coût optimal. Nous présenterons ici les approches essentielles qui ont été développées :

- les premières cherchent simplement à minimiser le nombre d'étapes des plans-solutions (section 2.2, page 15),
- les secondes fournissent une solution de coût optimal parmi les solutions optimales en nombre d'étapes (section 2.3, page 27),
- enfin, un dernier type d'approches cherche à produire des solutions de coût optimal sans tenir compte du nombre d'étapes des plans (section 2.4, page 30).

Nous terminerons en présentant les benchmarks que nous utiliserons lors de nos expérimentations et qui sont issus des compétitions IPC.

2.1 Définitions préliminaires

Les différentes approches décrites dans cette section se réfèrent au cadre STRIPS. Dans ce cadre, on pose les définitions suivantes :

Définition 2.1.1 (Problème de planification STRIPS) *Un problème de planification STRIPS est un triplet* $\Pi = \langle A, I, B \rangle$ *tel que :*

1. *l'état initial* I *est un ensemble fini de propositions (aussi appelés fluents) ;*
2. A *est un ensemble d'actions, i.e. de couples* $\langle prec(a), eff(a) \rangle$*, où* $prec(a)$ *est l'ensemble des préconditions propositionnelles de l'action* a*,* $eff(a) = add(a) \cup del(a)$ *est l'ensemble des effets (ajouts et retraits) propositionnels de* a *;*
3. *le but* B *est l'ensemble des propositions qui doivent être satisfaites. Une proposition* p *est satisfaite dans un état* E *ssi* $p \in E$*.*

Définition 2.1.2 (Application d'une action) *L'application d'une action* a *à un état* E *(notée* $E \uparrow a$*) est possible ssi tous les fluents de* $prec(a)$ *sont satisfaits dans* E *; elle produit un état* $E' = (E - del(a)) \cup add(a)$*.*

Définition 2.1.3 (Plan séquentiel) *Un plan séquentiel est une séquence d'actions* a_i*, noté* $P = \langle a_1, \ldots, a_n \rangle$*.*

Définition 2.1.4 (Application d'un plan séquentiel) *L'application d'un plan séquentiel P à un état E_0 (noté $E_0 \uparrow P$) est possible ssi $\forall i \in \{1, \ldots, n\}, a_i$ est applicable dans E_{i-1} et produit l'état E_i. Dans ce cas, on dit que E_n est atteignable à partir de E et P est appelé plan séquentiel correct.*

Définition 2.1.5 (Plan-solution séquentiel) *Un plan séquentiel correct P est un plan-solution séquentiel (ou plan séquentiel valide) ssi E_0 est l'état initial I et que le but B est satisfait dans l'état final E_n.*

Définition 2.1.6 (Actions indépendantes) *Deux actions a et b sont des actions indépendantes (notée $a\#b$) ssi aucune action ne retire un fluent requis ou ajouté par l'autre :*

$$\Big((del(a) \cap (prec(b) \cup add(b))) \cup (del(b) \cap (prec(a) \cup add(a)))\Big) = \emptyset$$

Définition 2.1.7 (Ensemble indépendant) *Un ensemble d'actions $Q_i = \{a_1, \ldots, a_n\}$ est un ensemble indépendant ssi tous les couples d'actions différentes qui le composent sont indépendantes deux à deux : $\forall\ a, b \in Q, a \neq b / a\#b$.*

Définition 2.1.8 (Application d'un ensemble indépendant) *L'application d'un ensemble indépendant d'actions $Q = \{a_1, \ldots, a_n\}$ sur un état E (notée $E \uparrow Q$) est possible ssi $\forall\ a \in Q$, $prec(a)$ est satisfait dans E. Il en résulte l'état E' tel que $E' = (E - \bigcup_{a \in Q} del(a)) \cup \bigcup_{a \in Q} add(a)$. Lorsqu'un ensemble d'actions Q est indépendant, l'application des actions de Q conduit à un état identique quel que soit leur ordre d'exécution.*

Définition 2.1.9 (Plan parallèle, étape) *Un plan parallèle est une séquence d'ensembles indépendants d'actions Q_i, noté $P = \langle Q_1, \ldots, Q_n \rangle$. Dans un plan parallèle, chaque ensemble indépendant correspond à une étape.*

Définition 2.1.10 (Application d'un plan parallèle) *L'application d'un plan parallèle P de n étapes à un état E_0 (noté $E_0 \uparrow P$) est possible ssi pour tout $i = 1, \ldots, n$, Q_i est applicable dans E_{i-1} et produit l'état E_i. Dans ce cas, on dit que E_n est atteignable à partir de E et P est appelé plan parallèle correct.*

Définition 2.1.11 (Plan-solution parallèle) *Un plan parallèle correct P est un plan-solution parallèle (ou plan parallèle valide) ssi E_0 est l'état initial I et que le but B est satisfait dans l'état final E_n.*

Considérons l'exemple de problème de transports suivant.

Exemple 2.1.12 (Problème de planification) *Les variables B, C, D, E et F représentent cinq villes, les variables v et w deux véhicules, et $c1$ et $c2$ deux caisses. Le fluent k_i représente le fait que le véhicule k est dans la ville i. Les véhicules k se déplacent entre les villes en utilisant les actions $trajet^k_{ij}$ où i désigne la ville de départ et j celle d'arrivée. Les fluents $c1_i$ et $c2_i$ représentent le fait que les caisses $c1$ et $c2$ sont dans la ville i, $c1_k$ et $c2_k$ le fait que les caisses $c1$ et $c2$ sont à bord du véhicule k. $c1$ et $c2$ peuvent être chargées dans le véhicule k dans la ville*

i par les actions $prendre1_i^k$ *et* $prendre2_i^k$ *et elles peuvent être déchargées du véhicule k dans une des villes i par les actions* $poser1_i^k$ *et* $poser2_i^k$*. Le problème de planification* $\Pi = \langle A, I, B\rangle$ *est défini par l'état initial* $I = \{v_F, w_F, c1_F, c2_F\}$*, le but* $B = \{c1_B, c2_B\}$ *et l'ensemble des actions* $A : \forall\, i \in \{B, C, D, E, F\}$, $\forall\, k \in \{v, w\}$,

- $\forall\, j \in \{B, C, D, E, F\}$, $trajet_{ij}^k : \langle\{k_i\}, \{k_j, \neg k_i\}\rangle$. *La figure 2.1 donne les trajets possibles.*
- $prendre1_i^k : \langle\{k_i, c1_i\}, \{c1_k, \neg c1_i\}\rangle$ *et* $prendre2_i^k : \langle\{k_i, c2_i\}, \{c2_k, \neg c2_i\}\rangle$.
- $poser1_i^k : \langle\{k_i, c1_k\}, \{c1_i, \neg c1_k\}\rangle$ *et* $poser2_i^k : \langle\{k_i, c2_k\}, \{c2_i, \neg c2_k\}\rangle$.

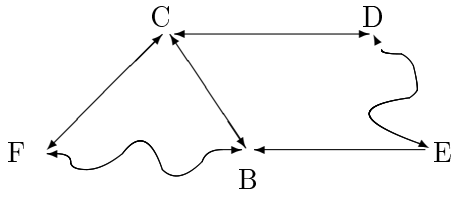


FIG. 2.1 – Déplacements possibles pour l'exemple 2.1.12.

Ce problème peut être résolu par un plan-solution P_1 optimal en nombre d'étapes : $\langle\{prendre1_F^v, prendre2_F^w\}, \{trajet_{FB}^v, trajet_{FB}^w\}, \{poser1_F^v, poser2_F^w\}\rangle$. Or ce plan, bien qu'optimal en nombre d'étapes, n'est pas optimal en nombre d'actions puisqu'il existe un plan-solution P_2 qui contient une action de moins : $\langle prendre1_F^v, prendre2_F^v, trajet_{FB}^v, poser1_B^v, poser2_B^v\rangle$. L'optimalité en nombre d'étapes ne garantit donc pas l'optimalité en nombre d'actions.

2.2 Planification optimale en nombre d'étapes

Nous présenterons dans cette section différentes approches optimales en nombres d'étapes en commençant par les plus anciennes. Dans le cas particulier où chaque étape ne peut contenir qu'une seule action, les plans optimaux en nombre d'étapes sont également optimaux en nombres d'actions. Ce cas sera détaillé dans les sections 2.3, page 27, et 2.4, page 30, lorsque nous considérerons le cadre de la planification de coût optimal avec des coûts uniformes.

2.2.1 Planification par recherche heuristique

Dans les espaces d'états

La planification par recherche heuristique dans les espaces d'états a initialement été utilisée dans le planificateur STRIPS [Fikes et Nilsson, 1971] puis, après une période de désintérêt, est revenue au premier plan avec l'utilisation d'heuristiques efficaces dans le planificateur HSP [Bonet et Geffner, 1999] et ses successeurs. Un espace d'états est un graphe dans lequel chaque nœud représente un état du monde et où chaque arc est étiqueté par une action qui permet la transition entre deux états. Planifier consiste alors à développer une partie de cet espace d'états pour y rechercher un état satisfaisant le but. Un plan-solution correspond au chemin parcouru depuis l'état initial jusqu'au but. Les performances de cette approche dépendent essentiellement de la qualité de l'heuristique utilisée pour ordonner les nœuds qui restent à visiter. Pour estimer la

distance de l'état courant à un état satisfaisant le but, ces heuristiques utilisent le plus souvent une relaxation du problème originel. [Bonet *et al.*, 1997] proposent dans le planificateur ASP une méthode polynomiale d'estimation de l'heuristique pour le problème relaxé obtenu par la suppression des retraits des actions.

Pour obtenir une solution optimale, l'heuristique doit être minorante (ou admissible), c'est-à-dire qu'elle ne doit jamais surestimer le coût réel de la distance entre l'état courant et un état satisfaisant le but. Ainsi, [Bonet et Geffner, 1998, 2001] définissent l'heuristique h_{max} dans laquelle les coûts des préconditions des actions du plan-solution au problème relaxé sont agrégés par l'opérateur max. Malheureusement, cette heuristique minorante est peu informative et ne donne pas de bons résultats car elle ignore les dépendances entre les fluents. [Haslum et Geffner, 2000] l'améliorent en proposant l'heuristique h^m qui approxime le coût d'un ensemble de fluents par le coût du plus coûteux sous-ensemble de taille maximum m. [Haslum et Geffner, 2001 ; Haslum, 2006] montrent ainsi que l'heuristique h^2 est plus informative que h_{max} dans un ensemble de planificateurs optimaux en durée d'exécution qui permettent d'obtenir une solution séquentielle de coût optimal (cf. sous-section 2.4.2, page 32). Enfin, [Rintanen, 2006] montre comment utiliser l'heuristique h_{max} dans des problèmes comportant des préconditions disjonctives et des effets conditionnels. L'heuristique h_{max} ignore les interactions négatives entre les sous-buts (actions qui pourraient les détruire) puisqu'elle ignore les retraits des actions. Le planificateur ALTALT [Nguyen et Kambhampati, 2000 ; Nguyen *et al.*, 2002] introduit l'heuristique admissible "set-level" qui prend en compte ces interactions négatives. Cette heuristique est basée sur le graphe de planification de GRAPHPLAN [Blum et Furst, 1995] (cf. sous-section 2.2.2, page ci-contre) qui permet d'estimer le nombre d'étapes minimum pour obtenir un ensemble de fluents (respectivement un ensemble d'actions) en tenant compte des exclusions mutuelles binaires.

Dans les espaces de plans partiels

Dans les espaces de plans partiels, l'espace de recherche est un graphe dans lequel chaque nœud représente un plan en construction et où chaque arc est étiqueté par une opération permettant de transformer un plan partiel en un autre (ajout d'une action pour satisfaire une précondition, ajout d'une contrainte d'ordre pour résoudre un conflit...). Planifier consiste alors à rechercher une suite d'opérations de modification de plans partiels permettant de passer d'un plan initial vide (ne contenant que deux opérateurs représentant l'état initial et le but) à un plan-solution (plan dans lequel toutes les préconditions sont assertées par des opérateurs du plan et où il n'existe plus de conflit). [Sacerdoti, 1975] fût le premier à introduire les stratégies de moindre engagement (least commitment) qui consistent à n'effectuer certains choix (ordonnancements des actions, instanciation des variables) que lorsque la construction du plan le nécessite absolument. Ces stratégies furent la base de nombreux travaux qui aboutirent à une première formalisation des méthodes de planification dans les espaces de plans partiels [Chapman, 1987]. TWEAK, le planificateur de Chapman, est bâti autour d'un critère de vérité modal (modal truth criterion) qui permet de déterminer la valeur de vérité de tous fluents appartenant aux préconditions des actions d'un plan partiel. [Mccluskey, 1988] puis [Jacopin et Puget, 1990 ; Jacopin, 1993] simplifient ce critère de vérité jusqu'à obtenir un critère minimal qui ne comportent plus de situations mais uniquement des actions. Les planificateurs de la famille de TWEAK ne protègent aucun des sous-buts qui sont établis, ce qui les conduits à devoir asserter à nouveau des fluents qu'ils détruisent. Le planificateur SNLP (Systematic Non Linear Planning) [McAllester

et Rosenblitt, 1991] introduit la notion de liens causaux : une fois que l'on a trouvé une action qui établit un sous-but du plan partiel (établisseur), on crée un lien causal entre l'établisseur et l'action qui le requiert ce sous-but (demandeur). Ce lien causal est protégé des interactions négatives mais aussi positives (actions qui pourraient établir à nouveau un sous-but). [Knoblock et Yang, 1993] comparent TWEAK et SNLP et montrent que la protection des liens causaux ne permet pas à SNLP d'être toujours plus performant que TWEAK. Le planificateur UCPOP [Penberthy et Weld, 1992 ; Weld, 1994] étend le planificateur SNLP à un sous-ensemble du langage ADL. Basé sur UCPOP, le planificateur REPOP [Nguyen et Kambhampati, 2001] utilise des heuristiques basées sur le graphe de planification de GRAPHPLAN [Blum et Furst, 1995] (cf. sous-section 2.2.2, de la présente page) et apporte plusieurs améliorations en utilisant notamment des interdépendances entre actions. REPOP obtient ainsi des solutions de meilleure qualité que le planificateur ALTALT [Nguyen et Kambhampati, 2000].

Dans un cadre temporel dans lequel on associe des durées aux actions, on cherche le plus souvent à minimiser la durée totale de l'exécution du plan-solution. [Vidal et Geffner, 2004, 2006] ont ainsi développé le planificateur temporel CPT, qui est optimal en durée d'exécution et qui combine les possibilités d'exploration des espaces de plans partiels avec des règles d'élagage puissantes et saines. Les techniques CSP sont utilisées pour raisonner sur les supports, les relations de précédence et les liens causaux en faisant intervenir des actions qui n'appartiennent pas encore au plan partiel. CPT s'est classé second de la compétition IPC'2004 dans la catégorie temporelle et optimale en durée d'exécution.

Si les deux techniques précédentes (espaces d'états et espaces de plans partiels) sont les plus anciennes, un autre type d'approche plus récente s'intéresse à la recherche d'une solution de longueur k fixée (en nombre d'étapes). Rechercher un plan de longueur bornée est intéressant car c'est un problème de complexité NP alors que la recherche d'une solution de longueur quelconque est, dans le cadre de la planification propositionnelle STRIPS, PSPACE-Complet à cause de l'existence de problèmes n'ayant que des plans-solutions de longueur exponentielle [Bylander, 1994]. Ces approches (dites par compilation) travaillent essentiellement suivant deux méthodes :

- les premières basées sur le planificateur GRAPHPLAN construisent d'abord, à partir du problème, une structure de taille k fixée (appelée graphe de planification), puis cherchent à en extraire une solution,
- les deuxièmes codent (sous forme de problème (W)SAT, IP ou de (W)CSP) tous les plans d'une longueur k donnée, puis cherchent, en utilisant un solveur dédié, une solution à ce codage pour finir par en déduire le plan-solution. En général, ces codages appréhendent la structure des plans-solutions potentiels sous l'angle des espaces d'états ou des espaces de plans partiels.

Lorsqu'un plan de taille k ne peut être trouvé, la valeur de k est incrémentée et le processus itéré jusqu'à ce qu'un plan-solution soit trouvé. Ce processus incrémental garantit l'obtention d'une solution optimale en nombre d'étapes.

2.2.2 GRAPHPLAN et ses successeurs

Le planificateur GRAPHPLAN [Blum et Furst, 1995, 1997], procède en deux étapes : il construit d'abord une structure de taille k fixée, appelée graphe de planification, susceptible

de contenir tous les plans-solutions de nombre d'étapes maximum k, puis cherche à en extraire un plan-solution. En cas d'échec, la valeur de k est incrémentée et le processus est réitéré. Les plans-solutions qu'il produit sont donc optimaux en nombre d'étapes. GRAPHPLAN a été à l'origine de progrès considérables en planification notamment grâce à l'utilisation du graphe de planification. Cette structure a en effet été utilisée directement, de manière relaxé ou étendue dans de nombreuses approches car elle permet de stocker des informations utiles à la résolution du problème et peut être construite en temps et en espace polynomial [Kambhampati *et al.*, 1997 ; Weld, 1999]. Un exemple de graphe de planification sera détaillé dans la section 4.2, page 56.

Construction du graphe de planification

La construction du graphe de planification permet de mettre en évidence de nombreuses informations comme, par exemple, toutes les actions susceptibles de mener au but. Ce graphe est un graphe orienté composé d'une succession de niveaux composés de nœuds d'actions et de fluents, et d'arcs de préconditions, d'ajouts et de retraits. Le niveau 0 contient uniquement les fluents de l'état initial. Chaque niveau $i > 0$ est ensuite composé de toutes les actions applicables à partir des fluents du niveau $i-1$, et des fluents produits par les actions du niveau i. Les arcs représentent les relations entre les actions et les fluents : les actions du niveau i sont reliées aux fluents préconditions du niveau $i-1$ par des arcs préconditions, et à leurs ajouts et retraits par des arcs d'ajouts et de retraits.

Le maintien des fluents du niveau $i-1$ au niveau i est assuré par des actions factices appelées *noop* qui ont un unique fluent comme précondition et ajout. GRAPHPLAN propose ainsi une solution au problème du décor [McCarthy et Hayes, 1969]. A chaque niveau du graphe, on calcule des exclusions binaires (mutex) entre paires d'actions et paires de fluents. Deux actions sont dites mutex à un niveau s'il est impossible de les appliquer simultanément à ce niveau. Deux fluents sont mutex à un niveau s'ils ne peuvent pas être présents simultanément dans ce niveau. Deux actions qui ne sont pas mutex à un niveau donné y sont indépendantes.

Le graphe est étendu niveau par niveau jusqu'à ce que les fluents du but soient présents dans le niveau courant et qu'il n'existe pas de mutex entre eux.

Extraction d'une solution du graphe de planification

La phase de construction du graphe permet de stocker les exclusions binaires entre actions et entre fluents mais ne permet pas d'obtenir directement une solution. En effet, il faut vérifier qu'il n'existe pas d'exclusions d'ordre supérieur entre les actions (ou les fluents) d'un même niveau. Dans GRAPHPLAN, la phase d'extraction utilise un algorithme classique de recherche arrière avec backtrack pour rechercher un plan-solution.

En cas de succès de cette phase d'extraction, l'algorithme retourne un plan-solution optimal en nombre d'étapes d'actions indépendantes puisque la construction a été incrémentale et qu'aucun plan-solution n'a été trouvé auparavant. Sinon on mémorise cet échec comme nogood pour éviter de refaire plus tard un travail identique et on poursuit la construction du graphe de planification sur un niveau supplémentaire avant de relancer la phase d'extraction. L'algorithme est complet et s'arrête en fournissant un plan-solution ou en signalant un échec si le graphe contient deux niveaux successifs identiques (mêmes actions, fluents, mutex et nogoods) qui n'ont pas de solution.

Extensions et optimisation de GRAPHPLAN

De nombreux planificateurs ont étendu l'expressivité des problèmes traités par GRAPHPLAN. Le planificateur IPP [Koehler *et al.*, 1997] est ainsi capable de résoudre des problèmes de planification décrits avec le langage ADL. [Kambhampati *et al.*, 1997] détaillent comment prendre en compte les fluents négatifs, les effets conditionnels, les quantificateurs ou les préconditions disjonctives tout en conservant les propriétés originelles de GRAPHPLAN. Le planificateur SGP [Weld *et al.*, 1998] étend GRAPHPLAN au cadre incertain et optimise la probabilité des actions du plan-solution.

La phase de construction du graphe de planification a également été améliorée. [Kambhampati *et al.*, 1997] diminuent la taille des niveaux du graphe en ne prenant en compte que les actions susceptibles de mener au but. Cette approche oblige néanmoins à construire le graphe à partir de l'état initial et à partir du but. En cas de poursuite de la construction, il est alors nécessaire de refaire ce travail. Les planificateurs TGP [Smith et Weld, 1999] et STAN [Long et Fox, 1999] montrent comment implémenter efficacement le graphe de planification en stockant un seul ensemble de nœuds d'actions et un seul ensemble de nœuds de fluents. La gestion et le stockage des arcs ont aussi été améliorés. Le traitement des exclusions mutuelles peut ainsi être simplifié en mémorisant les mutex permanents c'est-à-dire ceux qui seront toujours présents quelque soit le niveau du graphe (par exemple les mutex d'actions ont des effets contradictoires). D'autres mutex, une fois qu'ils auront disparus à un niveau du graphe, ils ne réapparaitront plus et il est donc inutile de les recalculer dans les niveaux suivants [Smith et Weld, 1999].

Pour la procédure d'extraction, différentes heuristiques ont été proposées pour en améliorer les performances :

- l'heuristique "noop-first", utilisée par GRAPHPLAN, sélectionne les actions en commençant par choisir préférentiellement les actions noop pour établir un fluent, avant les autres actions disponibles pour essayer de diminuer le nombre d'actions des solutions. [Kambhampati et Nigenda, 2000 ; Cayrol *et al.*, 2000] montrent que dans de nombreux domaines cette stratégie augmente le temps de recherche.
- L'heuristique "plus-petit-domaine-d'abord" [Kambhampati, 2000] choisit d'abord le sous-but asserté par le plus petit nombre d'actions mais, même en supprimant les actions non applicables à cause des mutex, le temps d'exécution n'est que faiblement amélioré.
- L'heuristique "level-based" [Cayrol *et al.*, 2000 ; Kambhampati et Nigenda, 2000] utilise le niveau d'apparition d'un fluent (respectivement d'une action) dans le graphe de planification. Généralement, plus le niveau d'apparition d'un fluent est élevé, plus l'obtention de ce fluent est difficile. Commencer par les fluents ayant un niveau d'apparition les plus élevés, en utilisant des actions de niveau d'apparition le plus faible, améliore très significativement le temps de résolution mais augmente le nombre d'actions.
- L'heuristique "noop-first-level-based" [Vidal, 2001] combine "noop-first" et "level-based" pour améliorer la qualité de la solution en nombre d'actions tout en conservant les performances de "level-based".

[Hoffmann et Geffner, 2003] montrent comment diminuer le facteur de branchement, dû à un très grand nombre d'actions exécutables à partir de l'état courant, dans les domaines fortement parallèles.

Relaxation de la relation d'indépendance

Pour diminuer le nombre d'étapes nécessaires à l'obtention d'un plan-solution, il est possible de relaxer la relation d'indépendance en une relation d'autorisation [Dimopoulos *et al.*, 1997]. Alors qu'un ensemble d'actions est indépendant ssi n'importe quel ordre d'exécution des actions qui le composent est possible et laisse inchangé l'état résultant, il est dit autorisé lorsqu'il existe au moins un ordre d'exécution pour ces actions qui mène au même état résultant. Le planificateur LCGP [Cayrol *et al.*, 2001] utilise cette relation pour produire des plans-solutions composés d'ensembles d'actions autorisés, ce qui diminue le nombre d'exclusions mutuelles, réduit généralement le nombre de backtracks et augmente les performances de manière significative. D'autres planificateurs, en particulier ceux qui utilisent les approches SAT, ont également utilisé cette relation (cf. sous-section 2.2.3, page 23). Par la suite, nous considérerons que, par défaut, c'est la relation d'indépendance qui est utilisée.

Les idées novatrices introduites par GRAPHPLAN ont été largement utilisées par la plupart des approches qui compilent un problème de planification sous forme de problème (W)SAT, IP ou de (W)CSP.

2.2.3 Planification par satisfiabilité

Alors qu'apparaissait GRAPHPLAN, [Kautz et Selman, 1992] montraient comment encoder manuellement des problèmes de planification du domaine des cubes sous forme de problèmes SAT. Depuis cet article originel, de nombreux autres travaux ayant trait à la planification par satisfiabilité (SAT) ont été développés. Ils montrent tous comment transformer un problème de planification de taille k fixée en une base de clauses. La valeur de k est incrémentée jusqu'à ce qu'un modèle satisfaisant la base puisse être fourni par un solveur dédié. Son décodage fournit un plan-solution qui est donc optimal en nombre d'étapes. L'efficacité de cette approche dépend, pour une grande partie, de la qualité du codage (évaluée par le nombre de variables propositionnelles et de clauses, la longueur des clauses, etc.). Un des avantages essentiels de cette méthode est qu'elle permet de bénéficier directement des progrès des solveurs SAT [Gomes *et al.*, 2008] : grâce à l'utilisation de cette approche, le planificateur SATPLAN'06 est toujours, près de dix ans après le succès de son prédécesseur BLACKBOX, leader de la compétition IPC'2006 dans la catégorie "planificateurs optimaux en nombre d'étapes".

Codages directs des problèmes

Dans l'article original de SATPLAN [Kautz et Selman, 1992], les codages "linéaires" utilisés sont issus du calcul des situations [McCarthy et Hayes, 1969] et nécessitent l'utilisation de "frame axioms" qui indiquent, pour chaque action, que les fluents qu'elle n'affecte pas conservent leur valeur de vérité après son application. Le codage utilisé impose l'application séquentielle des actions ce qui entraîne une augmentation importante de la taille du codage (en nombre de variables, de clauses) avec le nombre d'actions des plans. [Kautz et Selman, 1996] développent plusieurs codages dans les espaces d'états avec noops et avec "frame axioms" explicatifs. Ils remarquent que l'utilisation de ces derniers permet le parallélisme, contrairement aux "frame axioms" classiques qui imposent l'utilisation d'une unique action à chaque niveau. Ils comparent leur système SATPLAN aux planificateurs de référence UCPOP [Penberthy et Weld, 1992] et

GRAPHPLAN et montrent que l'approche SAT peut être compétitive. Les codages de [Kautz et Selman, 1996] seront ensuite largement repris, étudiés et améliorés, en particulier dans [Ernst *et al.*, 1997 ; Mali et Kambhampati, 1999 ; Vidal, 2001 ; Maris *et al.*, 2004]. [Kautz *et al.*, 1996] montrent comment encoder automatiquement des problèmes de planification en problèmes SAT. Ils introduisent l'utilisation du parallélisme et proposent la première version des codages dans les espaces de plans partiels. Ils comparent la taille de différents codages. [Ernst *et al.*, 1997] étudient l'impact de différentes représentations des actions pour le codage de problèmes de planification. Leur planificateur MEDIC est le premier qui prend directement en entrée la description STRIPS d'un problème de planification pour le résoudre automatiquement grâce à un solveur SAT. [Mali et Kambhampati, 1998b] étudient l'impact de plusieurs opérations de raffinement (ordre total / partiel, recherche avant / arrière / bidirectionnelle, espaces d'états / de plans) sur la taille et l'efficacité de différents codages. [Mali et Kambhampati, 1998a] améliorent les performances des codages dans les espaces de plans en réduisant le nombre de variables et de clauses nécessaires. [Mali et Kambhampati, 1999] réalisent une comparaison systématique entre les codages dans les espaces d'états et de plans. Ils améliorent les notations, proposent deux codages plus efficaces que ceux de [Kautz et Selman, 1996] et [Ernst *et al.*, 1997], et démontrent également que les codages les plus compacts sont ceux des espaces d'états. [Rintanen, 2003] montre comment, pour des problèmes de planification présentant des symétries, on peut rajouter aux codages des contraintes de rupture de symétrie qui augmentent de manière importante les performances de la résolution.

Codages utilisant le graphe de planification

[Kautz et Selman, 1996] suggèrent que le codage du graphe de planification pourrait permettre de se concentrer sur les seules actions susceptibles de mener au but. [Baioletti *et al.*, 1998] reprennent cette idée dans leur planificateur CSAT-PLAN, et encodent le graphe de planification pour produire une base de clauses plus compacte. Le planificateur BLACKBOX [Kautz et Selman, 1998, 1999] réutilise les optimisations faites dans STAN [Long et Fox, 1999] pour encoder très efficacement les actions, fluents, et exclusions mutuelles du graphe de planification comme un problème SAT et en extraire une solution. Ce planificateur se montre plus performant que SATPLAN et GRAPHPLAN. BLACKBOX permet l'utilisation de plusieurs solveurs SAT et de la procédure d'extraction de GRAPHPLAN. Il comporte de nombreuses options et permet, par exemple, l'utilisation séquentielle de plusieurs solveurs sur une certaine durée, avec leurs différentes options de fonctionnement.

Le planificateur SATPLAN'04 [Kautz, 2004], qui est une mise à jour du planificateur BLACKBOX, remporte la compétition IPC'2004 dans la catégorie des planificateurs optimaux en nombre d'étapes. Ce résultat est inattendu puisque, par rapport à la version précédente de BLACKBOX, SATPLAN'04 n'intègre pas de nouveau codage ou prétraitement, et qu'il n'utilise pas non plus de procédure de propagation des mutex. [Kautz, 2006] expliquent les raisons de ce succès par les gains en performance des solveurs SAT, la difficulté des problèmes posés et leur forte structuration. En 2006, le planificateur SATPLAN'06 [Kautz *et al.*, 2006] intègre une procédure de propagation des mutex mais seulement pour les fluents (la prise en compte des mutex d'actions entraîne une saturation rapide de la mémoire). Il remporte, à nouveau la compétition IPC'2006 dans la catégorie optimale en nombre d'étapes à égalité avec le planificateur MAXPLAN [Zhao *et al.*, 2006 ; Xing *et al.*, 2006 ; Chen *et al.*, 2007]. Ce dernier intègre à SATPLAN'04 des mutex

de longues distances (londex) qui permettent une réduction de l'espace de recherche en généralisant les mutex classiques. Ils peuvent capturer des contraintes entre actions situées à différents niveaux et sont automatiquement générés par une analyse d'un graphe de transition du domaine de planification, lui-même construit à partir d'une représentation multivaluée du domaine.

Alors que les premières approches SAT [Kautz et Selman, 1992, 1996 ; Kautz *et al.*, 1996] s'intéressaient à des codages pour lequels on connaissait à l'avance la longueur minimale d'un plan-solution, la plupart des approches suivantes cherchent à trouver un modèle aux codages représentant tous les plans d'une longueur fixée en commençant à une longueur minimale et en augmentant cette longueur tant qu'un plan-solution n'est pas trouvé. Au contraire, MAXPLAN utilise un plan non optimal, obtenu rapidemment par le planificateur FF [Hoffmann, 2005], pour chercher une solution dans le graphe de planification contenant un nombre maximum d'étapes. Puis il décrémente le nombre de niveaux du graphe jusqu'à obtenir un graphe non soluble, la dernière solution obtenue est alors optimale en nombre d'étapes. Combinant les stratégies de résolution de MAXPLAN et de BLACKBOX, [Gregory *et al.*, 2007] cherchent la première étape qui résout le but (souvent différente de la première étape qui contient les buts) en ajoutant des contraintes sous forme de CSP. D'autres approches comme [Rintanen, 2004 ; Rintanen *et al.*, 2006] étudient des stratégies basées sur l'évaluation parallèle ou intercalée de plusieurs formules qui peuvent éviter l'évaluation de certaines formules insatisfiables très difficiles. Elles permettent des améliorations importantes des temps de résolution mais au prix d'une perte de la garantie d'optimalité.

Le codage de BLACKBOX a également été amélioré dans le planificateur PARA-L [Robinson *et al.*, 2008b] qui utilise un codage plus compact et plus efficace. Ce codage est obtenu en diminuant la taille de représentation des actions, en factorisant certaines contraintes, et en détectant localement des prémisses des clauses représentant les mutex ou les interférences entre un même opérateur. [Do *et al.*, 2000] proposent une méthode qui permet de déterminer, à partir du graphe de planification, la pertinence des fluents pour le but du problème. Elle calcule des exclusions mutuelles binaires entre fluents et actions correspondant à leur pertinence respective : en employant une des deux actions, on exclut l'autre, et le prouveur SAT peut ainsi effectuer des propagations. Les performances obtenues avec le prouveur RELSAT sont améliorées de façon modérée (au plus 6 fois), alors qu'elles sont dégradées avec le prouveur SATZ.

Lorsque l'on crée une base de clauses à partir d'un problème de planification, on perd la structure de ce dernier alors qu'elle peut servir à guider la résolution. Pour pallier cet inconvénient, [Rintanen, 1998] propose une méthode inspirée de la procédure de Davis et Putnam [Davis et Putnam, 1960 ; Davis *et al.*, 1962] et appliquée directement sur les actions et les fluents que l'on peut obtenir à partir de la description STRIPS du problème (pour un nombre d'étapes donné). Les performances observées sont largement supérieures à celles de GRAPHPLAN mais inférieures à SATZ et WALKSAT qui sont deux solveurs SAT. Cette idée est reprise dans le planificateur DPPLAN [Baioletti *et al.*, 2000] qui code le graphe de planification en une base de clauses sans utiliser les relations du graphe et emploie des variables propositionnelles dont les valeurs sont contraintes pour représenter les actions et les fluents. La phase d'extraction d'un modèle n'utilise pas un solveur SAT mais implémente une procédure de Davis et Putnam afin de propager les conséquences des affectations des variables de manière non-directionnelle (contrai-

rement à ce que fait GRAPHPLAN). Les performances de DPPLAN sont comparables à celles obtenues avec SATZ. Le planificateur LCDPP [Vidal, 2001, 2002], en utilisant la relation d'autorisation (cf. paragraphe de la sous-section 2.2.2, page 20), améliore les performances de DPPLAN.

[Büttner et Rintanen, 2005] proposent d'améliorer le nombre d'actions contenues dans les plans-solutions. Une fois obtenu un premier plan-solution optimal en nombre d'étapes, ils ajoutent des contraintes sur le nombre d'actions pour chercher une solution qui reste optimale en nombre d'étapes mais de meilleure qualité en nombre d'actions.

Codages utilisant la relation d'autorisation

Pour améliorer les codages SAT, la relation d'indépendance entre les actions peut être remplacée par la relation d'autorisation qui permet la présence d'actions à un même niveau s'il existe au moins un ordre d'exécution possible pour ces actions (cf. paragraphe de la sous-section 2.2.2, page 20). [Dimopoulos *et al.*, 1997 ; Cayrol *et al.*, 2001 ; Vidal, 2001 ; Maris *et al.*, 2004] montrent que l'utilisation de cette relation d'autorisation permet de produire des codages plus compacts (en nombre de clauses, de variables) et permet d'améliorer les performances de SATPLAN [Kautz et Selman, 1992], de BLACKBOX, de MEDIC et les codages de [Mali et Kambhampati, 1999]. [Rintanen *et al.*, 2004, 2006 ; Wehrle et Rintanen, 2007] proposent une étude détaillée de différents codages et des algorithmes de résolution des bases de clauses. Ils obtiennent des résultats compétitifs par rapport aux planificateurs FF [Hoffmann et Nebel, 2001] et HSP [Bonet et Geffner, 2001] qui sont des planificateurs performants non optimaux alors qu'ils retournent des plans optimaux en nombre d'étapes d'actions (indépendantes ou autorisées) et en nombre d'actions. Les temps de résolution des codages de [Rintanen *et al.*, 2006] et [Wehrle et Rintanen, 2007] restent en général inférieurs à ceux de PARA-L [Robinson *et al.*, 2008b].

L'approche SAT a également été appliquée à d'autres cadres comme la planification optimale en nombre d'actions avec des préférences simples [Giunchiglia et Maratea, 2007b], ou pour obtenir des solutions de coût optimal pour la planification bornée [Chen *et al.*, 2008a] (cf. section 2.3, page 27).

2.2.4 Planification par programmation linéaire

Le succès des approches SAT pour la planification a initié le développement des approches utilisant la programmation linéaire (LP). Une instance d'un programme linéaire est définie par un ensemble de variables, une fonction objectif et un ensemble de contraintes linéaires exprimées sous forme d'égalités et d'inégalités entre des variables. Si toutes les variables ont des valeurs entières, nous avons un programme linéaire en nombres entiers (IP). Si toutes les valeurs sont 0 ou 1, on obtient une programme binaire en nombres entiers (0-1-IP). Une solution doit satisfaire toutes les contraintes et minimiser (ou maximiser) la fonction objectif. Cette approche a l'avantage de simplifier la modélisation de problèmes de planification comprenant des contraintes numériques et des fonctions objectif linéaires.

[Bylander, 1997] a été le premier à utiliser la programmation linéaire pour coder une heuristique pour la planification de type STRIPS. Un plan d'ordre partiel relaxé, avec des contraintes sur le nombre d'actions, y est codé par une instance d'un programme linéaire et un solveur LP est

utilisé pour déterminer l'existence d'une solution. Cette heuristique admissible permet au planificateur LPLAN de retourner des plans optimaux en nombre d'actions pour un nombre d'étapes donné. Les performances obtenues sont cependant inférieures à celles des approches SAT ou à celles qui utilisent sur le graphe de planification. Comme c'est le cas pour les approches SAT, l'efficacité de l'approche par LP en planification dépend essentiellement de la manière dont est codé le problème de planification en un programme linéaire. [Vossen *et al.*, 1999] comparent deux formulations d'un problème de planification STRIPS en programme linéaire. La première traduit le codage SAT obtenu par le planificateur SATPLAN [Kautz et Selman, 1996] en un programme binaire en nombres entiers. Mais cette approche est moins efficace que les approches SAT ou que celles qui sont basées sur le graphe de planification. Dans la deuxième formulation de [Vossen *et al.*, 1999], les variables du programme binaire en nombres entiers ne représentent plus directement la présence (ou l'absence) d'un fluent à une étape mais les changements de valeurs des fluents. Cette approche modélise le fait qu'un fluent est vrai dans un état si et seulement s'il est ajouté dans cet état par une action ou si sa valeur vrai est progagée depuis un état précédent. Les résultats obtenus sont plus encourageants [Vossen *et al.*, 2001]. [Dimopoulos, 2001] propose différentes idées pour améliorer le deuxième codage de [Vossen *et al.*, 1999]. Elles sont intégrées dans le planificateur OPTIPLAN [van den Briel et Kambhampati, 2005] qui termine second de la compétition IPC'2004 dans la catégorie "planificateurs optimaux en nombre d'étapes". OPTIPLAN utilise le graphe de planification de GRAPHPLAN pour diminuer le nombre de variables et ajoute la modélisation du retrait d'un fluent, non présent dans les préconditions de l'action, dans les changements de valeur des fluents. Alors qu'OPTIPLAN représente les transitions atomiques entre les états, [van den Briel *et al.*, 2005] utilisent des variables d'états multi-valuées qui représentent un ensemble de fluents. Le premier codage, nommé 1SC (One State Change), considère un seul changement d'état pour chaque variable par étape dans le plan mais reste similaire à OPTIPLAN et conserve l'optimalité en nombre d'étapes (d'actions indépendantes). Les autres codages perdent toute notion d'optimalité en nombre d'étapes (d'actions indépendantes ou autorisés) en permettant plus d'un changement d'état pour chaque variable dans une même étape et en relaxant les règles permettant à des actions d'appartenir à une même étape. [van den Briel *et al.*, 2008] montrent que ces codages permettent d'obtenir un planificateur IP performant pour étendre la fonction objectif au cadre de la satisfaction partielle de problèmes de planification [Do *et al.*, 2007] (cf. sous-section 2.3.5, page 29).

La programmation linéaire permet enfin de modéliser facilement des problèmes comportants des contraintes numériques (consommation, production ou utilisation de ressources). Les planificateurs ILPPLAN [Kautz et Walser, 1999] et LPSAT [Wolfman et Weld, 1999, 2001] permettent de résoudre de tels problèmes et demeurent optimaux en nombre d'étapes.

2.2.5 Planification par satisfaction de contraintes

La programmation par contrainte (CP) propose un ensemble de méthodes et d'algorithmes permettant de résoudre efficacement des problèmes fortement combinatoires, modélisés sous forme d'un problème de satisfaction de contraintes (CSP). Un CSP est décrit par un ensemble de contraintes, un ensemble de variables et, pour chacune, un ensemble de valeurs possibles (domaine). Pour obtenir une solution au CSP, on doit assigner à chaque variable une valeur appartenant à son domaine respectif et l'ensemble des contraintes doit être satisfait. Les problèmes

SAT sont des CSP qui ne comportent que des variables et des contraintes booléennes et les programmes linéaires sont des CSP dans lesquels les variables ont des valeurs entières ou réelles et où les contraintes sont linéaires.

Pour produire des solutions optimales en nombre d'étapes, les techniques CSP ont été utilisées de deux manières différentes :

- pendant la résolution du problème de planification ou,
- en compilant le problème de planification en CSP puis en utilisant des solveurs CSP pour produire une solution.

Techniques CSP pour la résolution de problèmes

Le planificateur MOLGEN (MOLecular GENerator) [Stefik, 1981] a été le premier à utiliser la propagation de contraintes. Après avoir décomposé les problèmes en sous-problèmes indépendants, l'ajout de contraintes durant la phase de résolution lui permettait de traiter des inter-dépendances existantes entre les sous-solutions.

De nombreux travaux de planification ont utilisé des techniques CP [Nareyek *et al.*, 2005 ; Baptiste *et al.*, 2006]. [Yang, 1992 ; Joslin et Pollack, 1996 ; Kambhampati, 1994 ; Laborie, 2003] utilisent ainsi les techniques CSP pour traiter les contraintes de liens causaux et de conflits, ou des contraintes de ressources. [Kambhampati *et al.*, 1996 ; Kambhampati, 1998] démontrent l'efficacité de l'utilisation du backtrack intelligent et des explications d'échecs pour la planification. Puis ils améliorent l'extraction de solutions du graphe de planification en combinant des propagations de contraintes (qui correspondent aux contraintes d'exclusions mutuelles binaires) et des techniques basées sur les explications d'échec (des contraintes de mutex d'ordre supérieur peuvent être la cause d'échecs) [Kambhampati, 1997, 2000]. Ces travaux ont été étendus jusqu'à utiliser un solveur CSP pour résoudre les CSP issus du graphe de planification.

Dans le cadre temporel, les actions ont des durées d'exécution et on cherche à minimiser la durée totale d'exécution du plan-solution (makespan). Les planificateurs IXTET [Laborie et Ghallab, 1995], ZENO [Penberthy et Weld, 1994] et RAX [Jónsson *et al.*, 2000] modélisent l'aspect temporel et les consommations de ressources par des CSP pour élaguer la recherche en propageant les contraintes. [Vidal et Geffner, 2006] combinent les possibilités d'exploration des espaces de plans partiels avec des règles d'élagage puissantes dans le planificateur CPT. Ils utilisent des actions qui n'appartiennent pas encore au plan-solution et des techniques CSP pour raisonner sur les supports, les relations de précédence et les liens causaux. CPT est classé second lors de la compétition IPC'2004 dans la catégorie temporelle optimale en makespan puis demeure le seul planificateur temporel optimal à participer aux compétitions IPC'2006 et IPC'2008.

Compilation des problèmes de planification en CSP

CPLAN [van Beek et Chen, 1999] est le premier planificateur qui code un problème de planification en un CSP. Il obtient de bonnes performances avec un algorithme de résolution qui combine le forward-checking avec du backtrack intelligent. Mais le codage des problèmes et des domaines doit être fait par l'utilisateur. Les idées qui sont à la base de CPLAN sont reprises et améliorées dans le planificateur GP-CSP [Do et Kambhampati, 2000, 2001] qui génére de manière automatique, à partir du graphe de planification, un CSP dynamique (DCSP). Un DCSP (ou CSP conditionnel) [Mittal et Falkenhainer, 1990] contient des variables et des contraintes d'activités.

Une solution est une affectation des variables actives qui respectent les contraintes. Un DCSP peut être traduit en un CSP en ajoutant dans chaque domaine une valeur représentant l'inactivité de la variable et en interdisant cette valeur pour les variables actives. Dans GP-CSP, le codage est réalisé ainsi :

- une variable est créée pour chaque fluent d'un niveau donné du graphe de planification ;
- le domaine de chaque variable contient les actions qui ajoutent le fluent représenté par cette variable, augmenté de la valeur $\perp$ qui signifie que la variable est inactive ;
- les contraintes modélisent les préconditions des actions, les effets et les exclusions mutuelles entre les fluents et entre les actions.

Comparé au codage SAT de BLACKBOX, le codage de GP-CSP permet de réduire le nombre de contraintes liées aux mutex et l'espace mémoire nécessaire. [Do et Kambhampati, 2000, 2001] comparent, avec le même solveur CSP que CPLAN, différents algorithmes de résolution et heuristiques d'ordonnancement. GP-CSP a de meilleures performances que GRAPHPLAN et BLACKBOX en combinant du forward-checking avec de l'apprentissage par explication d'échecs basés sur les no-good (ensemble d'affectations non compatibles).

Alors que CPLAN utilise les espaces d'états comme modèle, [Jacopin et Penon, 2000] propose un codage automatique d'un problème de planification propositionnel en un CSP binaire arithmétique dans les espaces de plans partiels. Leur codage est basé sur des liens d'activation c'est-à-dire des intervalles sur lesquels un fluent est activé : un fluent devient actif par son établissement et le reste jusqu'à sa désactivation par son retrait. Les bornes sont des instants qui appartiennent à une échelle temporelle commençant à l'instant initial et terminant avec l'obtention du but. Une solution du problème de planification est obtenue lorsque l'on trouve une valeur possible pour toutes les bornes des intervalles qui respectent les contraintes. Pour cela, le codage associe une variable à chaque borne des intervalles dont le domaine associé est un ensemble d'instants. Les contraintes sont données sous forme arithmétique à partir des préconditions et des effets des actions (l'état initial et le but étant considérés comme des actions particulières). Ce codage performant peut être calculé en temps réel et a été utilisé dans des jeux vidéos.

Plus récemment, dans le planificateur CSP-PLAN, [Lopez et Bacchus, 2003] ont proposé un codage direct d'un problème de planification en CSP binaire qui ne repose pas sur le graphe de planification mais en contient toutes les relations, augmentée de contraintes supplémentaires comme, par exemple, des contraintes binaires additionnelles ou de séquences. Contrairement à GP-CSP, ce codage contient une variable pour chaque fluent et pour chaque action dans chaque étape. Un domaine binaire représente la présence ou l'absence, du fluent ou de l'action, dans cette étape. En éliminant les variables mono-valuées et en raisonnant sur l'ensemble des contraintes, CSP-PLAN réduit le nombre de variables et de contraintes et montre de bonnes performances comparé à GRAPHPLAN, GPCSP et BLACKBOX. CSP-PLAN peut également prendre en compte des contraintes de ressources mais ne maintient que l'optimalité en nombre d'étapes.

[Srivastava, 2000 ; Srivastava et Kambhampati, 1999] regroupent, sous le terme de ressources, des fluents qui modélisent un même type d'objets, et séparent le problème d'allocation de ressources du problème de résolution logique des buts. Les deux problèmes sont codés sous forme de CSP. Cette approche diminue le nombre de variables de la partie logique et permet ainsi de résoudre davantage de problèmes que GRAPHPLAN.

2.3 Planification bornée de coût optimal

Dans la section précédente, nous avons présenté des approches qui fournissent une solution optimale en nombre d'étapes. Il a été montré que déterminer l'existence d'un plan de longueur bornée est NP-complet [Garey et Johnson, 1979] pour certains domaines de benchmarks [Helmert, 2003] et l'augmentation d'un critère d'optimisation rend les problèmes encore plus difficile à résoudre dans la pratique. Cependant, de nombreuses applications réelles nécessitent de telles optimisations. Nous présentons donc, dans cette section, plusieurs approches permettant d'obtenir des solutions de coût minimal parmi les solutions optimales en nombre d'étapes. Pour cela, nous nous plaçons dans un cadre où les actions ont des coûts.

2.3.1 Définitions préliminaires

Définition 2.3.1 (Problème de planification valué) *Un problème de planification avec actions valuées est un triplet $\Pi = \langle A, I, B\rangle$ tel que :*

1. *l'état initial I est un ensemble fini de propositions (fluents) ;*
2. *A est un ensemble d'actions, i.e. de triplets $\langle prec(a), eff(a), cout(a)\rangle$, où $prec(a)$ est l'ensemble des préconditions propositionnelles de l'action a, $eff(a)$ est l'ensemble des effets propositionnels de a : ajouts et retraits de propositions $(add(a) \cup del(a))$, $cout(a)$ est un entier naturel non nul représentant le coût de l'application de l'action a ;*
3. *le but B est l'ensemble des propositions qui doivent être satisfaites.*

Définition 2.3.2 (Métrique d'un plan) *La qualité d'un plan P est estimée par une fonction appelée métrique du plan. Nous considèrerons ici les métriques additives telles que*

$$metrique(P) = \sum_{a \in P} cout(a)$$

Nous noterons $\mathcal{P}_k$ l'ensemble des plans-solutions de longueur inférieure ou égale à k étapes. $\mathcal{P}_k^*$ est l'ensemble des plans-solutions de $\mathcal{P}_k$ qui minimisent la métrique :

$$\forall\ P \in \mathcal{P}_k^* : metrique(P) = \min_{P' \in \mathcal{P}_k} metrique(P')$$

Définition 2.3.3 (Plan k-optimal / Coût k-optimal) *Un plan k-optimal, noté P_k^*, est un plan-solution appartenant à $\mathcal{P}_k^*$ c'est-à-dire un plan-solution qui minimise la métrique parmi les solutions de longueur inférieure ou égale à k étapes. C_k^* est le coût d'un tel plan.*

Définition 2.3.4 (Problème de planification k-optimal) *Un problème de planification k-optimal est un problème de planification avec actions valuées dans lequel les solutions appartiennent à $\mathcal{P}_k^*$.*

Considérons l'exemple 2.1.12 (détaillé dans la section précédente) dans lequel nous supprimons une caisse et où nous ajoutons des coûts à chaque action :

Exemple 2.3.5 (Problème de planification valué) *le problème comporte les variables B, C, D, E et F qui représentent cinq villes, les variables v et w deux véhicules, et c une caisse. Le véhicule k se déplace entre les villes i et j en utilisant l'action $trajet_{ij}^k$. Désormais, le coût*

d'application de cette action $trajet_{ij}^{k}$ *dépend de la distance entre les villes. Le chargement d'une caisse c dans un véhicule k par l'action* $prendre_{i}^{k}$ *dans la ville i a un coût de 5. Cette caisse c peut être déchargée du véhicule k dans une des villes i par l'action* $poser_{i}^{k}$ *de coût 3. Le problème de planification valuée* $\Pi = \langle A, I, B \rangle$ *est défini par l'état initial* $I = \{v_F, w_F, c_F\}$*, le but* $B = \{c_B\}$ *et l'ensemble des actions* $A : \forall\, i \in \{B, C, D, E, F\}$,

- $\forall\, j \in \{B, C, D, E, F\}$, $trajet_{ij}^{v} : \langle \{v_i\}, \{v_j, \neg v_i\}, cout_{ij} \rangle$ *et* $trajet_{ij}^{w} : \langle \{w_i\}, \{w_j, \neg w_i\}, 2 \times cout_{ij} \rangle$*. La figure 2.2 donne les trajets possibles et les coûts associés.*
- $\forall\, k \in \{v, w\}$, $prendre_{i}^{k} : \langle \{k_i, c_i\}, \{c_k, \neg c_i\} \rangle$.
- $\forall\, k \in \{v, w\}$, $poser_{i}^{k} : \langle \{k_i, c_k\}, \{c_i, \neg c_k\} \rangle$.

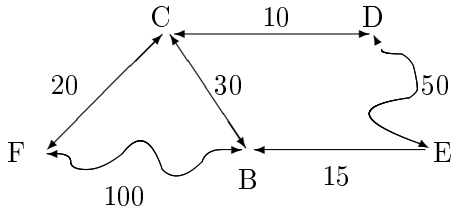


FIG. 2.2 – Déplacements possibles et leurs coûts associés pour l'exemple 2.3.5.

Ce problème contient un seul plan-solution qui minimise la métrique parmi les solutions de longueur inférieure ou égale à 3 étapes : $P_3^* = \langle prendre_F^v, trajet_{FB}^v, poser_B^v \rangle$ de coût 108.

2.3.2 Avec des coûts uniformes

Lorsque tous les coûts des actions sont uniformes, chercher une solution k-optimale revient à chercher une solution qui minimise le nombre d'actions parmi les solutions de longueur k. Dans ce cadre, le planificateur LPPLAN [Bylander, 1997] montre que la programmation linéaire peut être utilisée. Une heuristique admissible, basée sur une relaxation des contraintes d'ordre, estime le nombre d'actions de la solution. Cependant, LPPLAN n'est comparé qu'avec des approches optimales en nombre d'étapes qui montrent des performances inférieures aux approches basées sur le graphe de planification ou aux approches SAT. [Rintanen *et al.*, 2006] montrent que le cadre SAT permet également d'obtenir des solutions séquentielles optimales en nombre d'actions. Leur planificateur SATPLANNER transforme un graphe de planification, basé sur une relation d'autorisation, en une formule propositionelle qui est résolue par un solveur SAT.

2.3.3 Avec des coûts non uniformes

[Chen *et al.*, 2008a,b] ont montré comment utiliser le cadre SAT dans la planification bornée de coût optimal. Leur planificateur PLAN-A est basé sur le dernier codage de SATPLAN et le coût des actions est reporté sur les variables. Leur nouvel algorithme, appelé DPLL-OPT, améliore l'algorithme DPLL [Davis *et al.*, 1962] et leur fonction objectif représente la somme des coûts des variables. PLAN-A résout plus efficacement un plus grand nombre de problèmes que le planificateur SATPlan$\prec$ [Giunchiglia et Maratea, 2007b] qui peut retourner des solutions optimales en nombre d'actions pour la planification de longueur fixée.

Enfin, la programmation logique a été utilisée dans le cadre de la planification bornée avec actions valuées [Eiter *et al.*, 2003]. Le coût de chaque action est statique ou dépend de l'étape dans laquelle l'action est appliquée. Différents types d'optimalité sont développés selon que l'on cherche le plan le plus court en nombre d'étapes ou le moins coûteux.

2.3.4 Avec des préférences

Une extension de SATPLAN, nommé SATPLAN(P) [Giunchiglia et Maratea, 2007a], permet d'obtenir une solution qui maximise des préférences qualitatives et quantitatives simples. Les préférences y sont représentées sous forme de poids pour obtenir une fonction objectif à maximiser. En considérant les problèmes ne contenant que des buts souples, c'est-à-dire des problèmes dont l'objectif n'est pas de satisfaire tous les buts mais de maximiser les préférences, les résultats de SATPLAN(P) sont comparables aux résultats de SGPLAN, vainqueur (sans optimalité) de la compétition IPC'2006 pour les problèmes comportant ce type de préférences.

Le planificateur FGP [Miguel *et al.*, 2000, 2001] utilise les CSP flexibles dans le cadre de la planification flexible. Dans ce contexte, la satisfaction des préconditions n'est pas toujours obligatoire et peut être relaxé moyennant une pénalité sur la qualité de la solution. Les différents buts et les actions du problème comportent des degrés de satisfaction, et le degré de satisfaction de la solution est le niveau minimal entre les actions utilisées et le but obtenu.

Afin de représenter des relations de préférences conditionnelles plus complexes que celles traitées par FGP, le planificateur PREFPLAN [Brafman et Chernyavsky, 2005] utilise les "Conditional Preference Networks" (CP-nets) [Brafman et Domshlak, 2002]. Le formalisme CP-nets permet de décrire comment les préférences d'une valeur pour une variable dépendent de la valeur d'une autre variable. PREFPLAN utilise le codage du problème de planification de GP-CSP, et adapte le solveur CSP pour imposer un ordre d'instanciation consistant avec le CP-net. Les solutions obtenues sont pareto-optimales, c'est-à-dire qu'aucun critère ne peut plus être amélioré sans en dégrader un autre.

Le planificateur PPLAN [Bienvenu *et al.*, 2006] garantit l'obtention du meilleur plan pour une longueur donnée. Pour cela, PPLAN utilise un langage, basé sur une logique temporelle linéaire du premier ordre, pour exprimer des préférences qui peuvent être temporelles.

2.3.5 Avec des bénéfices

Dans le cadre de la planification avec des bénéfices, des coûts sont associés aux actions et des utilités aux buts. On cherche un plan-solution qui optimise le bénéfice entre le gain obtenu par la résolution d'un sous-ensemble de buts et le coût des actions nécessaires à leur apparition. Les approches décrites ici garantissent l'optimalité du bénéfice seulement pour un nombre d'étape donné et non pas de manière globale.

Ce critère d'optimisation peut être exprimé dans les approches par programmation linéaire. OPTIPLAN [van den Briel et Kambhampati, 2005] est un planificateur performant qui compile le graphe de planification en un programme linéaire, puis le résout avec un solveur LP. En changeant la fonction objectif, en supprimant la contrainte qui oblige tous les buts à être satisfaits, puis moyennant quelques adaptations supplémentaires, [van den Briel *et al.*, 2004a,b]

montrent qu'OPTIPLAN peut trouver des solutions optimales en terme de bénéfices parmi les plans-solutions de longueur donnée. [Do *et al.*, 2007] étendent ces travaux au cadre dans lequel les coûts et les bénéfices peuvent être liés aux autres buts ou actions. Pour cela, ils combinent un programme linéaire et une heuristique admissible h_{max} mais ne détectent pas les dépendances d'actions pour achever un but. A l'inverse, [Bonet et Geffner, 2006] intègrent dans leur heuristique quelques interactions entre actions mais ils ignorent les exclusions mutuelles entre elles.

Le planificateur SATPLAN$\prec$ [Giunchiglia et Maratea, 2007b] est une extension de SATPLAN pour la planification bornée partielle. L'utilisation de OPSTAT [Giunchiglia et Maratea, 2006] lui permet de résoudre des instances SAT en minimisant le nombre d'affectations de la valeur vraie à un sous-ensemble de variables. Il est alors possible d'obtenir des solutions optimales en nombre d'actions et en maximisant le nombre de buts souples satisfaits.

2.4 Planification de coût optimal

Nous avons présenté, dans la section précédente, différentes techniques qui permettent d'obtenir une solution de coût optimal parmi les solutions optimales en nombre d'étapes. Or, ces solutions ne sont pas toujours globalement optimales. Considérons l'exemple 2.3.5, décrit dans la section précédente, où le plan-solution trouvé $P_3^* = \langle prendre_F^v, trajet_{FB}^v, poser_B^v \rangle$ a un coût $C_3^* = 108$. Ce plan-solution de coût 3-optimal n'est cependant pas de coût optimal, puisqu'il existe un plan de coût optimal en 4 niveaux $\langle prendre_F^v, trajet_{FC}^v, trajet_{CB}^v, poser_B^v \rangle$ de coût 58. Nous présenterons dans cette section plusieurs approches permettant d'obtenir une solution de coût optimal.

Définition 2.4.1 (Plan de coût optimal / Coût optimal) *Un plan de coût optimal P^* est un plan-solution qui minimise la métrique. C^* est le coût d'un tel plan.*

Définition 2.4.2 (Problème de planification de coût optimal) *Un problème de planification de coût optimal est un problème de planification avec actions valuées dans lequel les solutions sont des plans-solutions qui minimisent la métrique.*

2.4.1 Avec des coûts uniformes

Nous présenterons, dans cette sous-section, les approches de coût optimal dans le cadre où les actions ont un coût uniforme. Cette optimalité est équivalente à l'optimalité en nombre d'actions. Une approche optimale en nombre d'étapes peut être utilisée dans ce cadre en forçant simplement la séquentialité des solutions. Ces approches ayant été décrites dans la section 2.2, page 15, nous ne les redétaillons pas ici.

Dans les approches par IP

Le premier planificateur utilisant la programmation linéaire [Bylander, 1997] propose une heuristique admissible, basée sur la relaxation des contraintes d'ordre, qui permet l'obtention de solutions séquentielles optimales en nombre d'actions parmi les solutions optimales en nombre d'étapes. [van den Briel *et al.*, 2007] réutilisent cette heuristique pour estimer le nombre d'actions de la solution. Le codage modélise une action par une seule variable représentant l'occurence de

cette action dans la solution. Ce codage est indépendant du nombre d'étapes et permet d'obtenir des solutions séquentielles optimales en nombre d'actions. [van den Briel *et al.*, 2007] évoquent la possibilité d'étendre ces travaux à une optimalité en terme de coûts et l'extension détaillée dans [Benton *et al.*, 2007b] concerne la planification avec des bénéfices.

Dans les approches CSP

Le planificateur FDP (Filtrage et Décomposition pour la Planification) [Grandcolas et Pain-Barre, 2006b,a, 2007] représente les problèmes sous forme de CSP dans une structure proche du graphe de planification sans noop. FDP n'utilise pas de solveur externe mais intègre des mécanismes CSP spécifiquement élaborés pour la planification. Les règles de consistance et les mécanismes de filtrage utilisés sont très similaire à la consistance d'arc des techniques CSP. La décomposition consiste à partager l'ensemble des actions d'une étape en regroupant les actions qui retirent les mêmes fluents. L'algorithme de recherche est un IDA* avec une heuristique admissible de coût constant 1.

Dans les approches heuristiques

[Bonet *et al.*, 1997] ont été les premiers à proposer une heuristique basée sur la longueur d'une solution optimale (en nombre d'actions) d'une solution relaxée (en ignorant les retraits). Même si [Bylander, 1994] montrent que calculer la longueur d'une solution optimale dans un problème relaxé en ignorant les retraits est également NP-difficile, [Hoffmann, 2001, 2002, 2005] implémente l'heuristique admissible h^+ basée sur cette longueur. Il étudie alors les propriétés topologiques de nombreux domaines pour expliquer le succès des heuristiques utilisées dans des approches non-optimales et basées sur différentes approximations de h^+ [Bonet et Geffner, 2001 ; Hoffmann et Nebel, 2001 ; Edelkamp et Helmert, 2001].

Le planificateur HSPR* [Haslum et Geffner, 2000] utilise les travaux de [Bonet et Geffner, 1999] pour la planification optimale en nombre d'actions. Il effectue une recherche arrière (en partant des buts) dans un espace d'états en utilisant l'algorithme IDA*. L'heuristique admissible h^2 (ou *max-pair*) qu'ils utilisent est calculée une seule fois et estime le coût d'un ensemble de fluents par le plus coûteux sous-ensemble contenant au maximum deux élements. HSPR* est efficace pour trouver une solution séquentielle optimale en nombre d'actions mais il est inefficace lorsque les problèmes contiennent du parallèlisme.

Le planificateur BFHSP [Zhou et Hansen, 2004, 2006] réutilise l'heuristique de HSPR*. Il effectue une phase de recherche en arrière avec l'heuristique *max-pair*. En cas d'échec, il alterne avec une phase de recherche vers les buts en utilisant l'heuristique $h^1 = h_{max}$, issue de [Bonet et Geffner, 2001], qui estime le coût d'un sous-ensemble de buts par le maximum du coût des buts. Ils améliorent l'algorithme de recherche en utilisant IDA* en largeur d'abord. En diminuant de manière significative le nombre de nœuds qu'il est nécessaire de stocker lors de la résolution, ils retardent le risque de dépassement des capacités mémoire inhérent à ce type d'algorithme. Ils évoquent une extension pour prendre en compte des coûts non-uniformes. Comme BFHSP, le planificateur SEMSYN [Parker, 2004] retourne des solutions séquentielles optimales en nombre d'actions. Il utilise à la fois une recherche en avant pour trouver des actions applicables dans l'état courant, et une recherche à partir de sous-buts pour trouver un ensemble d'actions qui les

établisse.

Dans un algorithme classique de recherche heuristique en avant, [Helmert *et al.*, 2007] utilisent une heuristique basée sur des graphes abstraits de transitions pour diminuer l'espace de recherche en regroupant différents états. Leur planificateur LFPA (pour "Linear, F-Preserving Abstraction") montre que cette approche est compétitive lorsqu'elle est combinée avec des bases de données de motifs.

2.4.2 Avec des coûts non uniformes

Certaines extensions ou modifications des planificateurs précédents, optimaux en nombres d'étapes ou optimaux en coût dans un autre cadre, peuvent leur permettre d'être optimaux en coût dans le cas d'actions valuées de manière non-uniforme.

Dans les approches SAT

Le succès des approches SAT lors des précédentes compétitions a montré qu'elles permettaient d'obtenir rapidemment une première solution optimale en nombre d'étapes. Le planificateur CO-PLAN [Robinson *et al.*, 2008a], basé sur une version modifié du solveur de SATPLAN'06 [Kautz *et al.*, 2006], permet d'obtenir l'ensemble des solutions de longueur minimale. A partir des coûts des actions, qui avaient été jusque là ignorés, la solution optimale en nombre d'étapes et de coût minimum est utilisée comme une borne admissible du coût de la solution optimale dans une recherche en avant dans les espaces d'états. CO-PLAN participe à la compétition IPC de 2008 dans la catégorie de coût optimal.

Dans les approches CSP

Lors de la compétition IPC de 2008, le planificateur CPT [Vidal et Geffner, 2004, 2006], planificateur temporel optimal en durée d'exécution est étendu pour résoudre des problèmes séquentiels optimaux en coût en utilisant le coût des actions à la place des durées des actions et en forçant la séquentialité.

De même, le planificateur FDP [Grandcolas et Pain-Barre, 2007] est étendu à la résolution de problèmes séquentiels optimaux en coût. Le planificateur CFDP représente les problèmes sous forme de CSP et intègre des mécanismes CSP pour produire une solution séquentielle de coût optimal. CFDP utilise une première solution optimale ne nombre d'actions qui permet d'initialiser une borne inférieure de la longueur et une borne supérieure du coût de la solution optimale. Après avoir calculé tous les coûts des plans séquentielles qui ont une longueur inférieure à une borne fixée, cette borne est incrémentée pour chercher les plans de longueur supérieure. L'algorithme s'arrête quand il n'existe plus de solution moins coûteuse.

Dans les approches heuristiques

[Haslum, 2004, 2006] généralise les travaux faits dans TP4 [Haslum et Geffner, 2001] en estimant le coût d'un ensemble par celui du plus coûteux sous-ensemble d'une taille fixée. Haslum obtient, dans le cadre temporel, une série de planificateurs HSP* qui sont optimaux en nombre d'étapes et également de coût optimal. Différents algorithmes de recherche sont disponibles tel

que le Branch-and-Bound ou l'algorithme A*, la version par défaut étant l'algorithme IDA* dans une recherche en arrière. Un prétraitement, qui utilise des méthodes de programmation dynamique, permet de calculer une seule fois le coût des différents sous-ensembles. Ils sont alors stockés dans une table ce qui permet, à chaque étape, de calculer rapidemment la valeur de l'heuristique.

Les versions HSP_0^* (réimplémentation de TP4) et HSP_a^* utilisent l'heuristique h^2. Par rapport à TP4, HSP_a^* contient une étape supplémentaire entre le calcul de l'heuristique h^2 et le début de la recherche : pour améliorer la qualité de l'estimation, il recherche la valeur heuristique relaxée de sous-ensembles de taille supérieure à 2 et utilise l'algorithme IDAO*, adaptation du IDA* aux graphes ET/OU.

Depuis 2002, une version au moins de HSP* participait à chaque compétition IPC dans la catégorie temporelle optimale en durée d'exécution, sauf en 2006 car Haslum était membre du comité d'organisation. En 2008, deux versions sont présentées dans la catégorie non temporelle de coût optimal : HSP_0^* avec une recherche en arrière et HSP_F^* avec une recherche en avant. Elles utilisent l'algorithme A* avec une heuristique additive de type h^2. HSP_F^* obtient la seconde place en 2008 dans cette catégorie.

Avec des techniques de bases de données de motifs

Les heuristiques de bases de données de motifs (PDB) sont basées sur des abstractions de l'espace de recherche et semblent actuellement très prometteuses pour obtenir des heuristiques admissibles. Un motif correspond à une partie du problème (le modèle), assez petit pour être résolu de manière optimale par un algorithme de recherche exhaustif. Les résultats sont stockés en mémoire où ils constituent la base de données de motifs et permettent de définir une fonction heuristique admissible et cohérente.

[Edelkamp, 2001] a été le premier à utiliser les PDB comme heuristiques additives pour la planification en utilisant des variables multi-valuées comme motifs. [Haslum *et al.*, 2005 ; Helmert *et al.*, 2007] améliorent ce travail en incluant dans le calcul la prise en compte de contraintes d'exclusions mutuelles pour obtenir de très bonnes estimations (qui sont même exactes dans certains cas) pour des motifs d'une petite taille fixée. Ils introduisent également une construction incrémentale des motifs pour la recherche en arrière sans réussir à l'étendre facilement à une recherche en avant. L'efficacité de telles heuristiques dépend du choix des motifs. [Haslum *et al.*, 2007] posent le problème de la sélection des motifs comme un problème d'optimisation et proposent un algorithme de recherche locale dans une collection de motifs pour trouver de bons ensembles de motifs.

Une abstraction définit une fonction heuristique admissible comme un calcul de distances dans un petit espace d'états où les états sont en fait des regroupements arbitraires d'états. Les PDB sont alors un cas particulier où les regroupements sont choisis en fonction des valeurs des variables d'états contenus dans le motif considéré. Cette abstraction permet d'améliorer la valeur heuristique en considérant des compositions plus flexibles. Les premiers résultats empiriques pour la planification et la vérification de modèles sont prometteurs [Helmert *et al.*, 2008]. [Katz et Domshlak, 2008b] décomposent l'espace de recherche en utilisant des graphe causaux pour obtenir des motifs structurés, plus généraux que les motifs des PDB mais qui restent néanmoins faciles à résoudre.

[Coles *et al.*, 2008] combinent les heuristiques de la forme h^m de [Haslum et Geffner, 2000] et les heuristiques basées sur les PDB pour obtenir un ensemble d'heuristiques disjonctives-additives. L'heuristique admissible est représentée sous forme de graphe en alternant des nœuds max et somme, comme dans les graphes de [Haslum *et al.*, 2007]. Leur planificateur UPWARDS participe à la compétition IPC de 2008 dans la catégorie optimale en coût.

[Katz et Domshlak, 2008a] proposent d'utiliser la programmation linéaire pour composer de manière additive et optimale des heuristiques admissibles basées sur des abstractions dans une recherche en avant pour chaque état.

[Edelkamp et Kissmann, 2008a,c] exploitent les PDB symboliques pour améliorer la structure de données qui représente l'espace des états utilisé dans la recherche heuristique. Un PDB symbolique est une représentation compactée, basée sur des diagrammes de décision binaires (BDD), de PDB d'états propositionnels explicites. La recherche est combinée à une exploration du périmètre de l'espace de recherche dont l'expansion est limitée par la capacité de la mémoire. L'algorithme de coût optimal est une adaptation de l'algorithme de Dijkstra qui permet de trouver le plus court chemin entre deux sommets d'un graphe connexe dont le poids lié aux arêtes est positif ou nul.

Edelkamp et Kissmann sont les auteurs du planificateur GAMER qui remporte la compétition IPC'2008 dans la catégorie optimale en coût. GAMER utilise une représentation SAS^+ [Backstrom et Nebel, 1995] et effectue une recherche heuristique bidirectionnelle dans les espaces d'états (algorithme largeur d'abord et A*).

Avec des techniques de vérification de modèles

Edelkamp et Helmert sont les premiers à intégrer des techniques de vérification de modèles au cadre de la planification heuristique dans les espaces d'états. Ce sont des techniques qui appliquent un algorithme permettant de vérifier qu'un modèle donné satisfait une spécification, souvent formulée en termes de logique temporelle. Dans leur planificateur MIPS, une phase d'analyse infére des connaissances pour simplifier les descriptions des états et repérer des symétries.

Le planificateur MIPS a évolué en fonction des objectifs des compétitions IPC en y participant depuis 2000 (excepté en 2004 car Edelkamp était organisateur). En 2000 et 2002, [Edelkamp et Helmert, 1999, 2001 ; Edelkamp, 2003] proposent ainsi d'approximer l'heuristique admissible pour améliorer les performances de la version initiale. En 2006, deux versions ont été développées : MIPS-BDD et MIPS-XXL.

Edelkamp propose d'utiliser des diagrammes de décision binaires (BDD) pour représenter un ensemble d'états, MIPS-BDD [Edelkamp, 2005, 2006] retourne alors des solutions optimales en durée d'exécution dans le cadre non temporel. Il peut également optimiser des préférences ou une combinaison de variables numériques intervenant dans les actions et la durée d'exécution de la solution.

Le planificateur MIPS-XXL [Edelkamp *et al.*, 2006a,b ; Edelkamp et Jabbar, 2006] utilise une mémoire externe supplémentaire pour stocker tous les états générés. MIPS-XXL utilise deux algorithmes de recherche :

1. les cinq premières minutes de la résolution sont utilisées par un algorithme de recherche heuristique de type meilleur d'abord avec des variables d'états numériques. L'heuristique est semblable à celle utilisée dans le planificateur METRIC-FF [Hoffmann, 2003] qui est un planificateur efficace pour obtenir des solutions non-optimales. Pour améliorer la qualité de la solution, chaque obtention d'un plan-solution ajoute une contrainte aux buts pour poursuivre la recherche d'une solution de meilleur qualité. Les plans générés sont séquentiels et un ordonnancement de type PERT est fait a posteriori pour obtenir des plans temporels.
2. En cas d'échec de la première recherche, les vingt-cinq minutes restantes sont utilisées par un algorithme qui maintient la meilleur solution tout en garantissant une solution de coût optimal. Une exploration en largeur d'abord exploite une capacité de stockage plus importante tout en respectant les limites de l'espace alloué. Pour accélérer la recherche dans les problèmes ayant un espace de recherche profond, ils utilisent un algorithme de recherche meilleur d'abord focalisé (beam-search) dans lequel, à chaque étape, seuls les k meilleurs états sont développés.

En 2006, la catégorie optimale de la compétition IPC imposait des restrictions sur le temps de calcul et l'espace mémoire et MIPS-XXL n'a participé qu'à la catégorie non-optimale. En 2008 il a été modifié pour résoudre des problèmes séquentiels de coût optimal en satisfaisant les contraintes de la compétition. Il utilise un algorithme de branch-and-bound en largeur d'abord dans les espaces d'états avec un stockage des résultats intermédiaires moins volumineux et des techniques d'élagage qui tiennent compte des coûts des actions.

Dans d'autres approches

[Hickmott *et al.*, 2007] proposent d'utiliser un réseau de Petri déplié dans un algorithme de recherche heuristique pour la planification d'ordre partiel de coût optimal. L'avantage d'un réseau de Petri déplié est qu'il permet l'obtention d'une solution sans phase de recherche car il ne contient qu'une seule possibilité pour l'établissement d'un fluent. Leur planificateur PUP (Planning via Unfolding of Petri nets) montre que l'utilisation de réseaux de Petri dépliés permet de détecter des sous-problèmes indépendants qui peuvent être traités séparemment. Il montre également que la recherche heuristique peut utiliser une analyse d'atteignabilité basée sur le dépliage.

[Borowsky et Edelkamp, 2008] testent l'utilisation de la théorie des automates pour obtenir une solution optimale avec des coûts uniformes, ou des coûts non-uniformes et dépendants de l'état courant. Malheureusement, leur implémentation est inefficace.

2.4.3 Avec des bénéfices

La planification avec des bénéfices étend la planification avec actions valuées en ajoutant des utilités aux buts. On cherche alors à trouver un plan-solution qui optimise la différence entre les gains obtenus par la résolution d'un sous-ensemble de buts et le coût des actions nécessaires à leur résolution.

[Benton *et al.*, 2007a,b] présentent le planificateur BBOP-LP qui utilise une heuristique admissible basée sur les travaux de [Benton *et al.*, 2007b ; van den Briel *et al.*, 2007]. Cette heuristique utilise la programmation linéaire pour résoudre une relaxation du problème original dans laquelle l'ordre des actions est ignoré.

La compétition IPC[1] de 2008 met en avant cette problèmatique en créant une catégorie "planification optimale en terme de bénéfices". Les trois planificateurs qui ont concouru dans cette catégorie sont des extensions de planificateurs qui participent à la catégorie optimale en terme de coûts :

- Haslum présente une nouvelle version de HSP*, nommé HSP^*_P, qui cherche un plan optimal pour chaque sous-ensemble de buts (préalablement déterminés) avec un algorithme par régression IDA* et une variante de l'heuristique h_2. L'amélioration apportée consiste à alterner entre les ensembles de buts pour privilégier le plus prometteur (en terme de bénéfices).
- Edelkamp et Jabbar proposent une nouvelle version de MIPS-XXL qui obtient la seconde place. Ils utilisent un algorithme branch-and-bound en largeur d'abord dans les espaces d'états avec un stockage des résultats intermédiaires moins volumineux et des techniques d'élagage qui tiennent compte des coûts des actions.
- [Edelkamp et Kissmann, 2008b] proposent une recherche symbolique de plus court chemin qui n'explore pas tous les états. Ils adaptent l'algorithme de Dijkstra qui permet de trouver le plus court chemin entre deux sommets d'un graphe connexe dont le poids lié aux arêtes est positif ou nul. Ils utilisent également les BDD pour représenter et manipuler les relations entre les états et pour représenter des préférences. Leur planificateur GAMER est également le vainqueur de cette catégorie.

2.5 Description des benchmarks

Dans cette thèse, nous effectuerons un grand nombre d'expérimentations. Nous utiliserons différents benchmarks des compétitions IPC (International Planning Competition[2]) couvrant des domaines d'application variés dans lequel nous ajouterons aléatoirement un coût entier compris entre 1 et 21 à chaque action. Certains benchmarks des compétitions IPC sont inspirés des problèmes réels et sont fortement structurés [Hoffmann, 2005]. D'autres ont été crées spécialement pour la planification. Chaque domaine comporte une série de problèmes de difficulté croissante, qui est fonction du nombre d'actions et de fluents nécessaires à la représentation de l'état initial et du but. La résolution des problèmes les plus difficiles peuvent nécessiter des plans-solutions de plusieurs milliers d'actions. Nous présentons dans cette section les domaines que nous utiliserons dans nos tests.

Les problèmes de transports ont été modélisés dans plusieurs domaines :

- Le domaine *Logistics* permet de décrire des problème de transport de paquets d'un lieu à un autre en camion (d'une usine à un aéroport par exemple) ou en avion (entre deux villes).
- Le domaine *DriverLog* décrit un problème de livraison de paquets à l'aide de camions. Ces camions sont conduits par des chauffeurs qui se déplacent entre les camions à pied. Les chemins piétons et les routes pour les camions sont différents et donnés sous forme de deux réseaux.
- Dans le domaine *Mystery-prime*, différents emplacements contiennent des véhicules et des unités de carburant. Des objets et le carburant peuvent être transportés en utilisant les

[1] http ://ipc.informatik.uni-freiburg.de/
[2] http ://zeus.ing.unibs.it/ipc-5/

véhicules. Les capacités de chargement des véhicules doivent être respectés et ceux-ci ne peuvent se déplacer que si l'emplacement de départ contient du carburant. La quantité de carburant diminue lors des déplacements et le but est de déplacer les objets en minimisant la consommation de carburant.
- Le domaine *ZenoTravel* décrit un transport de passagers à l'aide d'avions qui utilisent différents modes de vols (économique, rapide, etc.).
- Dans le domaine *Depots*, des camions transportent des caisses qui doivent êtres mises sur des palettes avant d'être acheminées à leur destination. La mise en palettes des caisses est faite à l'aide de grues.
- *BlocksWorld* est le domaine le plus contraint : étant donné un ensemble de tours composées de cubes, le but est de construire un autre ensemble de tours en utilisant un seul bras mécanique pour déplacer les cubes.

Inspirés d'applications spatiales, le domaine *Satellite* décrit la planification d'une collection de tâches d'observation entre des satellites différemment équipés , le domaine *Rovers* utilise des robots terrestres pour trouver des échantillons et retransmettre des données. Le domaine *Storage* concerne le déplacement d'un certain nombre de caisses à partir de containers avec des élévateurs. Dans un dépot, chaque élévateur peut être déplacé selon une carte qui détaille les connections entre les aires du dépot.

Enfin, dans le domaine *Pipesworld*, l'objectif est de contrôler l'écoulement de plusieurs dérivés du pétrole par un réseau de canalisation en respectant des contraintes diverses comme la compatibilité des produits.

2.6 Synthèse

Dans ce chapitre, nous avons présenté un état de l'art des différentes approches de planification développées pour obtenir une solution de coût optimal. Deux stratégies sont essentiellement utilisées :
- Pour obtenir une solution de coût optimal pour un nombre d'étapes fixé, la plupart des approches compilent le problème de planification en un problème SAT, IP ou CSP. Elles procèdent généralement de manière incrémentale en cherchant une solution pour un nombre d'étapes donné et en incrémentant ce nombre jusqu'à l'obtention d'un plan-solution. Différentes méthodes d'extraction permettent ensuite d'obtenir une solution de coût optimal.
- Pour obtenir une solution de coût optimal indépendamment du nombre d'étapes, les approches par recherche heuristique apportent, à l'heure actuelle, les techniques les plus efficaces. Lors de la dernière compétition IPC'2008, huit planificateurs ont ainsi concouru dans la catégorie "sequential optimization track". Ils sont, en général, basés sur un algorithme de recherche optimal (A* ou IDA*) guidé par une heuristique admissible. Dans ce cadre, les approches par compilation ont été peu utilisées : [Robinson *et al.*, 2008a] utilisent un planificateur SAT pour obtenir une première solution qui est ensuite utilisée comme majorant par un algorithme de recherche heuristique. Deux autres planificateurs utilisant les CSP ont concouru dans la compétition IPC'2008 mais aucune publication ne détaille encore leur mode de fonctionnement. Ainsi, Vidal et Geffner modifient leur planificateur CPT pour obtenir une optimalité en coût en utilisant le coût des actions à la place

des durées des actions et en forçant la séquentialité. Grandcolas et Pain-Barre étendent leur planificateur FDP pour obtenir une solution de coût optimal en incrémentant la longueur de l'espace de recherche jusqu'à garantir qu'il n'existe pas de solution moins coûteuse.

Les techniques CSP ont donc souvent été utilisées pour obtenir des solutions de coût optimal, la plupart du temps avec succès. Récemment sont apparues des techniques de CSP intégrant des contraintes valuées (VCSP et WCSP [Schiex *et al.*, 1995 ; Bistarelli *et al.*, 1997, 1999]). Elles permettent de résoudre efficacement des problèmes d'optimisation combinés à des fonctions de coûts locales et leur utilisation pour la planification a, jusqu'à présent, été peu étudiée. Nous nous proposons, dans cette thèse, de développer une méthode de planification qui utilise ces techniques pour produire des solutions de coût optimal. Dans le chapitre suivant, nous commençerons par présenter en détail le formalisme des CSP pondérés.

Chapitre 3

Techniques de résolution pour les CSP pondérés

La méthode que nous allons développer dans cette thèse nécessite l'utilisation de techniques de satisfaction de contraintes pondérées (WCSP) pour coder un graphe de planification [Blum et Furst, 1997] en un WCSP. Le formalisme des WCSP étend celui des CSP et permet de modéliser des préférences pour certains tuples de valeurs, ou encore pour représenter des coûts de violation de contraintes. Ces techniques offrent un cadre rigoureux et des algorithmes performants pour rechercher des solutions qui minimisent le coût des violations de contraintes.

Dans la première section de ce chapitre, nous définirons rapidemment le formalisme des problèmes de satisfaction de contraintes (CSP) avant de présenter, dans la section 3.2 le formalisme des WCSP. Pour résoudre des WCSP de domaines finis, il est en théorie possible d'énumérer toutes les possibilités pour trouver une solution minimisant la fonction objectif. Cependant, le grand nombre de combinaisons possibles rend nécessaire l'utilisation de techniques pour réduire la combinatoire et pour guider la recherche vers les bonnes combinaisons. La section 3.3, page 43, présentera ainsi plusieurs opérations permettant de transformer un WCSP en un WCSP équivalent mais plus facile à résoudre. La section 3.4, page 43, décrira une méthode de résolution utilisée en pratique par les WCSP binaires et implémentées dans le solveur TOULBAR2[1]. Cette méthode nécessitera de bons minorants qui peuvent être améliorés par plusieurs techniques, dites de filtrage par cohérence locale. La section 3.5, page 45, détaillera plusieurs de ces techniques qui permettent, à partir des contraintes, de déduire des valeurs impossibles et de trouver une borne inférieure du coût de la solution. L'efficacité de ces techniques peut encore être améliorée par l'utilisation d'heuristiques de sélection de variables et de valeurs qui seront présentées dans la section 3.6, page 52.

3.1 CSP

Le formalisme des problèmes de satisfaction de contraintes (CSP, Constraint Satisfaction Problem) offre un cadre puissant pour la représentation et la résolution efficace de nombreux problèmes. Ces problèmes sont représentés par un ensemble de variables qui doivent, chacune, être affectées dans leur domaine respectif, tout en satisfaisant un ensemble de contraintes qui expriment des restrictions sur les différentes affectations possibles.

[1] http ://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro

Définition 3.1.1 (CSP [Montanari, 1974]) *Un problème de satisfaction de contraintes (CSP) est un triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ tel que :*

1. *$\mathcal{X} = \{X_1, \ldots, X_n\}$ est l'ensemble des n variables du problème,*
2. *$\mathcal{D} = \{D_1, \ldots, D_n\}$ est l'ensemble des domaines, c'est-à-dire l'ensemble des valeurs possibles pour chaque variable,*
3. *$\mathcal{C}$ est l'ensemble des contraintes du problème. Une contrainte $c \in \mathcal{C}$ est définie comme un couple (P_c, R_c) tel que :*
   - *P_c, appelé portée de c, est un sous-ensemble des variables de $\mathcal{X}$ sur lesquelles porte la contrainte c,*
   - *R_c est l'ensemble des tuples de valeurs qui satisfont c : $R_c \subseteq \prod_{j \in P_c} D_j$.*

Définition 3.1.2 (Affectation) *L'affectation d'une valeur v à une variable X_i est une paire $\{(X_i, v) : v_i \in D_i)\}$, notée $X_i \leftarrow v$. L'affectation d'un ensemble X est l'union de l'affectation des variables de X. L'affectation de l'ensemble des variables du problème est une affectation totale ; elle est dite partielle dans le cas contraire.*

Définition 3.1.3 (Projection) *Etant donnée une affectation $\mathcal{A}$ sur un ensemble V de variables, on note $\mathcal{A}[W]$ la projection de cette affectation sur l'ensemble $W \subset V$. De même, on appelle projection d'une contrainte c sur un ensemble W, notée $c[W]$, l'ensemble des projections sur W des tuples de la relation associée à c. En particulier, si $W = \{x\}$, $c[W]$ sera appelé support de c à la variable x.*

Définition 3.1.4 (Satisfaction de contrainte) *Une affectation $\mathcal{A}$ satisfait une contrainte $c \in \mathcal{C}$ ssi les valeurs des variables de P_c dans $\mathcal{A}$ appartiennent à la relation R_c et sont donc autorisées par R_c : $\mathcal{A}[P_c] \in R_c$.*

La définition du CSP est intimement liée à la notion de violation de contraintes et à celle de cohérence :

Définition 3.1.5 (Affectation cohérente pour un CSP) *Une affectation $\mathcal{A}$ est cohérente (ou consistante) si elle satisfait toutes les contraintes de $\mathcal{C}$: $\forall c \in \mathcal{C}, \mathcal{A}[P_c] \in R_c$; elle est dite incohérente (ou inconsistante) dans le cas contraire.*

Définition 3.1.6 (Solution d'un CSP) *Une solution d'un CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ est une affectation totale et cohérente.*

Définition 3.1.7 (Résolution d'un CSP) *Un CSP ayant au moins une solution est dit cohérent ou satisfiable ; il est incohérent ou insatisfiable dans le cas contraire. Le problème de satisfiabilité d'un CSP est un problème NP-Complet [Mackworth, 1977].*

3.2 CSP pondérés

Les CSP utilisent uniquement des contraintes définies au sens strict (on emploie aussi le terme de contrainte dure) pour permettre de modéliser l'autorisation ou l'interdiction de combinaisons de valeurs. De nombreux formalismes ont étendus les CSP pour modéliser des préférences pour

certains tuples de valeurs, ou encore des coûts de violation (notion de contrainte souple) comme par exemple les réseaux de contraintes floues [Rosenfeld *et al.*, 1976], les réseaux de contraintes possibilistes [Schiex, 1992], les réseaux de contraintes probabilistes [Fargier et Lang, 1993], les réseaux de contraintes basées sur un semi-anneau [Bistarelli *et al.*, 1995, 1997] ou les réseaux de contraintes valuées (VCSP, Valued Constraint Satisfaction Problem) [Schiex *et al.*, 1995 ; Bistarelli *et al.*, 1999]. Ces derniers ajoutent aux CSP une structure de valuation permettant de définir une structure algébrique caractérisant les coûts associés aux solutions du problème. Une structure de valuation est un triplet $S = (V, \oplus, \leq)$ où V est l'ensemble totalement ordonné par $\leq$ des coûts associés aux contraintes, $\oplus$ est un opérateur commutatif, associatif et monotone sur V pour combiner les coûts. La commutativité et l'associativité de l'opérateur $\oplus$ garantissent que la valuation d'une instanciation ne dépend pas de l'ordre dans lequel les valuations sont combinées. La monotonie garantit que les valuations d'instanciations ne peuvent décroître quand les violations de contraintes deviennent plus importantes.

Les CSP pondérés (WCSP, Weighted Constraint Satisfaction Problem) introduits dans [Bistarelli *et al.*, 1997 ; Schiex *et al.*, 1995] sont une sous-classe des VCSP où les coûts sont des entiers naturels ou infinis (i.e $V = \mathbb{N} \cup \{\infty\}$) et l'opérateur $\oplus$ est la somme standard sur les nombres naturels. Cette structure de valuation a la particularité d'être strictement monotone ($\forall a, b, c \in V$ tel que $(a < c) \wedge (b \neq \infty)$, on a $(a \oplus b) < (c \oplus b)$). Comme la somme de coûts finis ne permet pas l'obtention d'un coût infini, la propagation de coûts finis ne permet pas d'inférer d'inconsistance globale. Cet obstacle a été contourné dans les solveurs WCSP dont l'objectif est de trouver une affectation totale de coût inférieur à une valeur finie m. Cette valeur m est alors utilisée comme majorant à la place de l'infini, les valeurs de coûts supérieurs à m sont considérées comme inconsistantes et la propagation de coûts finis peut permettre d'inférer une incohérence globale. [Larrosa, 2002] donne une nouvelle définition des WCSP avec une structure de valuation $S(m) = \langle [0, \ldots, m], \oplus, \leq \rangle$ qui prend en compte ce nouveau majorant m tel que :

- le coût maximum $m \in [1, \ldots, \infty]$ est un entier utilisé pour représenter une contrainte incohérente comme par exemple une exclusion mutuelle,
- $[0, 1, \ldots, m]$ est l'ensemble des coûts entiers borné par m,
- $\leq$ est la relation d'ordre usuelle sur les entiers,
- $\oplus$ définit la somme bornée par m sur les valuations :
  - $a \oplus b = \min\{m, a + b\}$
  - $a \ominus b = \begin{cases} m & \text{si } a = m \\ a - b & \text{sinon.} \end{cases}$

Notons qu'en utilisant la structure de valuation $S(1)$ qui est idempotente ($\forall v \in [0, 1], (v \oplus v) = v$), le WCSP se réduit à un CSP : un coût de 0 signifie la satisfaction et 1 la violation. Nous étudierons des WCSP binaires, c'est-à-dire les WCSP dont les contraintes portent sur au plus deux variables [Larrosa et Schiex, 2003 ; de Givry *et al.*, 2005].

Définition 3.2.1 (WCSP binaire) *Un CSP binaire pondéré (WCSP) est un quadruplet* $\langle m, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ *tel que :*

1. $S(m) = \langle [0, \ldots, m], \oplus, \leq \rangle$ *est la structure de valuation utilisée,*
2. $\mathcal{X} = \{X_1, \ldots, X_n\}$ *est l'ensemble des n variables du problème,*
3. $\mathcal{D} = \{D_1, \ldots, D_n\}$ *est l'ensemble des domaines,*

4. *$\mathcal{C}$ est l'ensemble des contraintes pondérées du problème où chaque contrainte (appelée également fonction de coût) porte sur au plus deux variables :*
   - *Une fonction de coût binaire $c_{ij} : D_i \times D_j \rightarrow [0, \ldots, m]$ (pour $(i,j) \in E$, où $\langle N = \{1, \ldots, n\}, E\rangle$ est un graphe orienté) associe un coût à chaque paire de valeurs affectées simultanément aux variables (X_i, X_j).*
   - *Une fonction de coût unaire $c_i : D_i \rightarrow [0, \ldots, m]$ (pour $i \in N = \{1, \ldots, n\}$) associe un coût à chaque affectation possible de X_i.*
   - *Une fonction de coût d'arité nulle est une constante $c_\emptyset \in [0, \ldots, m]$ qui est ajoutée à n'importe quelle affectation.*
5. *Le coût d'une affectation totale $x \in D_1 \times \ldots \times D_n$ sur l'ensemble $\mathcal{X}$ est donné par :*

$$Cout(x) = c_\emptyset \ \oplus \bigoplus_{i \in N} c_i(X_i) \ \oplus \bigoplus_{(i,j) \in E} c_{ij}(x_i, x_j).$$

Pour simplifier la notation, on supposera qu'il existe une seule fonction de coût sur chaque paire de variables (X_i, X_j) que l'on notera de façon interchangeables $c_{ij}(a,b)$ ou $c_{ji}(b,a)$. Lorsque la fonction $c_\emptyset$ n'existe pas dans le problème originel, elle est ajoutée avec un coût nul. Comme cette fonction ne porte sur aucune variable, son coût doit être payé pour toute affectation. De plus, le coût (positif ou nul) des fonctions s'additionnent ; la valeur de $c_\emptyset$ constitue ainsi un minorant du coût de n'importe quelle affectation des variables et donc un minorant des solutions du problème. La différence majeure entre les WCSP et les CSP est donc que les affectations ont un coût :

Définition 3.2.2 (Affectation totale cohérente pour un WCSP) *Une affectation totale $\mathcal{A}$ est cohérente (ou consistante) si $\mathcal{A} = \Pi_{X_i \in \mathcal{X}} D_i$ et $Cout(\mathcal{A}) < m$; elle est dite incohérente (ou inconsistante) dans le cas contraire.*

Définition 3.2.3 (Solution d'un WCSP) *Une solution du WCSP $\langle m, \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$ est une affectation x totale, cohérente et qui minimise $Cout(x)$. Son obtention est un problème NP-Difficile [Meseguer* et al.*, 2006].*

Pour décrire les WCSP binaires, nous utilisons une représentation à base de graphes dans laquelle les sommets représentent les valeurs, et les arêtes pondérées représentent les paires de valeurs des contraintes binaires avec leur valuation associée. Nous omettons les arêtes de coût nul. Les contraintes unaires sont représentées par des poids associés aux sommets, les poids égaux à 0 sont omis.

Pour illustrer ces définitions et les notions qui suivront, la figure 3.1 présente un WCSP $\langle m, \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$ composé de deux variables X_1 et X_2 de domaines respectifs $D_1 = \{a_1, b_1\}$ et $D_2 = \{a_2, b_2\}$. Par exemple, la fonction de coût binaire $c_{12}(a_1, a_2) = 1$ signifie que l'affectation $\{X_1 \leftarrow a_1; X_2 \leftarrow a_2\}$ a un coût de 1. La fonction de coût unaire $c_2(b_2) = 3$ signifie que l'affectation $\{X_2 \leftarrow b_2\}$ a un coût de 3. Initialement, le coût minimum est $c_\emptyset = 0$ et une solution précédemment trouvée fournit un coût maximum de $m = 4$.

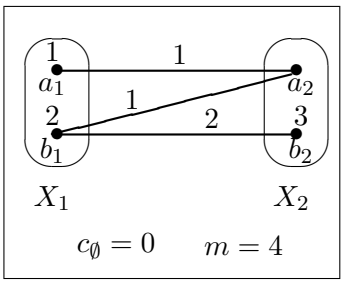


FIG. 3.1 – Exemple de WCSP binaire.

3.3 Opérations sur les WCSP binaires

La réduction de l'espace de recherche par propagation des inconsistances peut être généralisée des CSP aux WCSP, directement par propagation de coûts infinis [Meseguer *et al.*, 2006]. Les coûts finis peuvent aussi être propagés mais de telles propagations doivent être compensées afin de conserver un WCSP équivalent [Cooper et Schiex, 2004].

Définition 3.3.1 (Equivalence de WCSP) *Deux WCSP, définis sur le même ensemble de variables, sont équivalents s'ils ont le même ensemble de solutions et la même distribution de coûts pour toutes les affectations totales du problème.*

Les opérations de transformation préservant l'équivalence (EPT, Equivalence Preserving Transformations) [Cooper et Schiex, 2004] qui suivent ne peuvent être appliquées que si tous les coûts résultants sont non négatifs :

- La **projection unaire** (algorithme 1), définie pour une variable $X_i \in \mathcal{X}$, déplace le coût d'une fonction unaire $v \in D_i$ vers la fonction d'arité nulle $c_\emptyset$.
- La **projection** (algorithme 2), définie pour deux variables $X_i \in \mathcal{X}$ et $X_j \in \mathcal{V}(X_i)$, permet de déplacer le coût d'une fonction binaire portant sur X_i et X_j sur une fonction de coût unaire $v_i \in D_i$.
- L'**extension** (algorithme 3), opération duale de la projection, définie pour deux variables $X_i \in \mathcal{X}$ et $X_j \in \mathcal{V}(X_i)$, permet de déplacer le coût d'une fonction unaire $v_i \in D_i$ sur une fonction de coût binaire portant sur X_i etX_j.

Algorithme 1 Projection Unaire ($X_i \in \mathcal{X}$)

$\alpha \leftarrow min_{v \in D_i}\{c_i(v)\}$
pour tout $v \in D_i$ **faire**
  $c_i(v) \leftarrow c_i(v) \ominus \alpha$;
fin pour
$c_\emptyset \leftarrow c_\emptyset \oplus \alpha$;

3.4 Résolution de WCSP

Les approches de type "programmation dynamique" basées sur l'élimination de variables ou par arbre de jonction ont été largement utilisées pour résoudre des problèmes d'optimisation du

Algorithme 2 Projection ($X_i \in \mathcal{X}, v_i \in D_i, X_j \in \mathcal{V}(X_i)$)

$\alpha \leftarrow min_{v_j \in D_j}\{c_{ij}(v_i, v_j)\}$
pour tout $v_j \in D_j$ **faire**
  $c_{ij}(v_i, v_j) \leftarrow c_{ij}(v_i, v_j) \ominus \alpha$;
fin pour
$c_i(v_i) \leftarrow c_i(v_i) \oplus \alpha$;

Algorithme 3 Extension ($X_i \in \mathcal{X}, v_i \in D_i, X_j \in \mathcal{V}(X_i)$)

pour tout $v_j \in D_j$ **faire**
  $c_{ij}(v_i, v_j) \leftarrow c_{ij}(v_i, v_j) \oplus c_i(v_i)$;
fin pour
$c_i(v_i) \leftarrow c_i(v_i) \ominus c_i(v_i)$;

coût combiné de fonctions de coûts locales. Elles sont cependant intrinsèquement limitées par leur complexité exponentielle en temps et en espace lors de l'application à des modèles graphiques de grande largeur d'arbre. Au contraire, les approches de type "séparation et évaluation" (branch-and-bound) permettent de conserver une complexité spatiale raisonnable mais nécessitent de bons minorants (forts et peu coûteux) pour avoir une efficacité correcte. Cette méthode énumérative utilise la valuation locale de l'affectation courante comme minorant et la valuation de la meilleure solution connue comme majorant :

- Si le minorant est inférieur au majorant, on étend l'instanciation courante en affectant une nouvelle variable.
- Sinon, le sous-problème ne contient pas de solution et la recherche dans ce sous-arbre peut être arrêtée ; on revient en arrière sur la dernière variable instanciée et on l'affecte avec une nouvelle valeur. Si toutes les valeurs ont été essayées, on revient en arrière une nouvelle fois, et ainsi de suite.

En considérant que le but consiste à trouver une première solution optimale inférieure à m, nous pouvons couper les branches de l'arbre de recherche dès lors que $c_\emptyset \geq m$. Nous pouvons également supprimer les valeurs d'une variable $X_i \in \mathcal{X}$ dont le coût unaire est trop élevé pour appartenir à une solution optimale (algorithme 4). Enfin, nous pouvons attribuer un coût de m à un coût binaire $c_{ij}(a, b) \geq m \ominus (c_i(a) \oplus c_j(b) \oplus c_\emptyset)$ tout en obtenant un WCSP équivalent à celui d'origine.

Algorithme 4 Suppression de valeurs ($X_i \in \mathcal{X}$)

pour tout $v \in D_i$ **faire**
  si $c_\emptyset \oplus c_i(v) \geq m$ **alors**
    $D_i \leftarrow D_i \backslash \{v\}$;
  finsi
fin pour

3.5 Cohérences locales dans les WCSP binaires

Durant ces dernières années, des minorants de qualité croissante ont été définis en étendant des propriétés de cohérences locales au cas des WCSP par des méthodes de filtrage. Un algorithme de filtrage transforme un WCSP en un WCSP équivalent qui satisfait la propriété de cohérence locale considérée. Le WCSP obtenu permet de déduire des valeurs impossibles à partir des contraintes et d'incrémenter la borne inférieure (notée $c_\emptyset$) du coût de la solution. Les sous-sections suivantes présentent plusieurs propriétés de cohérences locales définies pour un WCSP binaire $\langle m, \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$.

3.5.1 Cohérence de nœud

En appliquant toutes les projections unaires possibles, on établit la cohérence de nœuds (NC, Node Consistency) [Larrosa, 2002].

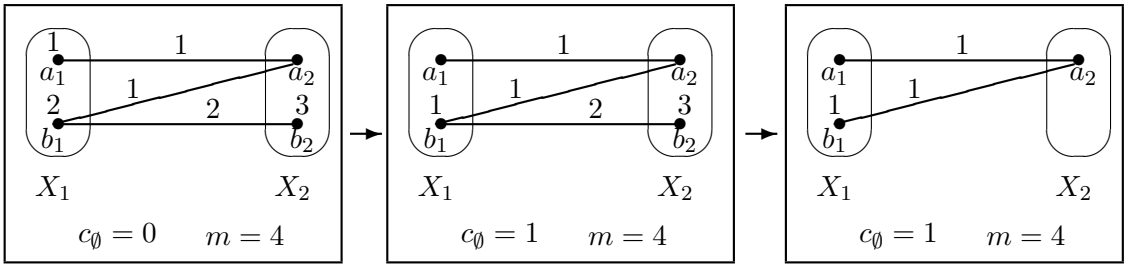
Définition 3.5.1 (Cohérence de nœud) *Une variable X_i est nœud-cohérente si :*

1. $\exists\ v_i \in D_i$ *tel que* $c_i(v_i) = 0$,
2. $\forall\ v_i \in D_i$, $c_\emptyset \oplus c_i(v_i) < m$ *(si* $c_\emptyset \oplus c_i(v_i) = m$ *alors* v_i *est une valeur impossible qui peut être éliminée du domaine :* $D_i \leftarrow D_i \backslash \{v_i\}$*)*

Un problème est nœud-cohérent si toutes les variables sont nœud-cohérentes.

Pour illustrer l'établissement de NC, nous utilisons l'exemple de la section 3.2 dont le WCSP est reproduit dans le cadre (a) de la figure 3.2. La variable X_1 ne respecte pas la première condition de NC car tous les coûts unaires sont positifs. Nous réalisons une projection unaire qui soustrait 1 de $c_1(a_1)$, $c_1(a_2)$ et ajoute 1 à $c_\emptyset$. Nous obtenons le WCSP du cadre 3.2.(b) dont les variables respectent la première condition.

La variable X_2 ne satisfait pas la deuxième condition de NC car l'affectation de la valeur b_2 a un coût unaire de 3 : $c_\emptyset + c_2(b_2) = 1 + 3 = 4 \not< m = 4$. L'affectation $\{X_2 \leftarrow b_2\}$ ne peut donc pas être incluse dans une solution et b_2 est supprimée du domaine de X_2. Le cadre 3.2.(c) montre un WCSP nœud-cohérent équivalent au problème de départ.



(a) Problème Initial (b) Etablissement de la première propriété de NC (c) Etablissement de NC

FIG. 3.2 – Exemple de simplification d'un WCSP par NC.

Le filtrage pour établir NC est uniquement composé de projection unaire et de supressions de valeurs dans les domaines. Il possède une complexité temporelle et spatiale en $\mathcal{O}(nd)$ où n

est le nombre de variables du problème et d la taille maximale des domaines et il est donc très rapide.

3.5.2 Cohérence d'arc souple

La cohérence d'arc (AC, Arc Consistency) dans les CSP est plus forte que la cohérence de nœud car elle fait intervenir le voisinage des variables. La cohérence d'arc souple (SAC, Soft Arc Consistency) [Schiex, 2000 ; Larrosa, 2002 ; Cooper et Schiex, 2004] est une extension d'AC pour les WCSP. Elle est établie par la propagation de tous les coûts infinis (entre les contraintes unaires et binaires) et par l'application des opérations de projection jusqu'à atteindre une convergence.

Définition 3.5.2 (Voisinage d'une variable) *Le voisinage d'une variable X_i, noté $\mathcal{V}(X_i)$, est l'ensemble des variables X_j tel qu'il existe une contrainte $c \in \mathcal{C}$ portant sur X_i et X_j.*

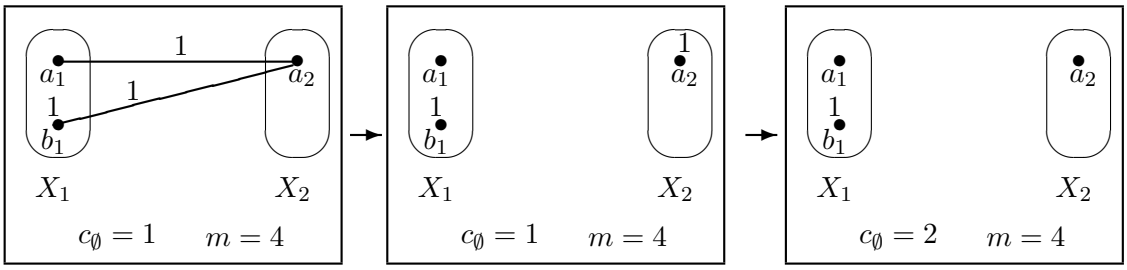
Définition 3.5.3 (Cohérence d'arc souple) *Une variable X_i est arc-cohérente souple si :*

1. $\forall\ X_j \in \mathcal{V}(X_i), \forall\ v_i \in D_i$, *il existe une valeur* $v_j \in D_j$ *tel que* $c_{ij}(v_i, v_j) = 0$ *et*
2. X_i *est nœud-cohérente.*
3. $\forall\ X_j \in \mathcal{V}(X_i), \forall\ v_i \in D_i, \forall v_j \in D_j\ c_{ij}(v_i, v_j) = m$ *si* $c_{ij}(v_i, v_j) \geq m \ominus (c_i(v_i) \oplus c_j(v_j) \oplus c_\emptyset)$.

Un problème est arc-cohérent souple si toutes les variables sont arc-cohérentes souples.

Comme exemple d'application de SAC, nous considérons le WCSP décrit dans la section 3.2. Ce WCSP est reproduit, après l'établissement de NC, dans le cadre (a) de la figure 3.3. La variable X_2 ne respecte pas la première condition de SAC car aucune contrainte portant sur a_2 n'est nulle. Nous réalisons une projection de $c_{12}(a_1, a_2)$ et $c_{12}(b_1, a_2)$ sur $c_1(a_2)$ pour obtenir le WCSP du cadre 3.3.(b) dont les variables respectent la première condition d'arc-cohérence souple.

La variable X_2 ne satisfait pas la condition de NC : en effet il n'existe pas de contrainte unaire de coût nul. Après une projection unaire de $c_1(a_2)$ sur $c_\emptyset$, nous obtenons le cadre 3.3.(c) qui montre un WCSP arc-cohérent souple.



(a) WCSP Initial (b) Etablissement de la première propriété de SAC (c) Etablissement de NC

FIG. 3.3 – Exemple de simplification d'un WCSP par SAC.

La complexité en temps de calcul de SAC est en $\mathcal{O}(ed^3 + n^2d^2)$ qui est borné par $\mathcal{O}(n^2d^3)$, où e est le nombre de contraintes binaires. Elle peut être optimisée en $\mathcal{O}(ed^3 + \min\{m, nd\} \times nd)$ en considérant que le nombre de projections unaires ne peut pas dépasser la valeur de la borne supérieure m du coût de la solution [Larrosa et Schiex, 2003 ; de Givry *et al.*, 2005]. La complexité

optimale spatiale est en $\mathcal{O}(ed)$. Dans le cas de structure de valuation strictement monotone (par exemple $m = \infty$), [Cooper, 2003] montre que la cohérence d'arc souple peut s'établir en temps $\mathcal{O}(ed^2)$ et en espace $\mathcal{O}(ed)$.

Maintenir la cohérence d'arc à chaque nœud de l'arbre de recherche est une technique courante dans les problèmes de satisfaction de contraintes. Mais, contrairement aux CSP, les WCSP n'ont pas, en général, une unique fermeture par arc-cohérence [Cooper et Schiex, 2004]. En d'autres termes, l'ordre des opérations a une influence sur le temps de calcul et la qualité de la borne. Trouver une fermeture optimale par arc-cohérence est un problème qui peut être résolu par un programme linéaire mais cette résolution est trop coûteuse en temps de calcul pour être réalisée à chaque nœud de l'arbre de recherche [Cooper *et al.*, 2007a]. Nous présenterons dans les sous-sections suivantes d'autres formes d'arc cohérence.

3.5.3 Cohérence d'arc totale

Une cohérence d'arc plus forte est appelée cohérence d'arc totale (FAC, Full Arc Concistency).

Définition 3.5.4 (Cohérence d'arc totale) *Une variable X_i est totalement arc-cohérente si :*

1. $\forall\ X_j \in \mathcal{V}(X_i), \forall\ v_i \in D_i$, *il existe une valeur* $v_j \in D_j$ *tel que* $c_{ij}(v_i, v_j) + c_j(v_j) = 0$ *et*
2. X_i *est nœud-cohérente.*
3. $\forall\ X_j \in \mathcal{V}(X_i), \forall\ v_i \in D_i, \forall v_j \in D_j\ c_{ij}(v_i, v_j) = m$ *si* $c_{ij}(v_i, v_j) \geq m \ominus (c_i(v_i) \oplus c_j(v_j) \oplus c_\emptyset)$.

Un problème est totalement arc-cohérent si toutes les variables sont totalement arc-cohérentes.

L'inconvénient majeur de FAC est que la plupart des problèmes n'ont pas de fermeture par FAC. Cette cohérence semble inutilisable en pratique mais les sous-sections suivantes présenteront une cohérence d'arc qui réutilisera partiellement FAC avec succès.

3.5.4 Cohérence d'arc directionnelle

Alors que FAC impose des contraintes de coût sur l'ensemble du voisinage d'une variable, la cohérence d'arc directionnelle (DAC, Directional Arc Consistency) [Cooper, 2003] n'est basée que sur une partie du voisinage, calculée en fonction de l'ordonnancement des variables. Nous considérons un ordre arbitraire sur l'instanciation des variables du problème tel que la variable X_i est instanciée avant X_j ssi $i < j$ que l'on note $X_i \prec X_j$.

Définition 3.5.5 (Voisinage (inférieur/ supérieur) d'une variable) *Le voisinage inférieur d'une variable X_i est l'ensemble : $\mathcal{V}^-(X_i) = \mathcal{V}(X_i) \cap \{X_j \in \mathcal{X} | X_j \prec X_i\}$. Par symétrie, le voisinage supérieur d'une variable X_i est l'ensemble : $\mathcal{V}^+(X_i) = \mathcal{V}(X_i) \cap \{X_j \in \mathcal{X} | X_i \prec X_j\}$.*

Afin d'avoir des coûts non nuls disponibles le plus tôt possible pendant la recherche, la cohérence d'arc directionnelle choisit toujours d'envoyer les coûts (par l'intermédiaire d'opérations de projection et d'extension) vers les variables de plus petit indice. Cette propagation a tendance à regrouper les coûts sur les mêmes variables, et permet ainsi d'arriver à une plus grande valeur de $c_\emptyset$ après l'établissement de la cohérence de nœud.

Définition 3.5.6 (Cohérence d'arc directionnelle) *Une variable X_i est directionnellement arc-cohérente si :*

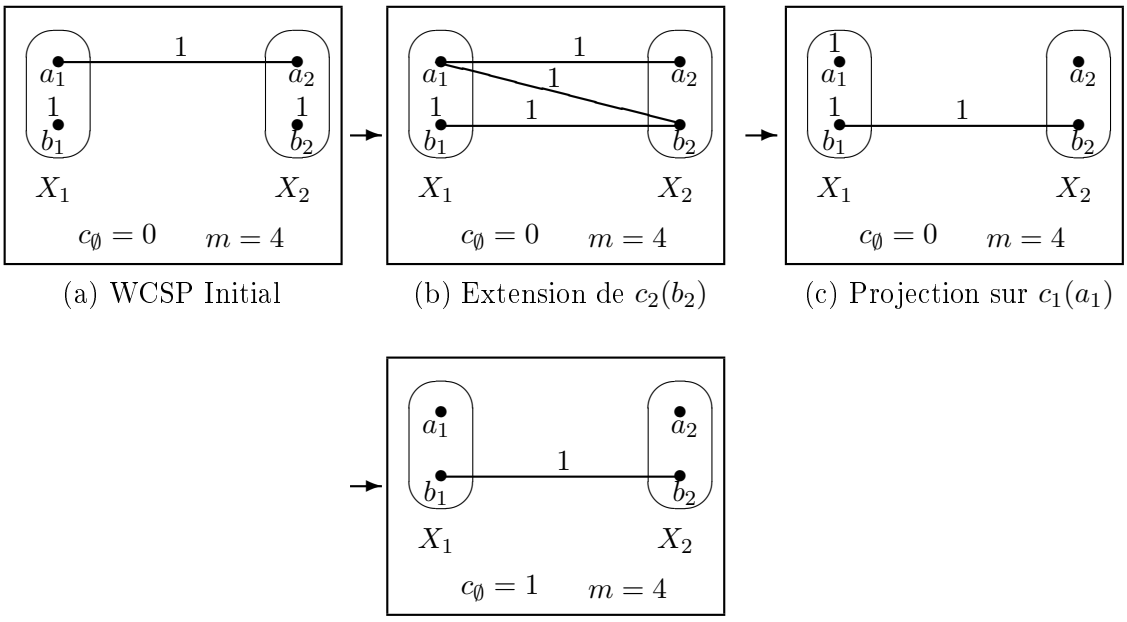
1. $\forall\ X_j \in \mathcal{V}^+(X_i), \forall\ v_i \in D_i$, *il existe une valeur* $v_j \in D_j$ *telle que* $c_{ij}(v_i, v_j) + c_j(v_j) = 0$ *et*
2. X_i *est nœud-cohérente.*
3. $\forall\ X_j \in \mathcal{V}(X_i), \forall\ v_i \in D_i, \forall v_j \in D_j\ c_{ij}(v_i, v_j) = m$ *si* $c_{ij}(v_i, v_j) \geq m \ominus (c_i(v_i) \oplus c_j(v_j) \oplus c_\emptyset)$.

Un problème est directionnellement arc-cohérent si toutes les variables sont directionnellement arc-cohérentes.

Comme exemple d'établissement de DAC, nous considérons le WCSP $\langle m, \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$ représenté dans le cadre (a) de la figure 3.4. Il est composé de deux variables X_1 et X_2 de domaines respectifs $D_1 = \{a_1, b_1\}$ et $D_2 = \{a_2, b_2\}$. La valeur a_1 de X_1 ne respecte pas la première condition par rapport à la variable X_2 : quelle que soit la valeur de X_2, l'affectation $X_1 \leftarrow a_1$ coûte 1. DAC reporte cette information sur la contrainte $c_1(a_1)$ en deux étapes :

1. une extension d'un coût de 1 à partir de $c_2(b_2)$ vers $c_{12}(a_1, b_2)$, $c_{12}(b_1, b_2)$ (cadre 3.4.(b)),
2. une projection d'un coût de 1 à partir de $c_{12}(a_1, a_2)$, $c_{12}(a_1, b_2)$ vers $c_1(a_1)$ (cadre 3.4.(c)).

La dernière opération consiste à faire une projection unaire de $c_1(a_1)$, $c_1(b_1)$ sur $c_\emptyset$ pour rétablir la cohérence de nœud. Le WCSP obtenu dans le cadre 3.4.(d) vérifie la propriété de cohérence d'arc directionnelle par rapport à l'ordre des variables $X_1 \prec X_2$.



(a) WCSP Initial (b) Extension de $c_2(b_2)$ (c) Projection sur $c_1(a_1)$

(d) Projection unaire sur $c_\emptyset$

FIG. 3.4 – Exemple de simplification d'un WCSP par DAC.

Comme la cohérence d'arc souple, la cohérence d'arc directionnelle est établie en temps $\mathcal{O}(ed^2)$ et en espace $\mathcal{O}(ed)$ [Larrosa et Schiex, 2003].

3.5.5 Cohérence d'arc directionnelle totale

L'algorithme de cohérence d'arc directionnelle totale (FDAC, Full Directional Arc Consistency) [Cooper, 2003 ; Larrosa et Schiex, 2003] est la combinaison de la cohérence d'arc direc-

tionnelle et de la cohérence d'arc souple.

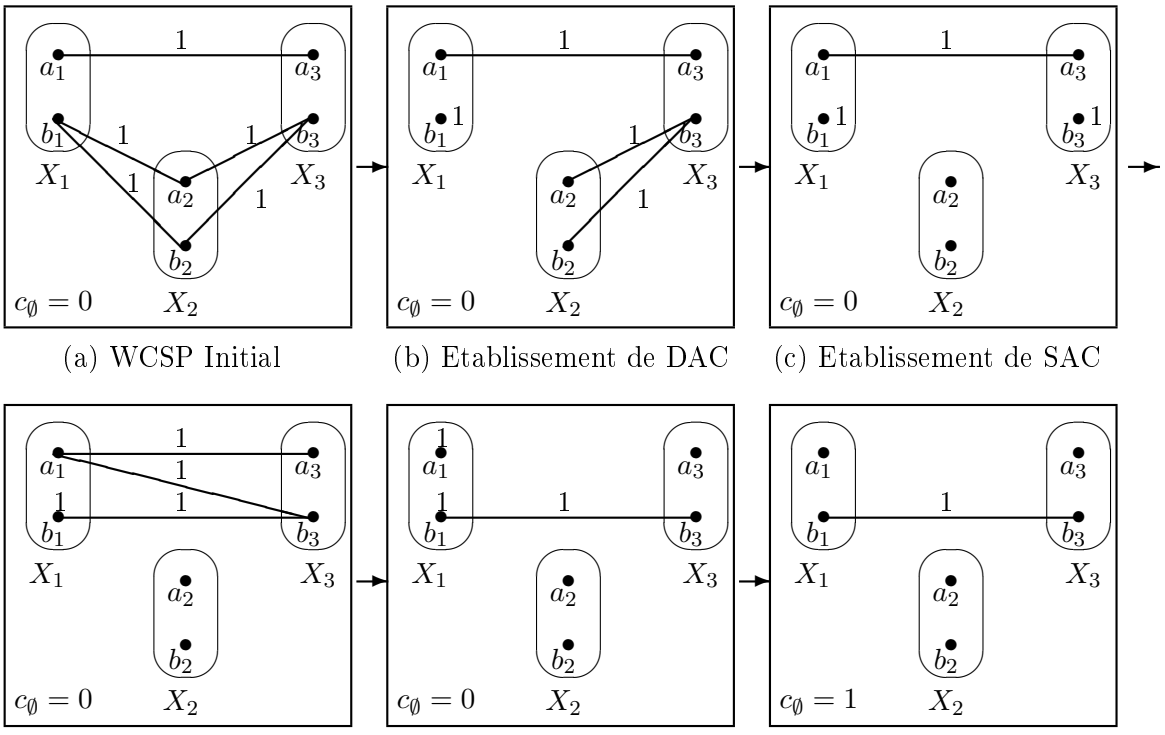
Définition 3.5.7 (Cohérence d'arc directionnelle totale) *Une variable est totalement arc-cohérente directionnelle si elle est arc-cohérente directionnelle et arc-cohérente souple. Un problème est totalement arc-cohérent directionnel si toutes les variables sont totalement arc-cohérentes directionnelles.*

Comme exemple d'application de FDAC, nous considérons le WCSP représenté dans le cadre (a) de la figure 3.5. Ce WCSP est composé de trois variables X_i de domaines $\{a_i, b_i\}$ $(i = 1, 2, 3)$. FDAC commence par projeter un coût de 1 à partir de $c_{12}(b_1, a_2)$, $c_{12}(b_1, b_2)$ vers $c_1(b_1)$ pour établir DAC (cadre 3.5.(b)). Puis une projection à partir de $c_{23}(a_2, b_3)$, $c_{23}(b_2, b_3)$ vers $c_3(b_3)$, montrée dans la figure 3.5.(c), permet d'établir la cohérence d'arc souple.

La variable X_1 n'est plus arc-consistante directionnelle par rapport à X_3. Nous rétablissons DAC en deux étapes :

1. Extension d'un coût de 1 à partir de $c_3(b_3)$ jusqu'à $c_{13}(a_1, b_3)$,$c_{13}(b_1, b_3)$ (cadre 3.5.(d)),
2. Projection d'un coût de 1 de $c_{13}(a_1, a_3)$,$c_{13}(a_1, b_3)$ vers $c_1(a_1)$ (cadre 3.5.(e)).

Une fois établie la cohérence de nœud par une projection unaire de c_1 sur $c_\emptyset$, la figure 3.5.(f) montre un WCSP totalement arc consistant directionnel par rapport à l'ordre des variables $X_1 \prec X_2 \prec X_3$.



(a) WCSP Initial (b) Etablissement de DAC (c) Etablissement de SAC

(d) Extension de $c_3(b_3)$ (e) Projection vers $c_1(a_1)$ (f) Projection unaire sur $c_\emptyset$

FIG. 3.5 – Exemple de simplification d'un WCSP par FDAC.

Nous avons vu que SAC a une complexité de $\mathcal{O}(ed^3 + \min\{m, nd\} \times nd)$ et DAC a une complexité de $\mathcal{O}(nd \times ed^2)$. Comme FDAC itère SAC et DAC dans le pire des cas pour chaque

variable, la complexité en temps de FDAC est $\mathcal{O}(end^3 + \min\{m, nd\} \times nd) = \mathcal{O}(end^3)$ [Larrosa et Schiex, 2003]. FDAC ne nécessite pas de structures supplémentaires par rapport aux autres cohérences d'arc et la complexité en espace reste donc en $\mathcal{O}(ed)$. Dans le cas où $m = \infty$, SAC est établie en temps $\mathcal{O}(ed^2)$ et FDAC peut aussi être établie en temps $\mathcal{O}(ed^2)$ [Cooper, 2003].

Si dans un WCSP, le coût maximum de la solution m vaut un, ou que tous les coûts sont nuls ou valent m, établir DAC revient à établir la cohérence d'arc directionnelle dans un CSP. Etablir FDAC revient à établir la cohérence d'arc.

3.5.6 Cohérence d'arc existentielle

La cohérence d'arc existentielle (EAC, Existential Arc-Consistency) [de Givry *et al.*, 2005] a pour but d'augmenter $c_\emptyset$ en détectant pour chaque valeur v_i dans le domaine de la variable X_i pour laquelle $c_i(v_i) = 0$ et dont il existe un voisin X_j tel qu'il est possible d'augmenter le coût de la contrainte unaire portant sur v_i en déplaçant vers $c_i(v_i)$ le coût de c_j et de la contrainte c_{ij}.

Définition 3.5.8 (Cohérence d'arc existentielle) *Une variable X_i est existentiellement arc-cohérente si :*

1. $\exists\ v_i \in D_i, c_i(v_i) = 0 \wedge \forall\ X_j \in \mathcal{V}(X_i), \exists\ v_j \in D_j$ *tel que* $c_{ij}(v_i, v_j) + c_j(v_j) = 0$ *et*
2. X_i *est nœud-cohérente.*
3. $\forall\ X_j \in \mathcal{V}(X_i), \forall\ v_i \in D_i, \forall v_j \in D_j\ c_{ij}(v_i, v_j) = m$ *si* $c_{ij}(v_i, v_j) \geq m \ominus (c_i(v_i) \oplus c_j(v_j) \oplus c_\emptyset)$.

Un problème est existentiellement arc-cohérent si toutes les variables sont existentiellement arc-cohérentes.

La complexité de EAC est de $\mathcal{O}(nd + ed^2) = \mathcal{O}(ed^2)$ en temps et $\mathcal{O}(ed)$ en espace.

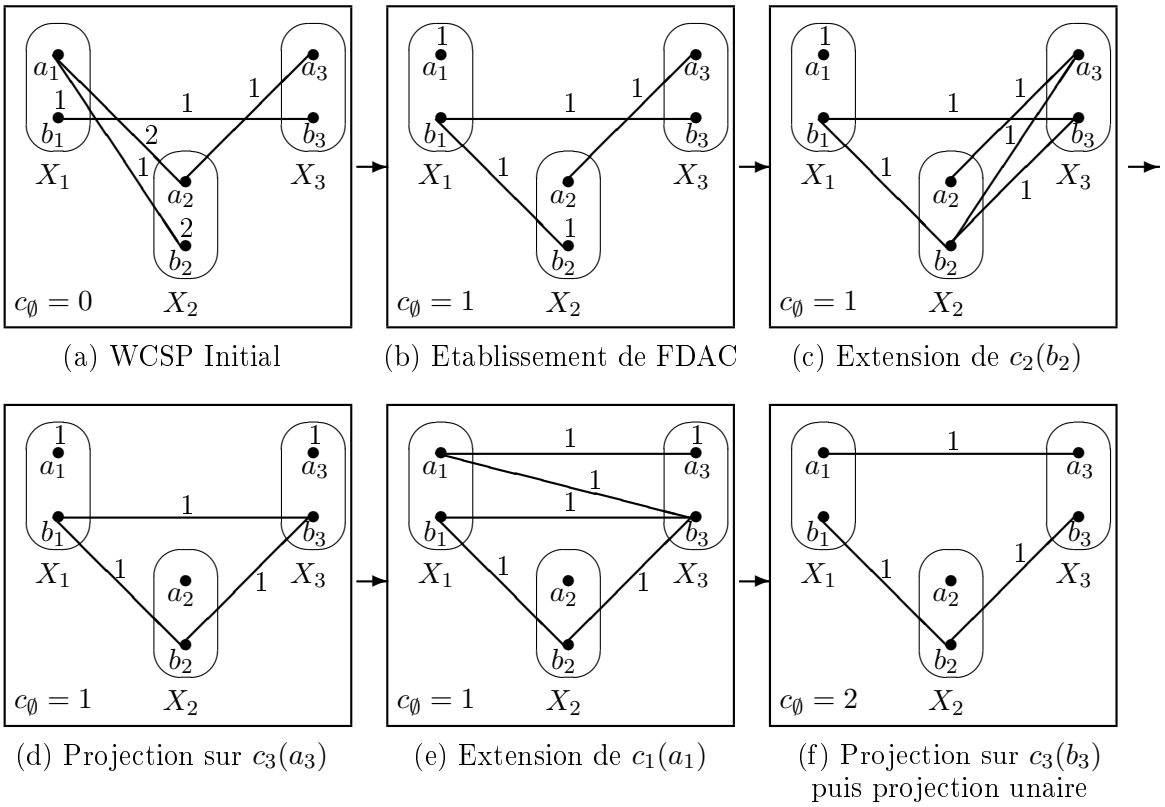
3.5.7 Cohérence d'arc existentielle et directionnelle

En pratique, EAC est établie conjointement avec FDAC pour obtenir une cohérence d'arc existentielle directionnelle (EDAC, Existential Directional Arc-Consistency) [de Givry *et al.*, 2005]. Alors que les précédentes cohérences s'attachent à obtenir une cohérence locale par rapport à chaque contrainte binaire considérée indépendamment, EDAC prend en compte globalement toutes les contraintes qui s'appliquent sur une variable.

Définition 3.5.9 (Cohérence d'arc existentielle et directionnelle) *Une variable est existentiellement arc-cohérente directionnelle si elle est existentiellement arc-cohérente et totalement arc-cohérente directionnelle. Un problème est existentiellement arc-cohérent directionnel si toutes les variables sont existentiellement arc-cohérente directionnelle.*

Considérons le WCSP décrit dans le cadre (a) de la figure 3.6 comme exemple d'application de EDAC. La première étape consiste à établir FDAC dont le WCSP résultant est donné dans le cadre 3.6.(b). Dans le domaine de la variable X_3, la valeur a_3 a un support dans X_2 avec un coût de 1 et, de la même façon, la valeur b_3 a un support dans X_1 de coût 1. En continuant à propager et à étendre des contraintes après l'application de FDAC, il est donc encore possible d'augmenter la borne inférieure $c_\emptyset$; pour augmenter $c_3(a_3)$, EDAC étend un coût de 1 à partir de $c_2(b_2)$ sur $c_{23}(b_2, a_3)$,$c_{23}(b_2, b_3)$ (cadre 3.6.(c)) puis projette un coût de 1 de $c_{23}(a_2, a_3)$,$c_{23}(b_2, b_3)$

vers $c_3(a_3)$ (cadre 3.6.(d)). De manière similaire EDAC augmente de 1 $c_3(b_3)$ par une extension de 1 à partir de $c_1(a_1)$ vers $c_{13}(a_1, a_3)$,$c_{13}(a_1, b_3)$ (cadre 3.6.(e)) suivi d'une projection de 1 de $c_{13}(a_1, b_3)$,$c_{13}(b_1, b_3)$ sur $c_3(b_3)$. Après une projection unaire de 1 de $c_3(a_3)$,$c_3(b_3)$ vers $c_\emptyset$, nous obtenons le WCSP satisfaisant EDAC qui est représenté dans le cadre 3.6.(f).



(a) WCSP Initial (b) Etablissement de FDAC (c) Extension de $c_2(b_2)$

(d) Projection sur $c_3(a_3)$ (e) Extension de $c_1(a_1)$ (f) Projection sur $c_3(b_3)$ puis projection unaire

FIG. 3.6 – Exemple de simplification d'un WCSP avec EDAC.

Nous avons vu que la complexité de EAC et DAC est de $\mathcal{O}(ed^2)$, la complexité de SAC est $\mathcal{O}(ed^3 + \min\{m, nd\} \times nd)$. EDAC est une itération de EAC, DAC et SAC. Dans le pire des cas, EDAC peut être itéré pour chaque valeur ou à chaque fois que m est incrémenté, la complexité de EDAC est ainsi de $\mathcal{O}(ed^2 \times max\{nd, m\} + ed^3 + \min\{m, nd\} \times nd) = \mathcal{O}(ed^2 \times max\{nd, m\})$ en temps et de $\mathcal{O}(ed)$ en espace [de Givry *et al.*, 2005].

Sur des instances de WCSP aléatoires, [de Givry *et al.*, 2005] montrent que EDAC permet d'améliorer significativement le temps de résolution du WCSP par rapport à FDAC, même si le temps de calcul pour chaque nœud de l'algorithme de recherche est augmenté (EDAC, tout comme FDAC, étant appliqué à chaque nœud de l'arbre de recherche).

3.5.8 Cohérence d'arc optimale

La définition de l'arc cohérence optimale (OSAC, Optimal soft arc consistency) [Cooper *et al.*, 2007a] montre qu'il est possible de pré-calculer, en temps polynomial, un ensemble de transformations de WCSP préservant l'équivalence qui déplacent des coûts rationnels et maximisent la

valeur de $c_\emptyset$. Cet algorithme, basé sur la résolution d'un programme linéaire de grande taille, est trop coûteux pour être employé à chaque nœud de la recherche, son utilisation est donc limitée à la phase de prétraitement.

3.5.9 Cohérence d'arc virtuelle

Enfin, la cohérence d'arc virtuelle (VAC, Virtual Arc Consistency) [Cooper *et al.*, 2008a,b] trouve et applique itérativement des séquences d'opérations de propagation de coûts fractionnaires qui nous garantissent la transformation d'un WCSP en un autre WCSP équivalent où $c_\emptyset$ est accru. Bien que plus faible que OSAC, VAC est plus rapide et est capable de résoudre le langage polynomial qui est défini par des fonctions de coût sous-modulaires. Le maintien de VAC pendant la recherche conduit à d'importantes améliorations sur des problèmes difficiles de grande taille comme la résolution de deux instances d'un problème d'affectation de fréquence qui étaient non résolues depuis plus de dix ans [Cabon *et al.*, 1999].

3.6 Heuristiques de sélection de variables et de valeurs

En programmation par contraintes, les heuristiques de sélection des variables et des valeurs jouent un rôle primordial dans la résolution d'un problème en permettant de définir des stratégies de recherche spécifiques et efficaces. D'autre part, plusieurs algorithmes de filtrage sont incrémentaux : si une cohérence est établie dans un nœud alors l'établissement de la cohérence locale dans un nœud fils peut être fait en ne considérant que les modifications entre celui-ci et le nœud père.

[Heras et Larrosa, 2006] étudient différents types d'ordonnancements pour les variables lors de l'établissement de FDAC :

1. Des ordres statiques qui sont calculés avant de débuter la recherche et maintenus fixes pendant la résolution. Ils ont l'avantage d'être exempts de calculs supplémentaires car ils n'interfèrent pas avec l'incrémentation de l'algorithme de filtrage. Mais ils sont basés sur les informations contenus dans le problème original : les modifications faites au cours de la recherche par l'affectation de variables ne sont pas pris en compte, l'information peut donc perdre de l'exactitude.
2. Des ordres dynamiques qui sont calculés à chaque nœud de l'arbre de recherche en utilisant les informations du sous-problème courant. Ils ont l'avantage d'être beaucoup plus précis que les ordres statiques mais nécessitent plus de temps de calcul. En effet, nous avons vu que FDAC est défini en fonction d'un ordonnancement, ainsi un WCSP peut satisfaire FDAC pour un ordonnancement donné et pas pour un autre. Une modification de l'ordonnancement des variables nécessite d'exécuter à nouveau FDAC pour rétablir cette propriété, l'établissement incrémental de FDAC est donc perdu.
3. Des réordonnancements dynamiques considèrent une séquence d'ordonnancement des variables. FDAC est alors établi pour chacun des ordonnancements à chaque nœud de l'arbre de recherche.

[Heras et Larrosa, 2006] montrent que l'ordonnancement statique le plus robuste est celui qui est basé sur l'ordonnancement inverse du degré des variables (le nombre de nœuds adjacents de la variable dans le graphe de contraintes). Pour un ordonnancement dynamique, il semble que

l'heuristique basée sur la somme des coûts unaires donne le meilleur résultat (mais cela n'est pas toujours valable en ce qui concerne les ordonnancements statiques). Enfin, les réordonnancements dynamiques n'ont été utiles que dans certains problèmes.

[de Givry *et al.*, 2003, page 7] obtiennent de bonnes performances pour résoudre des WCSP aléatoires avec FDAC en améliorant un ordonnancement dynamique. Cette heuristique, appelée "2-sided Jeroslow-like", ordonne les variables selon un ratio entre la taille de leur domaine et la somme des moyennes des coûts unaires et binaires des contraintes qui sont liées à cette variable. Après avoir choisi une variable, la valeur qui a un coût unaire le plus faible est affectée en premier.

3.7 Synthèse

Dans ce chapitre, nous avons présenté le formalisme des WCSP qui étend celui des CSP pour représenter des contraintes valuées et qui permet l'obtention de solutions qui minimisent le coût des violations de contraintes.

Nous avons décrit l'algorithme branch-and-bound qui est utilisé pour la résolution de WCSP binaires dans le solveur TOULBAR2[2]. Cette méthode nécessite de bons minorants qui peuvent être obtenus par plusieurs techniques, dites de filtrage par cohérence locale. Les techniques de filtrage décrites, notamment FDAC, EDAC et VAC, permettent, à partir des contraintes, de déduire des valeurs impossibles et de trouver une bonne borne inférieure du coût de la solution. Nous avons ainsi montré l'influence du choix de la propriété de cohérence locale sur la complexité en temps de calcul et sur la qualité des minorants obtenus. De plus, l'efficacité de la résolution peut être améliorée par l'utilisation de différentes heuristiques d'ordonnancement de variables. Les plus efficaces sont des heuristiques dynamiques, c'est-à-dire que les choix opérés doivent être élaborés au cours de la recherche et non au préalable de façon statique et définitive mais elles doivent également chercher à préserver la propriété d'incrémentalité des algorithmes de filtrage.

Le formalisme des WCSP offre ainsi un cadre rigoureux et des algorithmes performants pour résoudre efficacement des problèmes d'optimisation comportant des fonctions de coûts locales. Nous allons montrer dans le chapitre suivant comment utiliser ce formalisme dans le cadre de la planification avec actions valuées pour obtenir des solutions de coût optimal.

[2] http ://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro

Chapitre 4

Solution optimale pour un nombre de niveaux donné

Dans ce chapitre, nous présenterons une nouvelle approche permettant d'obtenir une solution P_k^* de coût optimal parmi les solutions de longueur inférieure ou égale à k étapes d'actions indépendantes [Cooper *et al.*, 2006a,b]. Cette méthode sera implémentée dans le planificateur GP-WCSP et utilisera exclusivement la relation d'indépendance. Nous emploierons donc ici le terme de niveau pour désigner une étape d'actions indépendantes.

Nous avons vu dans le chapitre 2 que la plupart des approches existantes ayant le même objectif utilisent le graphe de planification de GRAPHPLAN [Blum et Furst, 1997], structure disjonctive qui contient, pour un problème de planification donné, tous les plans-solutions potentiels d'une longueur k fixée. De plus, elles encodent le problème de planification dans un autre formalisme tel que SAT ou IP pour en extraire une solution de coût optimal parmi les solutions de longueur k fixée. Cependant, les approches qui utilisent les techniques CSP se limitent à l'optimalité en nombres d'étapes ou au cadre d'actions de coût uniforme. Or, nous avons montré, dans le chapitre 3, que le formalisme des CSP pondérés (WCSP) permettait d'obtenir des solutions de coût optimal de manière très efficace. Nous proposerons donc ici une méthode qui utilisera le graphe de planification et nous montrerons comment utiliser le cadre des WCSP pour trouver un plan-solution de coût optimal parmi les solutions de nombre de niveaux k fixé.

La première section introduira un exemple simple de problème de planification valuée. La section 4.2, page suivante, décrira ensuite le graphe de planification puis la section 4.3, page 57, montrera comment coder ce graphe en WCSP. La section 4.4, page 60, détaillera ensuite les méthodes de résolution de ce WCSP qui permettent d'obtenir un plan-solution de coût optimal pour un niveau k donné. Enfin, la section 4.5, page 74, présentera l'algorithme développé, le planificateur GP-WCSP et les résultats expérimentaux obtenus.

4.1 Exemple de problème de planification valuée

Pour illustrer nos méthodes, nous modifions l'exemple 2.3.5, présenté dans le chapitre 2, qui appartient à la classe des problèmes de plus court chemin pouvant être résolus en temps polynomial par la programmation dynamique. Il a été montré que déterminer l'existence d'un plan de longueur bornée est NP-complet pour certains domaines de benchmarks [Helmert, 2003]

mais l'ajout d'un critère d'optimisation rend les problèmes encore plus difficiles à résoudre en pratique.

Exemple 4.1.1 (Problème de planification valuée) *Les variables B, C, D, E et F représentent cinq villes, la variable v un véhicule, et c une caisse. Le fluent v_i représente le fait que le véhicule v est dans la ville i. v se déplace entre les villes en utilisant les actions $trajet_{ij}$ où i désigne la ville de départ et j celle d'arrivée. Le coût de cette action dépend de la distance entre les villes. Le fluent c_i représente le fait que la caisse c est dans la ville i, c_v le fait que la caisse est à bord du véhicule v. c peut être chargée dans le véhicule par l'action $prendre_i$ dans la ville i, avec un coût de 5. Cette caisse c peut être déchargée dans une des villes i par l'action $poser_i$ de coût 3. Le problème de planification valuée $\Pi = \langle A, I, B\rangle$ est défini par l'état initial $I = \{v_F, c_F\}$, le but $B = \{c_B\}$ et l'ensemble des actions $A : \forall\ i \in \{B, C, D, E, F\}$,*

- *$\forall\ j \in \{B, C, D, E, F\}$, $trajet_{ij} : \langle\{v_i\}, \{v_j, \neg v_i\}, cout_{ij}\rangle$. La figure 4.1 donne les trajets possibles et leurs coûts.*
- *$prendre_i : \langle\{v_i, c_i\}, \{c_v, \neg c_i\}, 5\rangle$.*
- *$poser_i : \langle\{v_i, c_v\}, \{c_i, \neg c_v\}, 3\rangle$.*

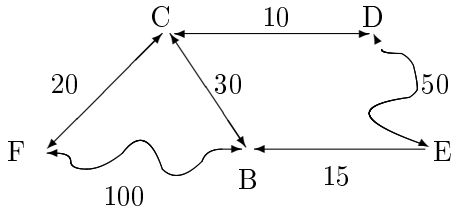


FIG. 4.1 – Déplacements possibles et leurs coûts associés de l'exemple 4.1.1.

Le premier plan-solution trouvé $P_3^* = \langle prendre_F, trajet_{FB}, poser_B\rangle$ a un coût $C_3^* = 108$. Nous allons montrer, dans ce chapitre, comment obtenir un tel plan-solution.

4.2 Construction du graphe de planification

Notre méthode est basée sur le graphe de planification de GRAPHPLAN, dans cette section nous rappelons son principe puis nous en présenterons un exemple.

Un graphe de planification [Blum et Furst, 1995, 1997] est un graphe orienté, construit en temps et en espace polynomial, composé de plusieurs niveaux qui comportent des nœuds d'actions et de fluents, et des arcs de préconditions, d'ajouts et de retraits. Le niveau 0 contient les fluents de l'état initial. Chaque niveau $i > 0$ est composé de toutes les actions applicables à partir des fluents du niveau $i-1$, et des fluents produits par les actions du niveau i. Les arcs représentent les relations entre les actions et les fluents : les actions du niveau i sont reliées aux fluents préconditions du niveau $i-1$ par des arcs préconditions, et à leurs ajouts et retraits du niveau i par des arcs d'ajouts et de retraits.

Le maintien des fluents du niveau $i-1$ au niveau i est assuré par des actions appelées *noop* qui ont un unique fluent comme précondition et ajout. Pour chaque niveau du graphe, on calcule des exclusions binaires (mutex) entre paires d'actions et paires de fluents d'un même niveau :

Définition 4.2.1 (Exclusion mutuelle [Blum et Furst, 1997]) *Deux actions a_1 et a_2 sont mutex à un niveau i du graphe (noté $mutex(a_1, a_2, i)$) ssi :*

1. *les actions ne sont pas indépendantes (cf. définition 2.1.7) ou*
2. *les actions ont en précondition des fluents mutex au niveau $i-1$ précédent (elles ne peuvent donc pas être déclenchées en même temps) : $\exists(p, q) \in prec(a_1) \times prec(a_2)/mutex(p, q, i-1)$.*

Deux fluents p et q d'un niveau i du graphe sont mutex ssi tous les couples d'actions qui les produisent à ce niveau sont mutex.

Le graphe est étendu niveau par niveau jusqu'à ce que les fluents du but soient présents dans le niveau courant et qu'il n'existe pas de mutex entre eux.

Définition 4.2.2 (Stabilisation du graphe [Blum et Furst, 1997]) *La stabilisation du graphe est atteinte lorsque deux niveaux successifs contiennent les mêmes actions et fluents, et les mêmes mutex d'actions et de fluents.*

Si la stabilisation du graphe est atteinte avant l'obtention des fluents du but sans mutex alors la construction du graphe se termine par un échec et le problème n'admet pas de solution.

Exemple 4.2.3 (Graphe de planification) *Considérons l'exemple 4.1.1 décrit à la page précédente. La figure 4.2 représente le graphe de planification construit jusqu'au niveau 3, premier niveau dans lequel tous les buts apparaissent sans mutex. Les fluents sont représentés par des nœuds rectangulaires, les actions par des nœuds ovales, les arcs représentent les mutex entre les fluents (en vert) et entre les actions (en bleu), et les flèches représentent les relations entre les fluents et les actions.*

Le graphe de planification comporte des actions qui ne sont pas susceptibles d'appartenir à un plan-solution de 3 niveaux. L'élimination de ces actions, des fluents et des arcs reliés à ces actions, produit un graphe, appelé graphe réduit, de largeur moins importante.

Définition 4.2.4 (Graphe réduit) *Un graphe réduit est un graphe de planification dans lequel, en partant des buts présents au niveau k, on supprime toutes les actions du même niveau (et leurs arcs de précondition) qui ne permettent pas l'obtention de ces fluents ; on recommence ensuite au niveau $k-1$ en supprimant toutes les actions qui ne permettent pas l'obtention de préconditions des actions du niveau k qui ont été conservées et ainsi de suite jusqu'au niveau 1.*

4.3 Codage d'un graphe en WCSP

Le codage, par un WCSP, d'un plan issu d'un graphe de planification avec actions valuées nécessite plusieurs modifications de la méthode initialement proposée par [Do et Kambhampati, 2001] pour le codage d'un graphe de planification classique en CSP dynamique (DCSP, Dynamic Constraint Satisfaction Problem) [Mittal et Falkenhainer, 1990]. Un DCSP (ou encore appelé CSP conditionnel) contient, initialement, un sous-ensemble de variables actives et la résolution consiste à trouver une affectation pour toutes les variables actives telle qu'elle soit consistante avec l'ensemble des contraintes. Dans leur codage, Do et Kambhampati représentent chaque

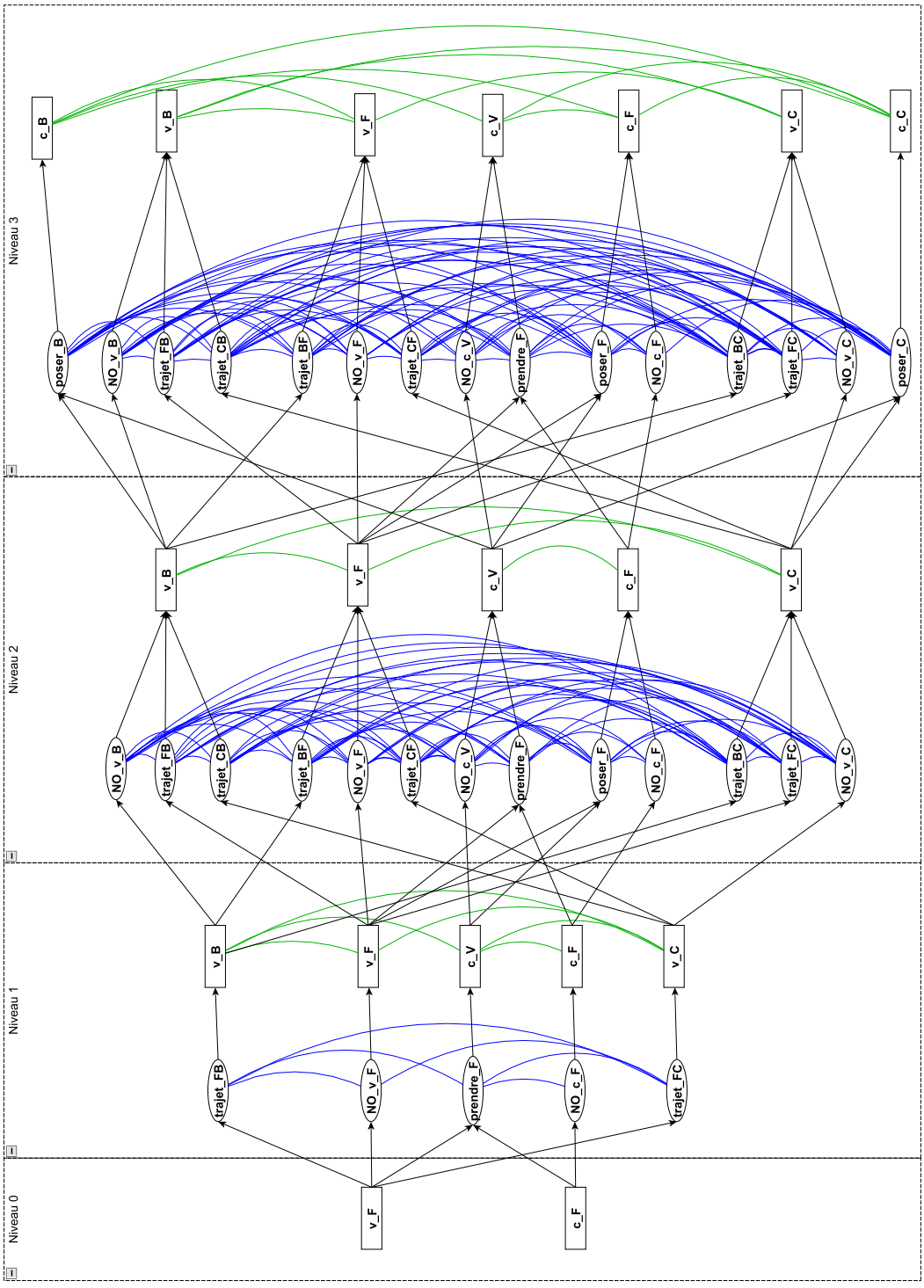


FIG. 4.2 – Graphe de planification de 3 niveaux de l'exemple 4.1.1.

fluent f à un niveau i du graphe de planification par une variable dans le CSP ; l'ensemble des actions qui ajoutent f constitue son domaine et les mutex produits durant la construction du graphe sont assimilés à des contraintes du CSP. L'affectation des valeurs aux variables est un processus dynamique car chaque affectation d'une variable à un niveau i active d'autres variables du niveau précédent (les préconditions de l'action choisie). Ainsi, une solution est une affectation d'une valeur (une action) à chaque variable (fluent) satisfaisant l'ensemble de contraintes (mutex).

Notre codage en un WCSP d'un graphe de planification réduit nécessite sept étapes :

1. *Réécriture du graphe :* en partant du dernier niveau du graphe réduit, nous renommons chaque fluent en f_i et nous numérotons chaque action en j, i et j étant des entiers strictement positifs. L'ordre dans lequel cette numérotation est réalisée est important car il influence l'efficacité de la résolution du WCSP associé (cf. section 3.6, page 52 et sous-section 4.4.3, page 70).
2. *Création des variables et des domaines :* pour chaque fluent n'appartenant pas à l'état initial, nous créons une variable dont le domaine est l'ensemble des actions qui produisent ce fluent. Pour les fluents n'appartenant pas au but, nous ajoutons la valeur -1 pour représenter sa non-activation éventuelle.
3. *Traduction des mutex entre fluents :* pour tous les fluents mutex f_i et f_j, l'exclusion mutuelle est traduite par une contrainte qui exprime le fait que f_i et f_j ne doivent pas être activés en même temps : $(f_i = -1) \vee (f_j = -1))$
4. *Traduction des mutex entre actions :* pour toutes les actions mutex a et b d'effets respectifs f_i et f_j $(f_i \neq f_j)$, l'exclusion mutuelle est traduite par une contrainte qui exprime le fait que f_i et f_j ne peuvent pas être activés en même temps par les actions a et b : $\neg((f_i = a) \wedge (f_j = b))$.
5. *Traduction des arcs d'activité :* l'activation d'un fluent f_i produit par une action a_i entraîne l'activation des préconditions de cette action. Ceci se traduit par la contrainte d'activité : $\forall f_j \in prec(a), (f_i = a) \Rightarrow (f_j \neq -1)$.
6. *Traduction du coût des actions :* pour chaque valeur a, nous ajoutons une contrainte unaire pour chaque affectation $f = a$ correspondant au coût de l'action a. Les Noops ont un coût nul.
7. *Prise en compte des ajouts multiples :* lorsqu'une action a produit plusieurs fluents $f_i \in effet(a)$, le coût de cette action est compté plusieurs fois si plusieurs d'entre eux sont utiles à la résolution du WCSP. Pour résoudre ce problème, nous créons un fluent intermédiaire f^{int} de domaine $\{a, -1\}$. Ainsi l'action a est remplacée par une action fictive a^{int} (de coût nul) dans les domaines des fluents f_i. Nous ajoutons également une contrainte d'activité $(f_i = a^{int}) \Rightarrow (f^{int} = a)$ entre le fluent f^{int} et chaque fluent $f_i \in eff(a)$.

Dans les schémas qui suivront, les arcs de coût infinis seront simplement représentés par des arcs non valués. Les variables contiendront les différentes valeurs de leur domaine et chaque contrainte unaire de coût non nul sera inscrite à gauche de la valeur associée.

Exemple 4.3.1 (WCSP issu d'un graphe de planification) *Considérons le graphe de planification de l'exemple 4.1.1 et représenté dans la figure 4.2. Le graphe réduit de niveau 3 et le*

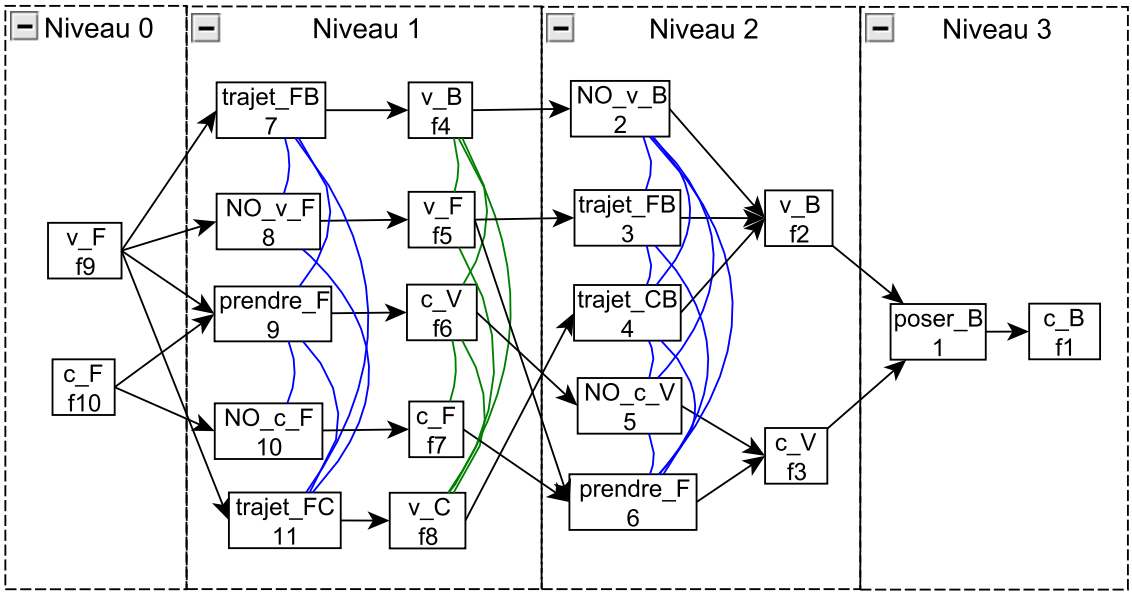


FIG. 4.3 – Graphe de planification après réduction et renommage de l'exemple 4.1.1.

renommage correspondant sont donnés dans la figure 4.3. Le codage du graphe en WCSP est présenté dans la figure 4.4. Les mutex entre les fluents et les actions, et les contraintes d'activités sont représentés par des contraintes binaires de coûts infinis. Ainsi tous les arcs entre les valeurs de domaines différents ont un coût infini. La section suivante décrira la résolution d'un tel WCSP.

4.4 Résolution de WCSP

Nous présenterons dans cette section notre méthode pour résoudre des WCSP issus d'un graphe de planification de k niveaux. Nous commencerons par donner les caractéristiques des WCSP utilisés lors des expérimentations. En effet, pour obtenir une résolution efficace, nous testerons différents algorithmes de filtrage et différents ordonnancements pour la sélection des variables et des valeurs du WCSP. Nous terminerons cette section en montrant comment résoudre le WCSP de l'exemple de la section précédente.

4.4.1 Caractéristiques des WCSP utilisés dans les tests

Nous effectuerons un grand nombre de tests sur des WCSP générés par le planificateur GP-WCSP et son extension GP-WCSP* que nous présenterons dans le chapitre 6. En effet, basé sur GP-WCSP, GP-WCSP* génère de plus gros WCSP que GP-WCSP en permettant d'obtenir une solution k-optimale tel que k est le nombre de niveaux minimum pour garantir l'obtention d'une solution de coût optimal. Cependant, la résolution des WCSP par ces planificateurs est simplifiée par la prise en compte du coût d'un plan-solution précédemment obtenu. Lors de ces tests, pour connaître la taille des WCSP réellement résolus, nous avons donc ignoré la valeur de

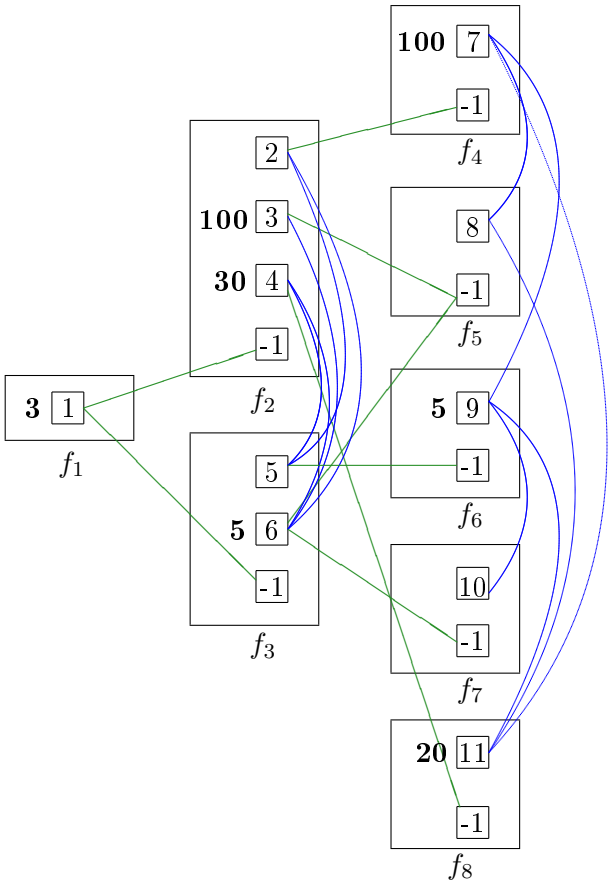


FIG. 4.4 – WCSP obtenu à partir du graphe de planification de la figure 4.3 ; les coûts unaires sont inscrits à côté des valeurs des domaines et les arcs relient des valeurs incompatibles dans des domaines distincts.

cette borne de coût qui permet de couper des branches de l'arbre de recherche. Sauf mention contraire, nous utilisons la version TOULBAR2.0.6 de la librairie TOULBAR2[1] [de Givry *et al.*, 2005] pour résoudre les WCSP issus des graphes de planification. Et les tests ont été effectués sur un PC Intel® Pentium® 4 cadencé à 3GHz avec un cache de 1024Kbyte, 1Gbyte de mémoire vive et sous Linux Fedora 8.

Le tableau 4.1 décrit les problèmes utilisés et les WCSP résultants : nom du problème, caractéristiques du plus gros WCSP résolu par GP-WCSP* (nombre de niveaux du graphe de planification, nombre de variables du WCSP, moyenne géométrique de la taille des domaines, nombre de contraintes et taille de l'espace de recherche).

4.4.2 Algorithmes de propagation de contraintes

Nous avons présenté, dans le chapitre 3, plusieurs algorithmes de propagation de contraintes très efficaces pour la résolution de WCSP. Ils travaillent notamment en augmentant la valeur

[1] http ://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro

Problème	Caractéristiques du plus gros WCSP				
	k	Variables	*moyenne* d_i	Contraintes	Espace de recherche
blocks01	6	179	2,74	2 148	2,04E+078
blocks02	20	747	2,82	8 160	3,06E+336
blocks03	6	147	2,70	1 483	2,96E+063
blocks04	20	1 157	2,78	16 579	8,69E+513
blocks05	20	1 273	2,79	19 033	5,23E+566
blocks06	20	1 023	2,77	14 071	6,63E+452
blocks07	20	1 703	2,74	30 576	9,01E+744
blocks08	10	781	2,72	14 457	6,07E+338
blocks09	20	1 287	2,73	20 546	1,50E+561
blocks10	20	1 515	2,70	27 279	1,95E+652
logistics01	13	265	3,16	2 941	2,47E+132
logistics02	12	245	3,14	2 624	5,05E+121
logistics03	15	330	3,21	3 796	2,00E+167
logistics04	12	284	3,13	3 353	5,31E+140
logistics05	12	286	3,15	3 342	2,87E+142
logistics06	7	98	2,76	646	1,81E+043
storage01	3	11	2,02	23	2,30E+003
storage02	3	11	2,02	23	2,30E+003
storage03	6	118	2,57	981	2,31E+048
driverlog01	16	495	3,59	7 871	3,27E+274
driverlog02	12	454	3,68	8 554	8,13E+256
driverlog03	13	490	3,68	9 664	1,82E+277
driverlog04	11	540	3,67	12 396	8,21E+304
driverlog07	9	601	3,77	18 219	1,04E+346
driverlog08	9	624	3,74	20 113	1,32E+357
zenotravel01	2	10	2	20	1,02E+003
zenotravel02	17	1 100	2,71	9 046	5,50E+256
zenotravel03	6	476	2,79	4 572	1,10E+212
satellite01	20	235	5,17	2 994	4,91E+167
satellite02	20	326	5,31	5 268	2,65E+236
satellite03	16	417	5,86	6 725	8,56E+319
satellite04	13	396	6,58	7 797	6,82E+323
depot01	16	818	3,05	12 675	1,01E+395
depot02	13	1 323	2,97	29 518	4,47E+058

TAB. 4.1 – Caractéristiques des WCSP utilisés dans les tests.

de la borne inférieure du coût de la solution. Sur des problèmes aléatoires et des problèmes spécifiques, [de Givry *et al.*, 2005] comparent les algorithmes NC, FDAC et EDAC, appliqués à chaque nœud de l'arbre de recherche, et montrent que EDAC permet d'améliorer significativement le temps de résolution du WCSP, même si le temps de calcul à chaque nœud est augmenté.

Pour déterminer les techniques les plus efficaces pour notre méthode, nous comparons les algorithmes NC, FDAC et EDAC, appliqués à chaque nœud de l'arbre de recherche, sur des WCSP issus des problèmes de planification. Nous comparerons également l'apport de l'établissement de VAC en prétraitement sur ces mêmes WCSP.

Comparaison entre FDAC et NC

Nous avons commencé par comparer le temps de résolution des WCSP en établissant la cohérence d'arc totalement directionnelle (FDAC) par rapport à l'établissement de la cohérence de nœud (NC). Pour ce test, nous avons utilisé la version TOOLBAR3.0 du solveur TOULBAR2 car elle implémente l'algorithme NC. Le graphe de la figure 4.5 représente, avec une échelle logarithmique, le temps de résolution des algorithmes FDAC et NC dans les domaines *Blocks*, *Logistics*, *Storage*, *Driverlog* et *Zeno*. Nous pouvons déduire de ce graphe que les temps de résolution avec FDAC et NC évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé ($R^2 = 0,917$) et son équation est $NC = 7,063 \times FDAC^{1,112}$. Cette courbe nous confirme que dans tous les problèmes testés FDAC est plus rapide que NC avec un gain important.

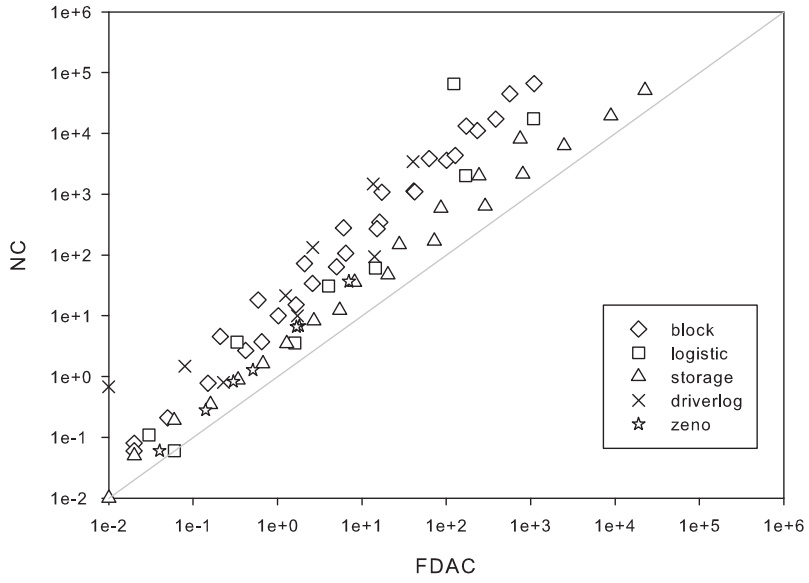


FIG. 4.5 – Comparaison des temps de résolution des WCSP utilisant FDAC ou NC (en secondes).

Nous avons également étudié le rapport entre les temps de résolution de NC et de FDAC et son évolution en fonction de la difficulté des problèmes. Pour cela, nous avons pris le temps de résolution de FDAC comme indice de difficulté des problèmes. Le graphe 4.6 donne, avec une échelle logarithmique, le rapport entre les temps de résolution de NC par rapport à FDAC en fonction du temps de résolution de FDAC. Nous pouvons observer que le rapport NC/FDAC

est compris entre 1 et 110, ce qui signifie que FDAC est toujours plus rapide que NC et jusqu'à 110 fois. L'ordonnée du point moyen correspond à la moyenne des rapports NC/FDAC et nous pouvons en déduire que NC est en moyenne $18,32$ fois plus lent que FDAC sur les problèmes testés. La répartition non homogène des points dans le graphe nous montre que, contrairement aux temps de résolution de FDAC et NC, l'écart entre ces temps de résolution n'évolue pas de manière homogène avec l'augmentation de la difficulté des problèmes.

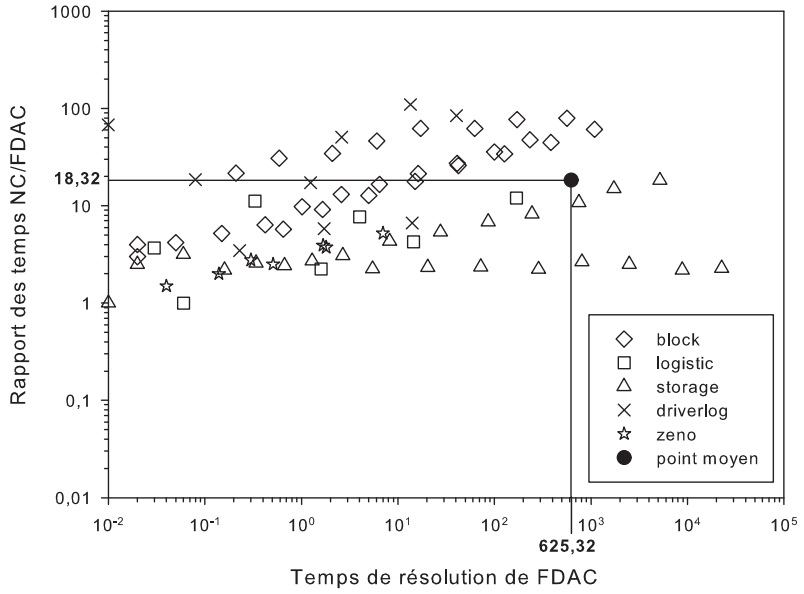


FIG. 4.6 – Rapport des temps de résolution de NC par rapport à FDAC en fonction du temps de résolution de FDAC.

Nous pouvons donc conclure de ces premiers tests que FDAC améliore de façon significative l'extraction d'une solution optimale des WCSP issus de la planification par rapport à NC.

Comparaison entre EDAC et NC

L'algorithme de cohérence d'arc directionnelle et existentielle (EDAC) est un autre algorithme de résolution. Nous avons comparé le temps de résolution des WCSP en établissant EDAC par rapport à l'établissement de la cohérence de nœud (NC). Pour ce test, nous avons utilisé la version TOOLBAR3.0 du solveur TOULBAR2 car elle implémente l'algorithme NC. Le graphe de la figure 4.7 représente, avec une échelle logarithmique, le temps de résolution des algorithmes EDAC et NC dans les domaines *Blocks*, *Logistics*, *Storage*, *Driverlog* et *Zeno*. Nous pouvons déduire de ce graphe que les temps de résolution avec EDAC et NC évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé ($R^2 = 0,9221$) et son équation est $NC = 6,5362 \times EDAC^{1,1316}$. Cette courbe nous confirme que dans tous les problèmes testés EDAC est plus rapide que NC avec un gain important.

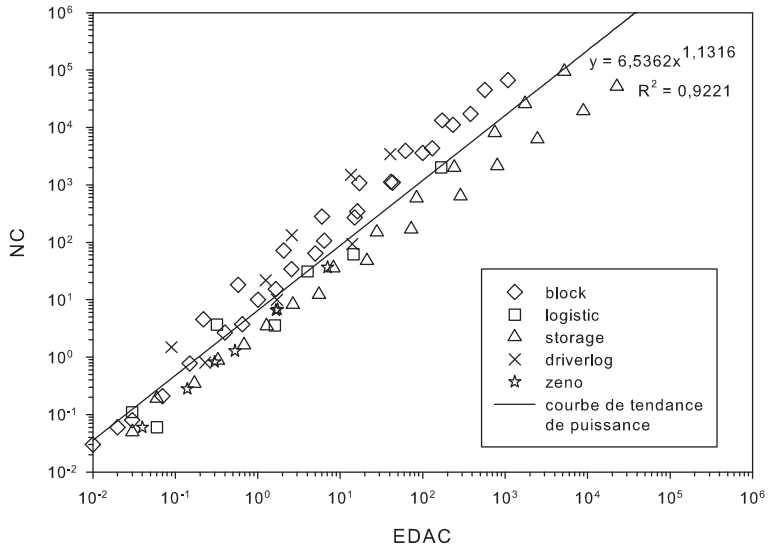


FIG. 4.7 – Comparaison des temps de résolution des WCSP utilisant EDAC ou NC (en secondes).

Nous avons également étudié le rapport entre les temps de résolution de NC et de EDAC et son évolution en fonction de la difficulté des problèmes. Pour cela, nous avons pris le temps de résolution de EDAC comme indice de difficulté des problèmes. Le graphe 4.8 donne, avec une échelle logarithmique, le rapport entre les temps de résolution de NC par rapport à EDAC en fonction du temps de résolution de EDAC. Nous pouvons observer que le rapport NC/EDAC est compris entre 1 et 110, ce qui signifie que NC est toujours plus lent que EDAC et jusqu'à 110 fois. L'ordonnée du point moyen correspond à la moyenne des rapports $NC/EDAC$ et nous pouvons en déduire que NC est en moyenne $17,62$ fois plus lent que EDAC sur les problèmes testés. La répartition non homogène des points dans le graphe nous montre que, contrairement aux temps de résolution de EDAC et NC, l'écart entre ces temps de résolution n'évolue pas de manière homogène avec l'augmentation de la difficulté des problèmes.

Nous pouvons en conclure que EDAC améliore significativement l'extraction d'une solution optimale des WCSP issus de la planification par rapport à NC.

Comparaison entre FDAC et EDAC

Comme les algorithmes FDAC et EDAC sont tous les deux plus rapides que NC, nous les avons comparé afin de trouver l'algorithme le plus performant pour les WCSP issus d'un graphe de planification. Le graphe de la figure 4.9 représente, avec une échelle logarithmique, le temps de résolution des algorithmes FDAC et EDAC dans les domaines *Blocks*, *Logistics*, *Storage*, *Driverlog* et *Zeno*. Nous pouvons déduire de ce graphe que les temps de résolution avec FDAC et EDAC évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé ($R^2 = 0,9987$) et son équation est $EDAC = 1,0275 \times FDAC^{1,0001}$. Cette courbe nous confirme que FDAC est légèrement plus rapide que EDAC sur les problèmes

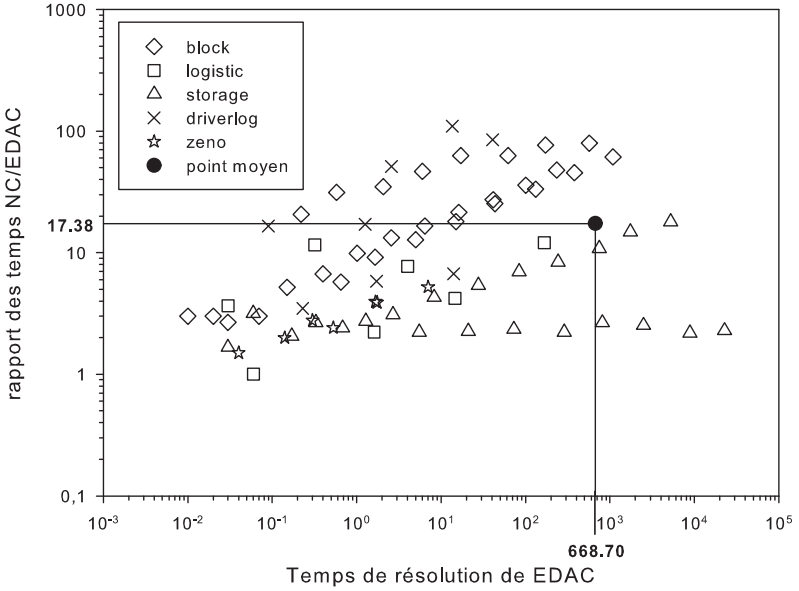


FIG. 4.8 – Rapport des temps de résolution de NC par rapport à EDAC en fonction du temps de résolution de EDAC.

testés.

Nous avons également étudié le rapport entre les temps de résolution de FDAC et de EDAC et son évolution en fonction de la difficulté des problèmes. Pour cela, nous avons pris le temps de résolution de FDAC comme indice de difficulté des problèmes. Le graphe 4.10 donne, avec une échelle logarithmique pour l'axe des abscisses, le rapport entre les temps de résolution de EDAC par rapport à FDAC en fonction du temps de résolution de FDAC. Nous pouvons observer que le rapport EDAC/FDAC est compris entre $0,95$ et $2,88$, ce qui signifie que sur certains problèmes FDAC est légèrement plus lent que EDAC, et que sur les autres problèmes EDAC nécessite plus de temps que FDAC et au maximum $2,88$ fois. De plus, la plupart des points du graphe ont une ordonnée comprise entre $0,95$ et $1,05$, ce qui signifie qu'un algorithme n'est pas significativement plus performant que l'autre et que cet écart ne dépend pas de la difficulté des problèmes. L'ordonnée du point moyen qui correspond à la moyenne des rapports $EDAC/FDAC$ nous permet de déduire qu'en moyenne EDAC nécessite 4% de temps de calcul en plus que FDAC sur les problèmes testés.

Nous pouvons conclure de ces tests que l'algorithme FDAC est l'algorithme le plus performant, parmi ceux testés à chaque nœud de l'arbre de recherche, pour résoudre des WCSP issus d'un graphe de planification. Nous allons maintenant chercher à améliorer la résolution de FDAC lors de la phase de prétraitement.

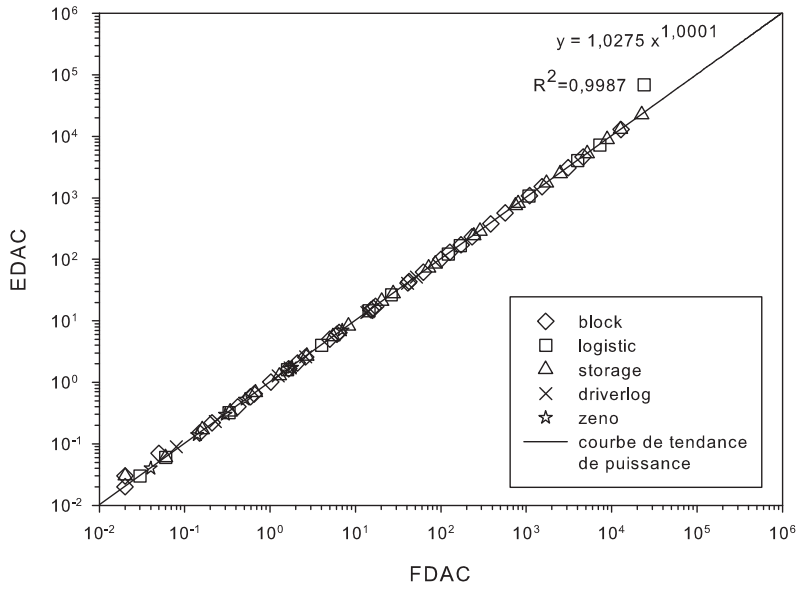


FIG. 4.9 – Comparaison des temps de résolution des WCSP utilisant EDAC ou FDAC (en secondes).

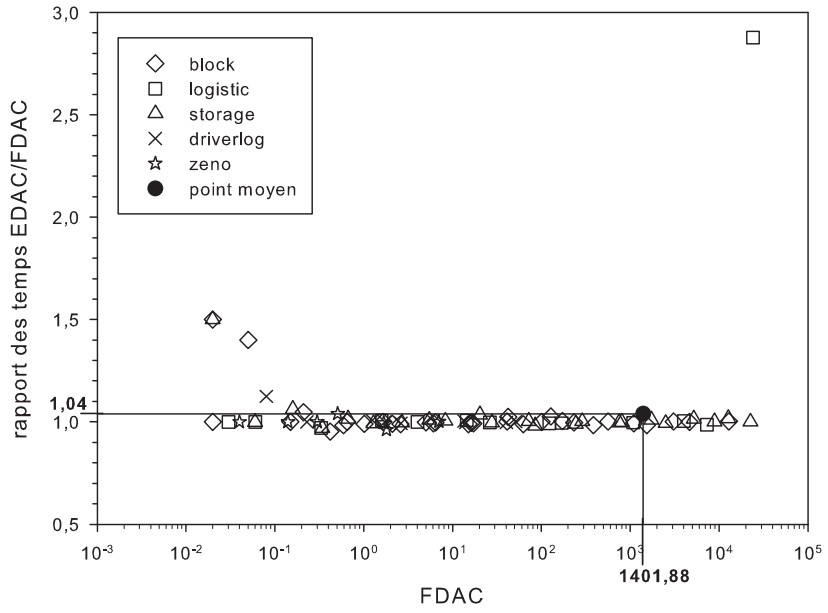


FIG. 4.10 – Rapport des temps de résolution de EDAC par rapport à FDAC en fonction du temps de résolution de FDAC.

Comparaison entre FDAC et VAC

Récemment, la technique de cohérence locale VAC a été introduit pour améliorer la valeur de la borne inférieure $c_\emptyset$ au coût de la solution. Nous avons testé l'apport de VAC en prétraitement car sa résolution nécessite des temps de calcul trop important pour être utilisé à chaque nœud de l'arbre de recherche. Pour ces tests, nous avons utilisé la version TOULBAR2.0.6 du solveur TOULBAR2 qui implémente les algorithmes FDAC et VAC. Nous avons donc comparé le gain obtenu par l'établissement de VAC par rapport à l'optimum qui est le coût de la solution optimale. Le graphe de la figure 4.11 représente, avec une échelle logarithmique pour l'axe des abscisses, le rapport du gain de $c_\emptyset$ par rapport à l'optimum, en fonction du temps de résolution en établissant VAC (noté VAC). On constate que dans 42% des problèmes la valeur de $c_\emptyset$ n'est pas modifiée (rapport nul) et que dans les autres problèmes la valeur de $c_\emptyset$ est augmentée jusqu'à atteindre une fois l'optimum. L'ordonnée du point moyen montre que la valeur de $c_\emptyset$ est augmentée en moyenne de 19% par rapport à l'optimum sur les problèmes testés.

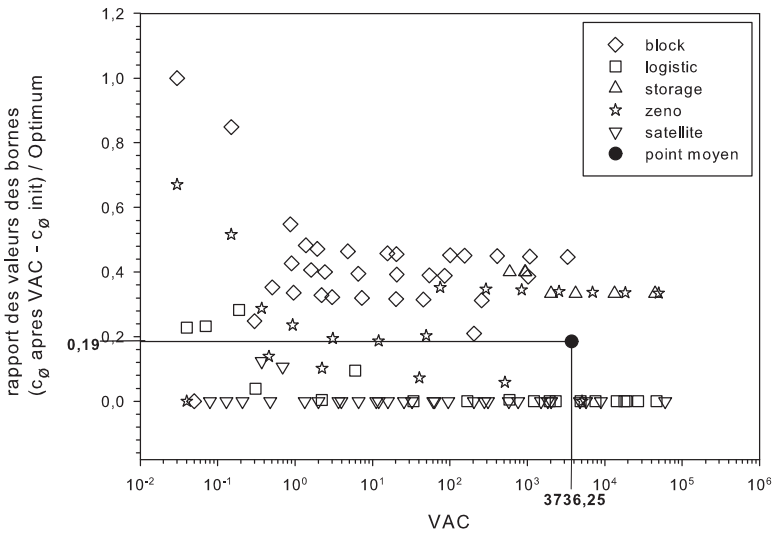


FIG. 4.11 – Rapport des gains de $c_\emptyset$ par rapport à l'optimum en établissant VAC, en fonction du temps de résolution en établissant VAC.

Le graphe de la figure 4.12 représente, avec une échelle logarithmique, le temps de résolution en utilisant VAC (noté VAC) et sans son utilisation (noté $sansVAC$). Nous pouvons déduire de ce graphe que les temps de résolution en utilisant ou non VAC évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé ($R^2 = 0,9739$) et son équation est $sansVAC = 0,3557 \times VAC^{1,0139}$. Cette courbe nous montre que l'utilisation de VAC augmente le temps de résolution pour la plupart des problèmes testés.

Nous avons également étudié le rapport entre les temps de résolution selon l'utilisation de VAC et son évolution en fonction de la difficulté des problèmes. Pour cela, nous avons pris le temps de résolution en utilisant VAC comme indice de difficulté des problèmes. Le graphe 4.13

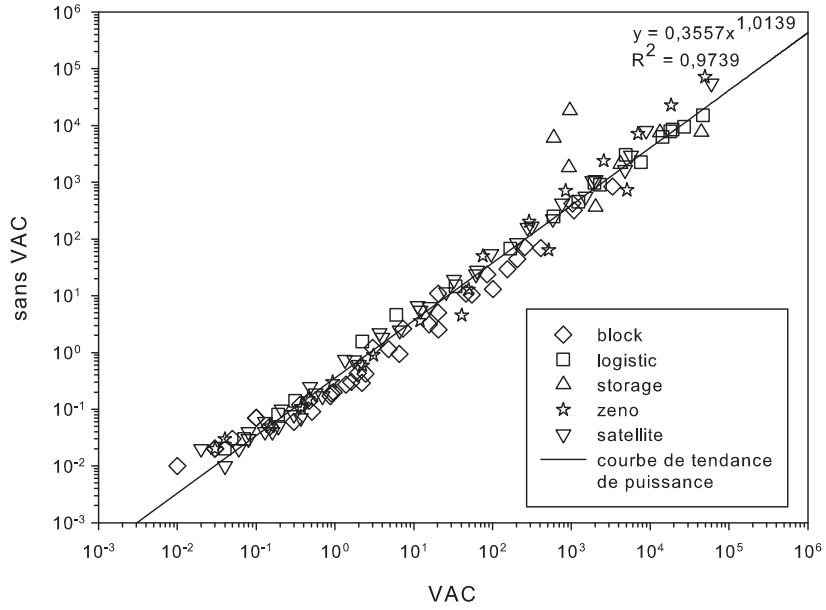


FIG. 4.12 – Comparaison des temps de résolution des WCSP utilisant VAC en prétraitement ou sans son utilisation (en secondes).

donne, avec une échelle logarithmique pour l'axe des abscisses, le rapport entre les temps de résolution en utilisant VAC par rapport à son absence en fonction du temps de résolution avec VAC. Nous pouvons observer que ce rapport est compris entre $0,05$ et $8,9$, ce qui signifie que sur certains problèmes l'utilisation de VAC permet d'améliorer la résolution, mais que sur les autres problèmes VAC nécessite plus de temps de calcul et au maximum $8,9$ fois. L'ordonnée du point moyen qui correspond à la moyenne des rapports nous montre que l'ajout en prétraitement de VAC nécessite $3,25$ fois plus de temps de calcul sur les problèmes testés.

Les trois instances pour lesquels l'utilisation de VAC est réellement avantageuse appartiennent au même problème *storage*03 et sont issues d'un graphe de planification comportant entre 15 et 17 niveaux. Ce succès s'explique par le faible écart entre $c_\emptyset$ (0 initialement, 2 après VAC) et la valeur optimum (qui vaut 5). Dans le plus gros WCSP parmi les trois (1034 variables ; 2,82 de moyenne géométrique pour la taille des domaines ; 12649 contraintes ; 7,03E+464 de taille de l'espace de recherche), l'augmentation de $c_\emptyset$ permet d'explorer seulement 206679 nœuds au lieu de plus de 36,7 millions. Cette instance est alors résolue en moins de 950 secondes alors qu'elle nécessitait plus de 18360 secondes sans utiliser VAC. Cependant, les cas pour lesquels l'utilisation de VAC est avantageuse n'apparaissent que dans ce problème et une instance du domaine *Zeno* car dans la plupart des WCSP testés, le temps d'établissement de VAC est supérieur au gain obtenu lors de la résolution du problème lui-même.

Nous pouvons conclure de ces tests que l'utilisation de VAC en prétraitement dégrade le temps de résolution des WCSP issus d'un graphe de planification.

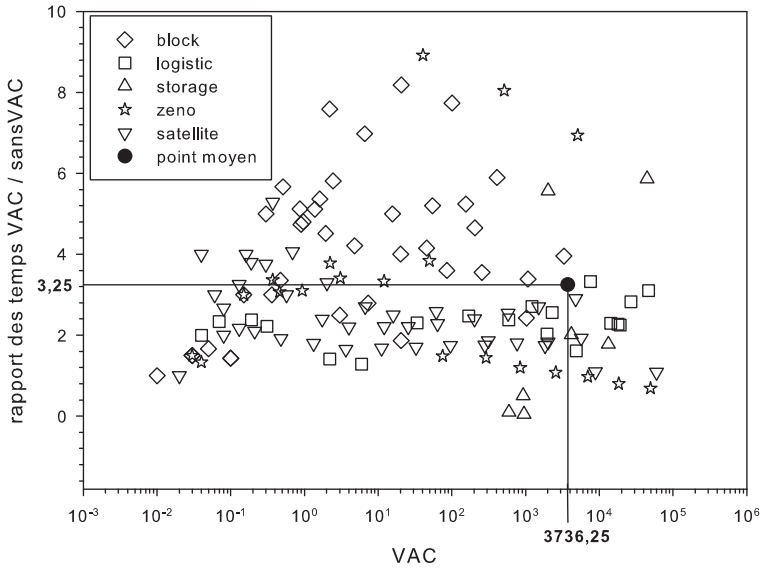


FIG. 4.13 – Rapport des temps de résolution en utilisant VAC par rapport à son absence en fonction du temps de résolution en établissant VAC.

Synthèse de la comparaison des algorithmes

Nous avons vu, dans cette sous-section, que les algorithmes FDAC et EDAC étaient en moyenne 17 fois plus rapides que NC. Nous avons alors comparé FDAC et EDAC entre eux et déterminé que EDAC étaient en moyenne 6% plus lent que FDAC. Nous avons également montré que l'ajout de l'algorithme VAC en prétraitement dégradait le temps de résolution. Ces résultats expérimentaux nous amènent à utiliser FDAC à chaque nœud de l'arbre de recherche pour résoudre efficacement les WCSP issus du graphe de planification. Nous pouvons encore améliorer la résolution des WCSP en utilisant de bonnes heuristiques d'ordonnancement comme le montre la sous-section suivante.

4.4.3 Heuristiques d'ordonnancement

Après avoir trouvé un bon algorithme de résolution, FDAC dans notre cas, nous pouvons utiliser des heuristiques de sélection des variables et des valeurs pour encore améliorer les temps de résolution en guidant plus efficacement les stratégies de recherche.

Nous avons donc testé plusieurs ordonnancements de variables comme un ordonnancement aléatoire, selon le nombre de contraintes, la taille du domaine et une combinaison de ces trois heuristiques. Pour ces tests, nous avons utilisé la version TOOLBAR3.0 du solveur TOULBAR2 car elle implémente différents ordonnancements de variables et de valeurs. Ces tests ont montré l'influence de l'ordonnancement des variables sur l'efficacité de résolution. Nous comparons ici les deux heuristiques d'ordonnancement de variables qui ont donné les meilleurs résultats dans nos tests : la première est issue des techniques WCSP, la seconde est couramment utilisée dans

des méthodes d'extraction de solution du graphe de planification :

- l'heuristique "2-sided Jeroslow-like" [de Givry *et al.*, 2003, page 7] (noté *Jeroslow*) a obtenu de bonnes performances pour résoudre des WCSP aléatoires : les variables sont ordonnées selon un ratio entre la taille de leur domaine et la somme des moyennes des coûts unaires et binaires des contraintes qui sont liées à cette variable.
- l'heuristique "level-based" (noté *LevelBased*) a montré son efficacité sur des benchmarks de planification (sans coût) : elle consiste à ordonner les variables selon la taille de leur domaine puis selon l'ordre inverse de leur niveau d'apparition dans le graphe de planification [Cayrol *et al.*, 2001 ; Do et Kambhampati, 2001].

Le graphe de la figure 4.14 représente, avec une échelle logarithmique, le temps de résolution des WCSP en utilisant les heuristiques "2-sided Jeroslow-like" ou "level-based" dans les domaines *Blocks*, *Logistics*, *Storage*, *Driverlog*, *Zeno* et *Satellite*. Nous pouvons déduire de ce graphe que les temps de résolution avec ces deux heuristiques évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé ($R^2 = 0,7693$) et son équation est $LevelBased = 3,6738 \times Jeroslow^{1,3959}$. Cette courbe nous montre que la plupart des problèmes testés sont résolus plus rapidement en utilisant l'heuristique "2-sided Jeroslow-like" qu'en utilisant "level-based".

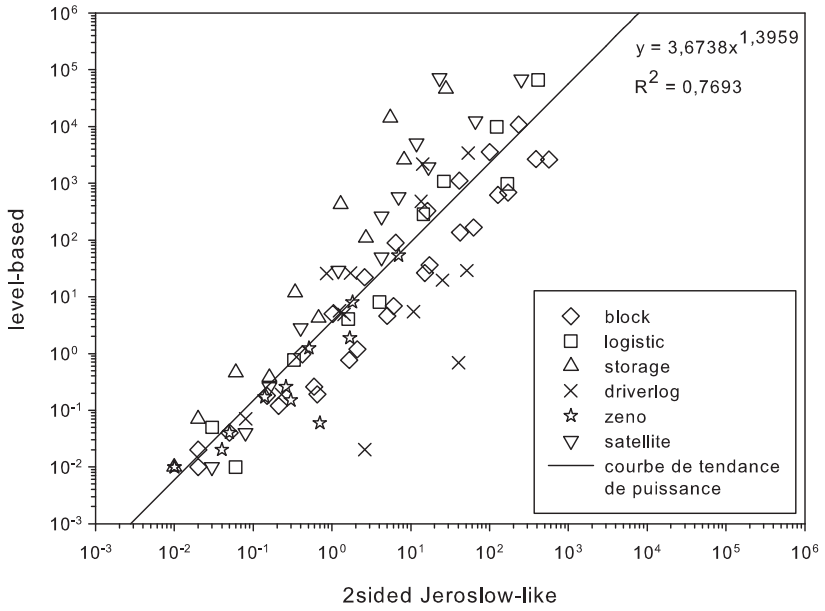


FIG. 4.14 – Comparaison des temps de résolution des WCSP avec les heuristiques "2-sided Jeroslow-like" et "level-based" (en secondes).

Nous avons également étudié le rapport entre les temps de résolution avec les heuristiques "2-sided Jeroslow-like" ou "level-based" et son évolution en fonction de la difficulté des problèmes. Pour cela, nous avons pris le temps de résolution avec l'heuristique "2-sided Jeroslow-like" comme indice de difficulté des problèmes. Le graphe 4.15 donne, avec une échelle logarithmique, le rapport entre les temps de résolution en utilisant "level-based" par rapport à l'utilisation de "2-sided

Jeroslow-like" en fonction du temps de résolution en utilisant *Jeroslow*. Nous pouvons observer que le rapport *LevelBased/Jeroslow* est compris entre $0,008$ et $3079,63$ et que l'utilisation de "2-sided Jeroslow-like" est plus lent que l'utilisation de "level-based" sur les problèmes testés nécessitant moins de 52 secondes pour "2-sided Jeroslow-like". Sur les autres problèmes, l'utilisation de "level-based" nécessite plus de temps de calculs que "2-sided Jeroslow-like" et au maximum $3079,63$ fois. L'ordonnée du point moyen qui correspond à la moyenne des rapports *LevelBased/Jeroslow* nous permet de déduire qu'en moyenne l'utilisation de l'heuristique "level-based" nécessite $116,15$ fois plus de temps de calcul qu'en utilisant l'heuristique "2-sided Jeroslow-like" sur les problèmes testés.

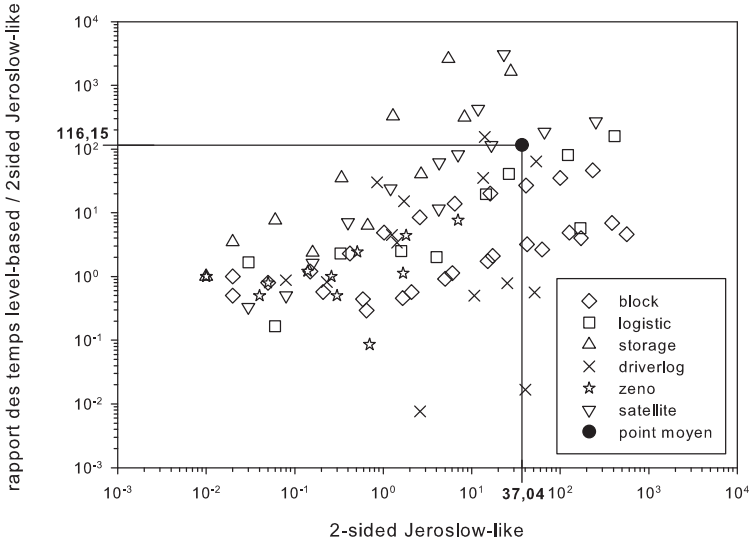


FIG. 4.15 – Rapport des temps de résolution des WCSP avec l'heuristique "level-based" par rapport à "2-sided Jeroslow-like" en fonction du temps de résolution en utilisant "2-sided Jeroslow-like".

Nous avons également comparé différentes heuristiques d'ordonnancement pour l'instantiation des valeurs des domaines :

1. selon l'ordre de leur niveau d'apparition dans le graphe de planification,
2. selon le coût unaire des valeurs et en cas d'égalité un ordonnancement aléatoire,
3. selon le coût unaire des valeurs et en cas d'égalité selon l'ordre de leur niveau d'apparition dans le graphe de planification.

Ces comparaisons n'ont pas apporté une diminution significative de temps de résolution.

En conclusion de ces tests, l'heuristique que nous utiliserons avec FDAC dans la suite des expérimentations est la suivante : sélection des variables avec l'heuristique "2-sided Jeroslow-like" et priorité aux valeurs de plus petit coût unaire. Ces choix nous permettent d'utiliser la version récente TOULBAR2.0.6 du solveur TOULBAR2 dans la suite de nos expérimentations pour résoudre les WCSP.

4.4.4 Exemple de résolution d'un WCSP

Considérons le WCSP issu d'un problème de planification de la figure 4.4 de la section précédente. L'application de FDAC établit une borne inférieure $c_\emptyset = 58$ avant même de commencer une recherche arborescente. Dans cet exemple, la différence entre la borne trouvée par FDAC et celle trouvé par NC est flagrante puisque l'application de NC produit seulement $c_\emptyset = 3$. Ainsi, bien que le problème original contient des coûts non nuls uniquement sur des contraintes unaires, la propagation des coûts sur les contraintes binaires permet l'obtention d'une bien meilleure borne. Le WCSP résultant de l'application de FDAC, pour l'ordonnancement $f_1 \prec f_2 \prec f_3 \prec f_4 \prec f_5 \prec f_6 \prec f_7 \prec f_8$, est donné dans la figure 4.16 où des coûts non nuls sont présents dans les fonctions de coûts unaires et binaires. Dans cet exemple, EDAC produit la même borne que FDAC. La résolution de ce WCSP produit la solution : $\{f_1 \leftarrow 1\,;\ f_2 \leftarrow 3\,;\ f_3 \leftarrow 5\,;\ f_4 \leftarrow -1\,;\ f_5 \leftarrow 8\,;\ f_6 \leftarrow 9\,;\ f_7 \leftarrow -1\,;\ f_8 \leftarrow -1\}$ qui correspond au plan-solution $P_3^* = \langle\{prendre_F\}, \{trajet_{FB}, noop_{c_v}\}, \{poser_B\}\rangle = \langle prendre_F, trajet_{FB}, poser_B\rangle$ de coût $C_3^* = 108$.

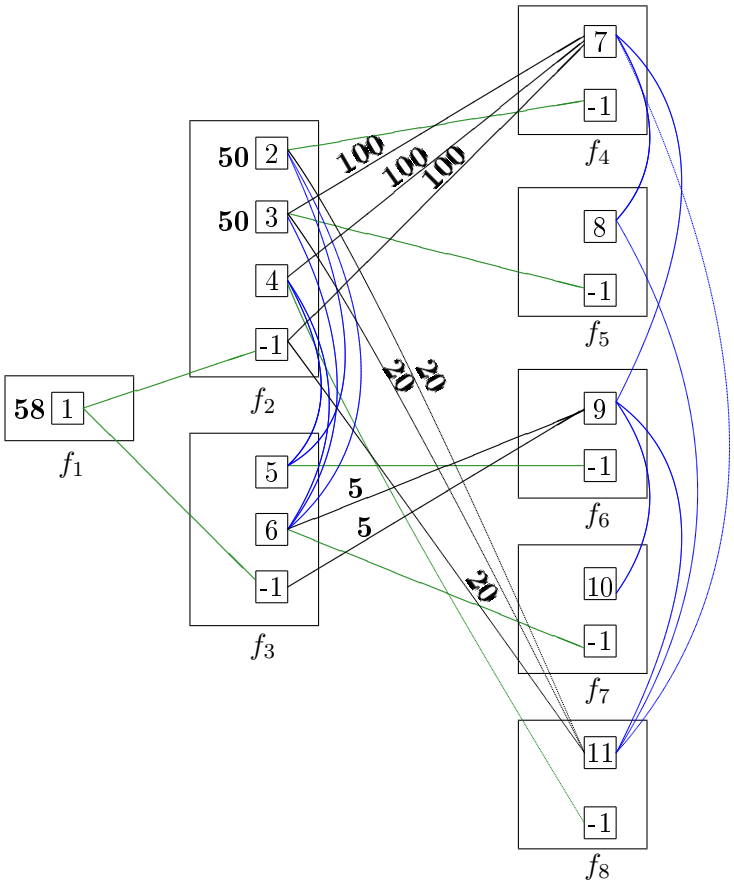


FIG. 4.16 – Résultat de l'application de FDAC sur le WCSP de la figure 4.4.

4.5 GP-WCSP

Nous présenterons dans cette section l'algorithme qui permet l'obtention d'une solution de coût optimal parmi les solutions qui minimisent le nombre de niveaux lorsque le problème admet une solution. Cet algorithme sera implémenté dans le planificateur GP-WCSP et nous terminerons cette section en présentant les résultats obtenus.

4.5.1 Algorithme de GP-WCSP

L'algorithme que nous avons développé (algorithme 5) comporte différentes parties que nous avons détaillées dans les sections précédentes. La première partie de l'algorithme consiste à construire le graphe de planification jusqu'à obtenir un niveau qui contient les buts sans mutex entre eux. Si le graphe est stabilisé avant de l'obtenir, la fonction *constructionGraphe* retourne un échec et l'algorithme s'arrête en signalant que le problème n'a pas de solution. En cas de succès, le graphe réduit est codé en un WCSP équivalent, en suivant le codage décrit dans la section 4.3, puis résolu (cf. section 4.4). Puis, tant qu'aucune solution n'est extraite du WCSP, GP-WCSP poursuit la construction du graphe de planification sur un niveau supplémentaire et cherche à en extraire une solution.

Cette approche donne un planificateur semi-complet puisque, si un plan-solution existe alors l'algorithme s'arrête en retournant un plan-solution P_k^* de coût optimal parmi les solutions qui minimisent le nombre de niveaux. Par contre, comme pour tous les planificateurs de ce type (par exemple SATPLAN [Kautz et Walser, 1999 ; Kautz *et al.*, 2006], GP-CSP [Do et Kambhampati, 2000] ou MAXPLAN [Xing *et al.*, 2006]) le problème de l'arrêt lorsqu'il n'y a pas de solution n'est pas résolu.

4.5.2 Résultats expérimentaux

Tous les tests ont été réalisés à partir des benchmarks détaillés dans la section 2.5 et ont été effectués sur un PC Intel® Pentium® 4 cadencé à 3GHz avec un cache de 1024Kbyte, 1Gbyte de mémoire vive et sous Linux Fedora 8. Les WCSP sont résolus avec la version récente TOULBAR2.0.6 du solveur TOULBAR2. Le tableau 4.17 présente les caractéristiques des problèmes testés avec le planificateur GP-WCSP. Dans ce tableau, le symbole "-" indique un échec de résolution d'un WCSP au bout de 100000 secondes ($\sim$ 27 heures) et "*" indique un dépassement de capacité mémoire lors du stockage du graphe de planification ou lors de la phase de codage du graphe en WCSP.

La première information qui ressort de ces expérimentations est que le nombre de problèmes résolus est plus élevé dans les domaines qui comportent le plus de contraintes. GP-WCSP résout les 15 premiers problèmes du domaine très contraint Blocks et 13 problèmes du domaine Logistic alors que seuls 4 problèmes du domaine Depots sont résolus. En effet, ces contraintes ont une influence sur le nombre d'actions, de fluents et de mutex présents dans le graphe de planification et donc sur la taille des WCSP. D'autre part, les expérimentations montrent que GP-WCSP est capable, dans la plupart des cas, de résoudre des problèmes nécessitant un graphe comportant moins d'une vingtaine de niveaux. Au delà, la taille du graphe est en général trop importante :

Algorithme 5 Résolution k-optimale d'un problème de planification valué $\Pi\langle A, I, B\rangle$

Fonction :

- *constructionGraphe* : construit le graphe de planification jusqu'à l'obtention de tous les buts sans mutex entre eux dans le niveau courant. Si le graphe est stabilisé avant de l'obtenir, alors il retourne un échec (cf. section 4.2).
- $constructionGrapheNiveauSuivant(G_k)$: retourne le graphe de $k+1$ niveaux (cf. section 4.2).
- $codageWCSP(G)$: retourne le wcsp correspondant au codage du graphe G réduit (cf. section 4.3).
- $resolutionWCSP(wcsp, C)$: retourne une solution optimale au *wcsp* et son coût. Si ce coût n'est pas strictement inférieur à C alors il retourne un échec (cf. section 4.4).

Algorithme GP-WCSP :

$G_k \leftarrow constructionGraphe(A, I, B)$
si la construction du graphe G_k est un échec **alors**
  echec // problème sans solution
finsi
$wcsp \leftarrow codageWCSP(G_k)$
$(P_k^*, C_k^*) \leftarrow resolutionWCSP(wcsp, \infty)$
tantque $P_k^* = echec$ **faire**
  $k \leftarrow k+1$
  $G_k \leftarrow constructionGrapheNiveauSuivant(G_{k-1})$
  $wcsp \leftarrow codageWCSP(G_k)$
  $(P_k^*, C_k^*) \leftarrow resolutionWCSP(wcsp, \infty)$
fin tantque
return (P_k^*, C_k^*).

les informations issues de ce graphe ne sont pas accessibles efficacement pour être traduite en WCSP ou la résolution des WCSP de cette taille devient difficile en moins de 27heures.

Les expérimentations effectuées montrent que le temps de résolution des WCSP résolus est compris entre 0,01 seconde et 5019 secondes avec une valeur médiane de 11,38 secondes, et, dans 90% des problèmes résolus, la résolution de chacun des WCSP de GP-WCSP est inférieure à une seconde. De plus, ces WCSP, résolus de manière optimale, comportent plusieurs centaines de variables, alors que les précédentes expérimentations faites sur des WCSP aléatoires montraient une limite pratique de 50 variables [Larrosa et Schiex, 2003 ; de Givry *et al.*, 2005 ; Cooper *et al.*, 2007a, 2008b]. Les bonnes performances de la recherche branch-and-bound sur des WCSP issus de problèmes de planification s'expliquent en partie par la structure du graphe de contraintes : les variables sont divisées en niveaux (qui correspondent aux niveaux du graphe de planification) et les contraintes binaires n'existent qu'entre des variables d'un même niveau ou entre deux niveaux adjacents.

Si un WCSP contient L niveaux, alors au niveau $\frac{L}{2}$, i.e. après avoir instancié la moitié des variables, nous pouvons encore appliquer la moitié des contraintes. Dans un WCSP aléatoire, quand la moitié des variables est instanciée, nous ne pouvons appliquer approximativement que le quart des contraintes. Cette idée peut être formaliser dans la définition suivante :

Définition 4.5.1 (WCSP linéairement incrémental) *Un WCSP est linéairement incrémental pour un ordre d'instanciation des variables $X_1, \ldots, X_n$ si pour tout $p \in \{1, \ldots, n\}$, le nombre c_p de contraintes pourtant sur les sous-ensemble de $\{X_1, \ldots, X_n\}$ satisfait $c_p = \frac{c}{n}(p + e(p, n))$, où c est le nombre total de contraintes et $|\frac{e(p,n)}{n}| \to 0$ quand $n \to \infty$.*

Un problème aléatoire n'est pas linéairement incrémental, car dans ce cas :

$$c_p \;=\; \frac{cp(p-1)/2}{n(n-1)/2} \;=\; O(\frac{cp^2}{n^2}).$$

Par contre, dans les problèmes de planification optimale considérés dans cette thèse, en supposant pour simplifier qu'il y a un nombre constant de variables dans chacun des L niveaux, et que L/n et $1/L$ tendent vers 0 quand $n \to \infty$, nous avons, pour p un multiple de n/L :

$$c_p \;=\; \frac{c(\frac{pL}{n} - 1)}{L-1} \;=\; \frac{c}{n}(p \;-\; \frac{n-p}{L-1}) \quad \text{et} \quad \frac{n-p}{n(L-1)} \to 0 \quad \text{quand} \quad n \to \infty$$

et donc le problème est linéairement incrémental. Comme autre exemple, considérons le WCSP d'un graphe planaire de contraintes sous forme d'une grille carrée. Nous pouvons également facilement prouver que le WCSP est linéairement incrémental. La même remarque peut être faite dans les WCSP binaires dont le graphe de contraintes est un graphe aléatoire avec une largeur induite bornée par une constante.

Supposons que nous ayons besoin en moyenne de hc contraintes ($0 \leq h \leq 1$) pour obtenir un coût supérieur ou égal au coût de la meilleure solution trouvée jusque là. Alors la profondeur espérée à laquelle une branche de l'arbre de recherche branch-and-bound pourra être coupée est facilement estimée à $h^{\frac{1}{2}}n + O(1)$ pour un problème aléatoire, et $hn + \circ(n)$ pour un problème linéairement incrémental. Si h est une constante et $h < 1$ alors l'arbre de recherche du problème linairement incrémental est plus petit d'un facteur exponentiel.

4.6 Synthèse

Dans ce chapitre, nous avons présenté une méthode basée sur le codage d'un graphe de planification par un WCSP. Cette méthode permet d'obtenir des plans solutions optimaux en coût pour un nombre de niveaux donné. Lorsque la résolution du WCSP issu du graphe de planification de k niveaux est un succès, l'algorithme GP-WCSP retourne un plan-solution k-optimal. Sinon l'algorithme incrémente le nombre de niveaux du graphe de planification jusqu'à extraire une solution du WCSP. GP-WCSP est donc un algorithme semi-complet s'il existe une solution.

Pour améliorer la résolution des WCSP issus d'un graphe de planification, nous avons comparé l'efficacité des algorithmes de résolution NC, FDAC, EDAC et VAC sur ces WCSP. Nous en avons déduit que, dans notre cadre, l'établissement de FDAC à chaque nœud de l'arbre de recherche est l'algorithme le plus efficace. Une comparaison de différentes heuristiques d'ordonnancement des variables et des valeurs, issues des graphes de planification ou spécifiques aux WCSP a montré que l'heuristique "2-sided Jeroslow-like" est la plus efficace et que l'ordonnancement des valeurs a peu d'influence sur l'efficacité de la résolution pour ces WCSP. Par la suite, nous utiliserons FDAC et l'heuristique "2-sided Jeroslow-like" pour résoudre les WCSP dans nos

algorithmes.

Les résultats du planificateur GP-WCSP montrent que cette approche permet d'obtenir une solution k-optimale. Son extension pour l'obtention d'une solution de coût optimal nécessite l'exploitation de connaissances issues d'une analyse des domaines et des problèmes. Nous avons pour cela développé un ensemble de techniques originales que nous allons maintenant présenter.

Problème	Plan-Solution			Temps de résolution
	k	nb	C_k^*	
block01	6	6	52	11,04
block02	10	10	62	4,45
block03	6	6	38	3,15
block04	12	12	97	0 : 02 : 31
block05	10	10	94	0 : 02 : 21
block06	16	16	112	0 : 06 : 57
block07	12	12	85	0 : 15 : 01
block08	10	10	133	0 : 19 : 51
block09	20	20	135	0 : 52 : 50
block10	20	20	146	1 : 24 : 23
block11	22	22	161	25 : 28 : 16
block12	20	20	157	6 : 21 : 21
block13	18	18	156	15 : 06 : 58
block14	20	20	182	31 : 56 : 20
block15	16	16	130	7 : 25 : 55
block16	*			
logistic01	9	20	149	1,00
logistic02	9	19	139	2,03
logistic03	9	15	146	1,00
logistic04	9	27	235	1,63
logistic05	9	17	129	6,72
logistic06	3	8	52	0,05
logistic07	9	25	179	3,21
logistic08	9	14	124	3,96
logistic09	9	26	246	6,80
logistic10	12	38	405	0 : 46 : 57
logistic11	-			
logistic12	11	31	306	1 : 30 : 08
logistic13	-			
logistic14	11	36	321	0 : 14 : 24
logistic15	10	31	318	0 : 34 : 52
logistic16	-			
storage01	3	3	31	0,01
storage02	3	3	39	0,02
storage03	3	3	33	0,04
storage04	8	8	50	12,44
storage05	6	9	51	3,32
storage06	6	9	62	0,70
storage07	14	14	127	0 : 46 : 13
storage08	8	12	93	0 : 11 : 29
storage09	*			
driverlog01	6	8	61	0,15
driverlog02	7	18	131	12,99
driverlog03	7	12	92	11,58
driverlog04	7	19	180	0 : 01 : 02
driverlog05	8	21	145	0 : 02 : 49
driverlog06	5	11	107	6,92
driverlog07	6	18	147	0 : 01 : 46
driverlog08	7	27	225	0 : 05 : 46
driverlog09	10	24	263	0 : 31 : 56
driverlog10	*			
satellite01	8	9	61	0 : 00 : 10
satellite02	12	13	81	0 : 02 : 06
satellite03	6	13	129	0 : 00 : 33
satellite04	10	22	125	0 : 10 : 44
satellite05	7	23	171	0 : 16 : 13
satellite06	8	27	183	0 : 58 : 11
satellite07	6	28	210	0 : 22 : 54
satellite08	-			
satellite09	6	35	354	4 : 13 : 28
satellite10	-			
zeno01	1	1	19	0 : 00 : 00
zeno02	5	6	53	0 : 00 : 04
zeno03	5	6	50	0 : 01 : 02
zeno04	5	12	81	0 : 00 : 32
zeno05	5	14	83	0 : 01 : 36
zeno06	5	12	85	0 : 02 : 27
zeno07	6	16	134	0 : 01 : 43
zeno08	*			
depot01	5	11	94	0 : 00 : 00
depot02	8	16	124	0 : 01 : 11
depot03	12	29	261	3 : 16 : 45
depot04	*			
depot05	*			
depot06	*			
depot07	10	24	205	2 : 37 : 34
depot08	*			

FIG. 4.17 – Résultats expérimentaux de GP-WCSP. **Plan-solution** : nombre de niveaux (k), nombre d'actions (nb), coût (C_k^*). **Temps de résolution** : la notation h :m :s représente respectivement les heures, minutes et secondes, sinon le temps est exprimé en secondes. Le symbole "-" indique un échec de résolution d'un WCSP au bout de 100000 secondes ($\sim$ 27 heures) et "*" indique un dépassement de capacité mémoire lors du stockage du graphe de planification ou lors de la phase de codage du graphe en WCSP.

Chapitre 5

Analyse des domaines et des problèmes

Il est maintenant acquis qu'une accélération du processus de planification peut être obtenue par l'analyse préalable des domaines et problèmes. De nombreuses approches ont été développées comme par exemple la détection d'invariants [Rintanen, 1998, 2000 ; Kelleher et Cohn, 1992 ; Gerevini et Schubert, 1998, 2000 ; Fox et Long, 1998, 2000 ; Kvarnström, 2002], l'élimination des fluents qui résultent de prédicats statiques [Weld, 1999 ; Hoffmann et Koehler, 1999], la prise en compte de symétries [Fox et Long, 1998, 2000 ; Guere et Alami, 2001], la décomposition des problèmes en niveaux hiérarchiques [Tsuneto *et al.*, 1998 ; Yang *et al.*, 1996 ; Knoblock, 1994 ; Bacchus et Yang, 1994 ; Bundy *et al.*, 1996] ou la sérialisation de sous-buts [Koehler, 1998 ; Koehler et Hoffmann, 2000 ; Porteous *et al.*, 2001 ; Hoffmann *et al.*, 2004].

Nous présenterons dans ce chapitre plusieurs nouvelles méthodes basées sur une analyse des domaines et des problèmes qui vont nous permettre de rechercher plus efficacement une solution de coût optimal. Dans la suite et pour simplifier, nous supposerons qu'un problème de planification optimal est un problème de planification de coût optimal.

La section 5.1 décrira la technique de relaxation la plus couramment utilisée dans le cadre non valué. Nous décrirons ensuite, dans la section 5.2, page 82, différentes stratégies qui permettent de déterminer des ensembles d'actions ou de fluents qui sont indispensables à la résolution du problème ou qui sont indispensables à l'obtention d'une solution de coût optimal [Cooper *et al.*, 2007b, 2008c]. Ces techniques ne considèreront qu'une occurence d'actions ou de fluents dans les solutions et nous montrerons ensuite, dans la section 5.3, page 98, de quelle manière on peut prendre en compte le nombre d'occurences d'actions et de fluents. Nous montrerons finalement, dans la section 5.4, page 115, comment nous pouvons transformer un problème de planification en un autre problème équivalent mais plus facile à résoudre.

5.1 Relaxation

Dans cette section, nous présenterons la méthode de relaxation la plus couramment utilisée qui consiste à ignorer le retraits des actions. Différentes applications en ont été faites dans le cadre non valué que nous pourrons appliquer au cadre valué. Nous étudierons ensuite l'utilisation de cette relaxation pour la résolution de coût optimal. Pour que les notions introduites dans cette

section aient un sens, nous nous plaçerons dans le cas où les problèmes admettent au moins un plan-solution et où toutes les préconditions d'actions ne contiennent aucune négation.

L'analyse d'une version relaxée du problème de planification a été employée par de nombreux planificateurs. La forme de relaxation la plus utilisée, que nous appelerons relaxation classique, est celle qui ignore les retraits des actions, i.e. $del(a) = \emptyset$ pour toutes les actions a du domaine. [Bonet *et al.*, 1997] ont été les premiers à développer cette relaxation dans leur planificateur HSP. Le problème Π y est relaxé en un problème plus simple Π^+ qui ignore tous les retraits des actions et permet d'obtenir une estimation heuristique de la longueur des solutions.

Définition 5.1.1 (Relaxation d'une action [Bonet *et al.*, 1997]) *La relaxation a^+ d'une action a consiste à en ignorer les retraits : $a^+ = \langle prec(a), add(a), cout(a)\rangle$.*

Définition 5.1.2 (Relaxation d'un problème [Bonet *et al.*, 1997]) *Un problème relaxé issu d'un problème $\Pi = \langle A, I, B\rangle$ est un triplet $\Pi^+ = \langle A^+, I, B\rangle$ où $A^+ = \{a^+ | a \in A\}$.*

Définition 5.1.3 (Relaxation d'un graphe [Bonet *et al.*, 1997]) *Un graphe relaxé, noté G^+, est un graphe de planification stabilisé et construit à partir du problème relaxé Π^+. La réduction de ce graphe relaxé, selon la définition 4.2.4, fournit un graphe relaxé et réduit que l'on notera G_r^+ et A_r^+ l'ensemble de ces actions.*

Cette relaxation peut être utilisée comme heuristique pour diriger la recherche d'une solution car c'est une méthode rapide pour détecter des problèmes qui n'ont pas de solution. Nous noterons $A^+[i]$ l'ensemble des actions (qui ne sont pas des noops) et $F^+[i]$ l'ensemble des fluents appartenant à un niveau i du graphe relaxé G_r^+ :

$$A^+[i] = A^+[i-1] \cup \{a \in A | prec(a) \subseteq F^+[i-1]\} \text{ pour } i \geq 1$$

$$F^+[i] = \begin{cases} I & \text{pour } i = 0 \\ F^+[i-1] \cup \bigcup_{a \in A^+[i]} add(a) & \text{pour } i \geq 1 \end{cases}$$

Lemme 5.1.4 *Soit Π un problème de planification. Si le problème relaxé Π^+ n'a pas de solution alors Π n'en a pas non plus [Bonet* et al.*, 1997].*

Cette relaxation peut être utilisée pour déterminer une borne inférieure au nombre d'actions des plans-solutions :

Lemme 5.1.5 *Soit le problème de planification $\Pi = \langle A, I, B\rangle$ et l'ensemble de fluents $F^+[i]$ $(i \geq 0)$ défini comme ci-dessus. Si k est le plus petit entier non négatif tel que $B \subseteq F^+[k]$, alors tous les plans-solutions de Π contiennent au moins k actions [Bonet* et al.*, 1997].*

La relaxation peut également être utilisée pour déterminer des actions qui n'appartiennent à aucun plan-solution. En effet, si le problème $\langle A, I, prec(a)\rangle$ n'a pas de solution alors l'action a n'appartient à aucun plan-solution du problème $\langle A, I, B\rangle$ quel que soit le but B. Dans le cadre d'une résolution de coût optimal, la détection des problèmes qui n'ont pas de plan-solution de coût inférieur ou égal à une constante M peut être utile. Malheureusement, comme le montre le théorème suivant, il n'existe aucun algorithme dont la complexité temporelle soit d'ordre polynomial pour déterminer si un problème relaxé Π^+ a une solution de coût borné par une constante M (en supposant $P \neq NP$).

Théorème 5.1.6 *Considérons la classe $\mathcal{P}^+$ des problèmes de planification optimaux $\Pi = \langle A, I, B\rangle$ dans lequel toutes les actions $a \in A$ n'ont pas de retrait : $del(a) = \emptyset$. Le problème qui consiste à déterminer, pour un problème $\Pi \in \mathcal{P}^+$ et une constante entière non négative M, si Π admet un plan-solution dont le coût n'est pas supérieur à la constante M est NP-difficile.*

Preuve : Nous prouvons ce résultat à partir d'une version de VSAT (Valued Satisfiability), une version pondérée de SATISFIABILITE. Une instance de VSAT consiste, pour un ensemble de n variables $X_1, \ldots, X_n$ et une fonction objectif, à minimiser la somme des fonctions de coûts dont les arguments sont un sous-ensemble des variables $X_1, \ldots, X_n$. Les fonctions de coût peuvent prendre des valeurs dans $\mathbb{R}_{\geq 0} \cup \{\infty\}$. Par exemple, les contraintes inviolables sont codées par les fonctions de coûts dont les valeurs appartiennent à $\{0, \infty\}$. VSAT(Γ) est l'ensemble des instances VSAT dans lesquelles toutes les fonctions de coût appartiennent à Γ.

Soit $\phi : \{0,1\}^2 \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ la fonction

$$\phi(x,y) = \begin{cases} \infty & \text{si } x = y = 0 \\ 0 & \text{sinon} \end{cases}$$

et soit $\psi : \{0,1\} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ la fonction $\psi(x) = x$. Il n'existe que huit classes traitables de fonctions de coût pour VSAT [Cohen *et al.*, 2006]. L'ensemble $\{\phi, \psi\}$ n'est un sous-ensemble d'aucune de ces huit classes traitables, ce qui implique que VSAT($\{\phi, \psi\}$) est NP-difficile.

Pour compléter la preuve, il suffit de donner une réduction polynomiale de VSAT($\{\phi, \psi\}$) vers le problème qui consiste à déterminer, pour un problème $\Pi \in \mathcal{P}^+$ et une constante entière non négative M, si Π admet un plan-solution dont le coût n'est pas supérieur à la constante M.

Soit $\mathcal{I}$ une instance de VSAT($\{\phi, \psi\}$) avec les variables $X_1, \ldots, X_n$. La fonction objectif à minimiser est de la forme :

$$\sum_{(i,j) \in E} \phi(X_i, X_j) \; + \; \sum_{i \in N} \psi(X_i)$$

pour les multi-ensembles $E \subseteq \{1, \ldots, n\}^2$ et $N \subseteq \{1, \ldots, n\}$.

Pour chaque couple $(i,j) \in E$, nous créons un fluent du but b_{ij}. Pour chaque $i \in \{1, \ldots, n\}$, nous créons une action $a_i = \langle \emptyset, \{b_{ij} : 1 \leq j \leq n\} \cup \{b_{ji} : 1 \leq j \leq n\}, c_i \rangle$ où c_i est le nombre de fois que i apparaît dans le multi-ensemble N. Soit $\Pi_{\mathcal{I}}$ le problème de planification $\langle A, \emptyset, B\rangle$, où $A = \{a_i : 1 \leq i \leq n\}$ et $B = \{b_{ij} : (i,j) \in E\}$. Les plans-solutions de $\Pi_{\mathcal{I}}$ dont le coût n'est pas supérieur à M correspondent exactement aux solutions de l'instance $\mathcal{I}$ de VSAT de coût non supérieur à M ($X_i = 1$ ssi a_i appartient au plan-solution). Cette réduction est clairement polynomiale. ∎

Nous avons vérifié expérimentalement la difficulté de résolution optimale de problèmes relaxés. Sur les problèmes testés, le temps de résolution du problème relaxé est supérieur au temps de résolution du problème original. Par exemple, le WCSP issu du graphe de planification de 14 niveaux du 4ième problème du domaine storage (751 variables, 24 valeurs au plus dans les domaines, 9223 contraintes) est résolu en 169,1 secondes alors que le WCSP issu du graphe de planification relaxé de 14 niveaux du même problème (753 variables, 13 valeurs au plus dans les domaines, 6579 contraintes) est résolu en 33568,12 secondes. Ces expérimentations montrent que le calcul d'une borne de coût minimum au problème original ne peut pas être basée sur la recherche d'une solution de coût optimal du problème relaxé. En effet, l'augmentation du nombre de variables et surtout la diminution du nombre de contraintes ne permettent pas de résoudre efficacement le problème relaxé de manière optimale.

5.2 Ensemble indispensable

Nous présenterons dans cette section différents ensembles que nous qualifierons d'indispensables : au moins un élément de cet ensemble est indispensable à l'obtention d'une solution. Tous ces ensembles peuvent être déterminés à partir de la méthode de relaxation définie dans la section précédente. L'utilisation de la construction du graphe relaxé permet de détecter ces ensembles en temps et en espace polynomial. Comme dans la section précédente, nous nous plaçerons dans le cas où les problèmes admettent au moins un plan-solution et où toutes les préconditions d'actions ne contiennent aucune négation.

Après avoir posé quelques notations, nous définirons des ensembles qui sont indispensables à l'obtention d'une solution. Nous présenterons ensuite des ensembles qui sont indispensables à l'obtention de solution de coût optimal. Ces ensembles sont composés d'une action, d'un fluent, d'un ensemble d'actions ou sont une combinaison d'un ensemble d'actions et d'un ensemble de fluents. Les lemmes introduits dans cette section seront prouvés dans la section suivante. Nous montrerons également commment coder ces notions dans un WCSP issu d'un problème de planification et codé selon la méthode décrite dans le chapitre précédent.

5.2.1 Notations

Afin de simplifier les formules qui suivront, nous utiliserons les notations suivantes. Soit f un fluent, $prec_f$ représente l'ensemble des actions qui ont en précondition f, eff_f l'ensemble des actions ayant f dans ses effets, add_f l'ensemble des actions qui ajoutent f, del_f l'ensemble des actions qui retirent f. C_{min} représentera le coût minimum des actions des problèmes et $mincout(X)$ le coût minimum des actions contenues dans un ensemble d'actions X : $mincout(X) = \min_{a \in X} cout(a)$.

Nous utilisons la méthode de relaxation classique (définition 5.1.2) pour des problèmes dans lesquels nous supprimons des actions ou des fluents :

- $A_{-(X,U)}$ représente l'ensemble des actions de A dans lequel on ignore les actions de X puis les fluents de U dans les effets des actions :

$$A_{-(X,U)} = \{\langle prec(a), eff(a) - U, cout(a)\rangle : a \in (A - X)\}$$

- $\Pi_{-(X,U)}$ représente le problème dans lequel on ignore les actions de X puis les fluents de U dans I et dans les effets des actions :

$$\Pi_{-(X,U)} = \langle A_{-(X,U)}, I - U, B\rangle$$

- $A^+_{-(X,U)}$ représente l'ensemble des actions de A^+ dans lequel on ignore les actions de X puis l'établissement des fluents de U :

$$A^+_{-(X,U)} = \{\langle prec(a), add(a) - U, cout(a)\rangle : a \in (A^+ - X)\}$$

- $\Pi^+_{-(X,U)}$ représente la relaxation du problème $\Pi_{-(X,U)}$ qui consiste à ignorer les actions de X puis les fluents de U dans I et dans les ajouts des actions, et tous les retraits :

$$\Pi^+_{-(X,U)} = \langle A^+_{-(X,U)}, I - U, B\rangle$$

– $G^+_{r,-(X,U)}$ est le graphe de planification réduit et stabilisé, correspondant au problème relaxé $\Pi^+_{-(X,U)}$ et $A^+_{r,-(X,U)}$ représente l'ensemble des actions contenues ce graphe.

5.2.2 Ensemble indispensable

Nous commencerons dans cette sous-section par définir les notions de fluent indispensable et d'action indispensable. Nous montrerons ensuite comment construire plusieurs ensembles indispensables d'actions. Nous généraliserons ces notions en introduisant des paires indispensables qui combinent un ensemble d'actions et un ensemble de fluents. Nous terminerons en construisant un ensemble d'ensembles indispensables d'actions.

Fluent indispensable

[Koehler, 1998 ; Koehler et Hoffmann, 2000 ; Porteous *et al.*, 2001 ; Hoffmann *et al.*, 2004] introduisent la notion de landmark qui est un fluent devant être satisfait au moins une fois durant l'exécution de chaque plan-solution :

Définition 5.2.1 (Landmark [Hoffmann *et al.*, 2004]) *Un fluent f est défini comme un landmark pour le problème $\Pi = \langle A, I, B\rangle$ ssi pour tous les plans-solutions $P = \langle a_1, \ldots, a_n\rangle$, f est satisfait au moins une fois durant l'application du plan-solution :*

$$\exists\, i \in \{0, \ldots, n\} : f \in I \uparrow \langle a_1, \ldots, a_i\rangle$$

Lorsque les sous-buts sont totalement indépendants les uns des autres, la résolution du problème peut être simplifiée en résolvant de manière indépendante chacun des sous-buts. Ceci peut théoriquement mener à une réduction exponentielle de la complexité en temps. Cependant, les sous-buts sont rarement indépendants et le calcul en pré-traitement d'un ordre permettant leur résolution est, dans le pire des cas, aussi difficile que la résolution elle-même du problème. Ainsi, soit de sévères restrictions sont posées sur la représentation [Cheng et Irani, 1989 ; Irani et Cheng, 1987], soit le calcul de cet ordre est considéré comme intraitable pour les gros problèmes [Joslin et Roach, 1989]. [Hoffmann *et al.*, 2004] proposent une méthode polynomiale qui permet d'obtenir une bonne estimation d'un ordre d'exécution des landmarks. Elle peut rendre l'algorithme de recherche incomplet mais, en général, le planificateur IPP, modifié pour l'utiliser, montre de bonnes performances. [Richter *et al.*, 2008] proposent une nouvelle approche pour utiliser les landmarks dans une pseudo-heuristique puis en la combinant avec d'autres heuristiques. L'ajout des informations issues des landmarks permet d'améliorer la résolution et l'obtention de solutions de meilleure qualité. Ils montrent également comment les landmarks et leur ordonnancement peuvent être trouvés en utilisant le formalisme de représentation basé sur des variables multi-valuées. Par rapport aux précédentes approches, leur algorithme garantit l'exactitude de l'ordonnancement des landmarks générés. Cependant, les landmarks calculés par [Hoffmann *et al.*, 2004] ne contribuent pas toujours réellement à la résolution du problème. Puisque un landmark peut par exemple être un effet de bord ajouté par une action indispensable. Nous avons donc défini la notion de fluent indispensable afin de déterminer les fluents réellement indispensables à la résolution du problème :

Définition 5.2.2 (Fluent indispensable) *Un fluent f est défini comme indispensable dans un problème Π ssi il n'existe aucun plan-solution pour le problème $\Pi_{-(\emptyset,\{f\})}$.*

Lemme 5.2.3 *Soit f un fluent pour le problème Π. Si le problème $\Pi^{+}_{-(\emptyset,\{f\})}$ n'a pas de solution alors $\Pi_{-(\emptyset,\{f\})}$ n'en a pas non plus et f est un fluent indispensable pour le problème Π.*

Preuve : Immédiate à partir du lemme 5.1.4. ∎

Exemple 5.2.4 (Fluent indispensable) *Dans l'exemple 4.1, page 55, v_B et c_v sont des fluents indispensables.*

Comme un landmark est un fluent qui est satisfait au moins une fois durant l'application de chaque plan-solution, tous les fluents indispensables sont bien des landmarks.

Notons que la définition de l'ensemble d'actions considérée dans le problème relaxé qui est proposée dans [Hoffmann *et al.*, 2004] est différente de notre définition : pour tester si un fluent f est un landmark, ils ignorent toutes les actions qui ajoutent f, alors que nous ignorons uniquement l'ajout de f pour vérifier si f est un fluent indispensable.

Exemple 5.2.5 *Soit les actions $A1 = \langle\emptyset, \{f1, f2\}, 1\rangle$, $A2 = \langle\{f1\}, \{b\}, 1\rangle$ et $A3 = \langle\{f2\}, \{b\}, 1\rangle$, et le problème $\Pi = \langle\{A1, A2, A3\}, I = \emptyset, B = \{b\}\rangle$. Pour déteminer si le fluent $f1$ est un landmark, l'approche de [Hoffmann* et al.*, 2004] ignore l'action $A1$ et on en déduit que $f1$ est un landmark. Dans notre approche, seul l'ajout de $f1$ est ignoré, le fluent $f2$ est donc ajouté par l'action $A1$ au graphe relaxé, $A3$ peut alors être appliquée pour produire le but b et l'on en déduit que $f1$ n'est pas un fluent indispensable.*

Action indispensable

[Hoffmann et Nebel, 2001] proposent des actions utiles (helpful) qui sont utilisées pour choisir un état prometteur comme successeur de l'état courant. Les actions utiles sont les actions du graphe relaxé qui sont applicables dans l'état pour lequel l'heuristique est calculée, augmentées par toutes les actions applicables dans cet état et produisant des fluents qui ont été déterminés comme étant des sous-buts au premier niveau du graphe de planification lors de l'extraction du plan. Ces actions permettent au planificateur FF d'utiliser une variation de l'algorithme de recherche locale hill-climbing en concentrant ses efforts sur des branches du graphe de recherche plus prometteuses que les autres et en oubliant les actions qui ne sont pas utiles. Quand il ne parvient pas à trouver une solution, la recherche est relancée depuis le début par un algorithme de recherche complet de type meilleur d'abord. D'autres travaux ont étendu cette idée au cadre non-optimal en introduisant par exemple la notion d'états anticipés [Vidal, 2004]. Nous définissons ici une notion plus forte en introduisant des actions qui sont indispensables à la résolution du problème.

Définition 5.2.6 (Action indispensable) *Une action a est définie comme indispensable dans un problème $\Pi = \langle A, I, B\rangle$ ssi elle appartient à tous les plans-solutions du problème Π.*

Lemme 5.2.7 *Si le problème $\Pi^{+}_{-(\{a\},\emptyset)} = \langle A^{+}_{-(\{a\},\emptyset)}, I, B\rangle$ n'a pas de solution alors $\Pi_{-(\{a\},\emptyset)} = \langle A_{-(\{a\},\emptyset)}, I, B\rangle$ n'en a pas non plus et a est une action indispensable au problème Π.*

Preuve : Immédiate à partir du lemme 5.1.4. ∎

Il est possible qu'une même action soit utilisée plusieurs fois dans tous les plans-solutions. La définition 5.2.6 peut être utilisée pour détecter ce cas de la manière suivante : soit une action indispensable a_i, nous remplaçons $a_i \in A$ par l'ensemble d'actions $\{a_{i1}, a_{i2}, \ldots, a_{im}\} \cup \{a'_i\}$ où $\forall\, j \in \{1, \ldots, m\}$, a_{ij} est une copie de a_i exceptée qu'elle a en plus une précondition $compteur^i_{j-1}$, un ajout $compteur^i_j$ et un retrait $compteur^i_{j-1}$. L'action a'_i est une copie de a_i qui a comme précondition supplémentaire $compteur^i_m$. Si nous ajoutons dans l'état initial $compteur^i_0$, alors il est facile de voir que a_{ij} n'est applicable que dans un état $E = I \uparrow P_1 \uparrow a_{i1} \uparrow P_2 \uparrow \ldots \uparrow a_{i(j-1)} \uparrow P_j$ où $\forall\, h \in \{1, \ldots, j\}$, P_h est un plan qui ne contient aucune copie de a_i. Si a_{im} est une action indispensable, alors a est une action m fois indispensable au problème Π (noté m-indispensable). Si une action indispensable a apparaît i fois dans n'importe quel plan-solution (par exemple le plan-solution P^*_k) alors elle est au plus i-indispensable.

Exemple 5.2.8 (Action indispensable) *Dans l'exemple 4.1, page 55, $poser_B$ et $prendre_F$ sont des actions* 1*-indispensables.*

Dans la suite de cette section, nous supposerons que les actions m-indispensables ($\forall\, m > 1$) ont été remplacées par les ensembles d'actions $\{a_{i1}, a_{i2}, \ldots, a_{im}\} \cup \{a'_i\}$ définis précédemment. La section suivante montrera une autre approche pour prendre en compte le nombre d'occurences des actions et des fluents dans les solutions.

Ensemble d'actions indispensable

Comme les problèmes de planification ne contiennent pas toujours des actions indispensables, nous avons étendu la définition d'action indispensable à un ensemble d'actions.

Définition 5.2.9 (Ensemble indispensable d'actions) *Un ensemble d'actions X est un ensemble indispensable d'actions dans un problème $\Pi = \langle A, I, B\rangle$ ssi tous les plans-solutions du problème Π contiennent au moins une des actions de X.*

L'ensemble des actions qui établissent un but constitue un ensemble indispensable d'actions. Les fluents indispensables peuvent également être utilisés pour inférer de nouveaux ensembles indispensables :

Lemme 5.2.10 (Ensemble indispensable dépendant des buts) *Si b est un fluent du but B, alors add_b, l'ensemble des actions ajoutant b, est un ensemble indispensable d'actions.*

Lemme 5.2.11 (Ensemble indispensable ajoutant un fluent indispensable) *Si f est un fluent indispensable tel que $f \notin I$, alors add_f, l'ensemble des actions qui ajoutent f, est un ensemble indispensable d'actions.*

Lemme 5.2.12 (Ensemble indispensable dépendant d'un fluent indispensable) *Si f est un fluent indispensable tel que $f \notin B$, alors $prec_f$, l'ensemble des actions qui ont f en précondition, est un ensemble indispensable d'actions.*

Preuve : Soit f un fluent indispensable, il n'existe donc pas de plan-solution au problème $\Pi^+_{-(\emptyset,\{f\})} = \langle A^+_{-(\emptyset,\{f\})}, I - \{f\}, B\rangle$. Supposons, par contradiction, que $prec_f = \cup_{f\in prec(a)} a$ ne soit pas un ensemble indispensable d'actions, il existe donc un plan-solution $P = \langle a_1, \ldots, a_n\rangle$ tel que $\forall i \in \{1, \ldots, n\}, f \notin prec(a_i)$ pour le problème $\Pi^+_{-(prec_f,\emptyset)} = \langle A^+_{-(prec_f,\emptyset)}, I, B\rangle$. Soit le plan $P' = \langle a'_1, \ldots, a'_n\rangle$ tel que $\forall i \in \{1, \ldots, n\}, a'_i = \langle prec(a_i), add(a_i) - \{f\}, cout(a)\rangle$. Si $f \notin B$, P' est applicable dans $I - \{f\}$ et produit le but B. P' est donc un plan-solution pour le problème $\Pi^+_{-(\emptyset,\{f\})}$, ce qui contredit l'hypothèse selon laquelle f est un fluent indispensable.

∎

Exemple 5.2.13 (Ensembles indispensables liés à un fluent indispensable)
Dans l'exemple 4.1, page 55, $\{poser_B\}$ est l'ensemble indispensable d'actions dépendant du fluent du but ; $\{prendre_F, prendre_B, prendre_C, prendre_D, prendre_E\}$ est un ensemble indispensable d'actions ajoutant c_v et $\{poser_F, poser_B, poser_C, poser_D, poser_E\}$ est un ensemble indispensable d'actions dépendant de c_v.

Deux remarques permettent de calculer de nouveaux ensembles indispensables d'actions à partir des actions indispensables déjà déterminées :

- Pour qu'une action indispensable soit applicable, il est nécessaire que ses préconditions non présentes dans l'état initial aient été établies par d'autres actions.
- Une action indispensable a qui n'ajoute pas de fluent du but, ajoute au moins un fluent qui doit être utilisé par une autre action.

Lemme 5.2.14 (Ensemble indispensable dépendant de préconditions) *Si f est une précondition d'action indispensable telle que $f \notin I$, alors add_f, qui est l'ensemble des actions ajoutant f, est un ensemble indispensable d'actions.*

Lemme 5.2.15 (Ensemble indispensable dépendant d'ajouts) *Soit a une action indispensable telle que $B \cap add(a) = \emptyset$. Si $X_a \in A$ est l'ensemble des actions qui ont en précondition un fluent ajouté par a : $X_a = \bigcup_{f\in add(a)} prec_f$, alors X_a est un ensemble indispensable d'actions.*

Exemple 5.2.16 (Ensembles indispensables dépendants d'actions indispensables)
Dans l'exemple 4.1, page 55, $\{trajet_{FB}, trajet_{CB}, trajet_{EB}\}$ est un ensemble indispensable d'actions dépendant d'une précondition de $poser_B$; $\{poser_F, poser_B, poser_C, poser_D, poser_E\}$ est un ensemble indispensable d'actions dépendant de l'ajout de $prendre_F$;

Le lemme 5.1.5 a montré, en utilisant un graphe de planification relaxé, comment obtenir une borne inférieure k au nombre d'actions des solutions d'un problème de planification. Nous pouvons en déduire que les k premiers niveaux du graphe relaxé, stabilisé et réduit sont un ensemble indispensable d'actions :

Lemme 5.2.17 (Ensemble indispensable basé sur les niveaux) *Si tous les plans-solutions d'un problème de planification optimal Π contiennent au moins k actions, alors l'ensemble des actions $A^+[i]$ qui appartiennent à G^+_r est un ensemble indispensable d'actions pour $i \in \{1, \ldots, k\}$ (où $A^+[i]$ est l'ensemble des actions appartenant au niveau i du graphe de planification relaxée (cf. section 5.1, page 79)).*

Preuve : Si $\langle a_1, \ldots, a_m \rangle$ est un plan-solution, alors clairement $a_i \in A^+[i]$ (pour $i = \{1, \ldots, m\}$) et $a_i \in G_r^+$. En supposant que tous les plans-solutions contiennent au moins k niveaux, nous savons que $m \geq k$. Pour tout $i = 0, \ldots, k-1$, tous les plans-solutions contiennent ainsi au moins une action contenue dans $A_r^+[i]$. L'ensemble des actions de $A^+[i]$ qui appartiennent à G_r^+ est donc un ensemble indispensable d'actions. ∎

Pour établir un minorant du coût d'un plan-solution, les ensembles indispensables d'actions sont d'autant plus intéressant que le coût minimum des actions de l'ensemble est élevé.

Exemple 5.2.18 (Ensemble indispensable dépendant du niveau)
Dans l'exemple 4.1, page 55, l'action la moins coûteuse $poser_B$ *est seulement présente à partir du niveau* 3 *dans le graphe de planification relaxé, stabilisé et réduit. Nous pouvons en déduire deux ensembles indispensables d'actions basés sur les niveaux* 1 *et* 2 *du graphe dont le coût minimum sera supérieur au coût minimum des actions. Pour notre exemple, l'un de ces ensembles d'actions issu du premier niveau du graphe relaxé est* $\{trajet_{FB}, prendre_F, trajet_{FC}\}$.

Soit $A_{\geq}^+ = \{a_1, a_2, \ldots, a_n\}$ l'ensemble des actions appartenant au graphe relaxé, stabilisé et réduit d'un problème $\Pi = \langle A, I, B \rangle$, ordonnées par coûts décroissants : $\forall\, j \in \{1, \ldots, n\}, a_j \in A_r^+$ et $\forall\, j \in \{1, \ldots, n-1\}, cout(a_j) \geq cout(a_{j+1})$.

Lemme 5.2.19 (Ensemble d'actions de plus fort coût) *Soit* $A^i = \{a_1, a_2, \ldots, a_i\}$ *un sous-ensemble de* $A_{\geq}^+$ *construit incrémentalement à partir de* $\{a_1\}$ *jusqu'à contenir une action* a_i *telle que :*
- $\Pi_{i-1}^+ = \langle A_{\geq}^+ - \{a_1, \ldots, a_{i-1}\}, I, B \rangle$ *admet une solution,*
- $\Pi_i^+ = \langle A_{\geq}^+ - \{a_1, \ldots, a_{i-1}, a_i\}, I, B \rangle$ *n'admet pas de solution.*

Alors A^i *est un ensemble indispensable d'actions.*

Quel que soit le problème, il existe toujours un ensemble indispensable d'actions de plus fort coût : dans le pire des cas, cet ensemble est $A_{\geq}^+ = \{a_1, a_2, \ldots, a_n\}$ où a_n a un coût minimum.

Exemple 5.2.20 (Ensemble indispensable d'actions de plus fort coût)
Dans l'exemple 4.1, page 55, $\{trajet_{FB}, trajet_{BF}, trajet_{DE}, trajet_{ED}, trajet_{BC}, trajet_{CB}\}$ *est un ensemble indispensable d'actions de plus fort coût.*

Paire Indispensable

Nous allons combiner les notions d'actions et de fluents indispensables en présentant ici la notion de paire indispensable. Une paire indispensable $\langle X, U \rangle$ est telle que tous les plans-solutions (optimaux et non optimaux) contiennent au moins une action de X ou satisfont, au moins à un instant, un fluent de U lors de l'application de la solution.

Définition 5.2.21 (Paire indispensable) *Une paire* $\langle X, U \rangle$*, où* X *est un ensemble d'actions et* U *est un ensemble de fluents, est une paire indispensable dans un problème* Π *si le problème* $\Pi_{-(X,U)}$ *n'a pas de solution.*

Nous pouvons détecter des paires indispensables $\langle X, U\rangle$ de la même manière que nous l'avons fait pour les actions et les fluents indispensables, en utilisant une relaxation du problème $\Pi_{-(X,U)}$. Bien qu'il ne soit pas envisageable de tester toutes les paires (X, U), nous pouvons tester en temps polynomial le problème $\Pi^{+}_{-(\{a\},\emptyset)}$ pour toutes les actions a et le problème $\Pi^{+}_{-(\emptyset,\{f\})}$ pour tous les fluents f, afin d'obtenir l'ensemble des paires composées d'une action ou d'un fluent indispensable.

Nous proposons dans le lemme suivant une autre méthode, non basée sur le problème relaxé, qui permet de construire des paires indispensables à partir de paires préalablement obtenues. La preuve de ce lemme sera donnée dans la section 5.3, page 98.

Lemme 5.2.22 *Si $\langle X, U\rangle$ est une paire indispensable, alors*

1. *si $a \in X$ et $f \in prec(a)$, alors $\langle X - \{a\}, U \cup \{f\}\rangle$ est une paire indispensable*
2. *si $f \in U$ et $f \notin I$, alors $\langle X \cup add_f, U - \{f\}\rangle$ est une paire indispensable.*

Ensemble d'ensembles indispensables

Comme nous le montrerons dans le chapitre suivant, les ensembles indispensables introduits précédemment permettent de trouver une borne inférieure au coût d'une solution optimale. Ils peuvent également permettre de détecter des actions devenues trop onéreuses pour être utilisées dans une solution de coût optimal (cf. sous-section suivante). Dans les deux cas, nous avons besoin de constituer un ensemble $\mathcal{X}$ d'ensembles indispensables d'actions qui n'ont aucune action en commun.

Nous pouvons déduire des lemmes 5.2.7, 5.2.10, 5.2.11, 5.2.12, 5.2.14, 5.2.15, 5.2.17 et 5.2.19 que le nombre d'ensembles indispensables d'actions que nous pouvons générer à partir de ces lemmes est d'ordre $\mathcal{O}(n_a + n_f + n_l)$ où n_a est le nombre d'actions, n_f est le nombre de fluent, n_l est le nombre de niveaux du graphe de planification relaxé et stabilisé. Le lemme 5.2.22 peut générer un nombre exponentiel d'ensembles indispensables d'actions. Considérons par exemple, un simple problème de planification qui consiste à transporter une caisse entre deux villes C et D. Il y a m routes, différentes et non reliées, permettant de relier C et D, chacune étant composée de n étapes distinctes. Il y a alors exactement m plans-solutions différents, chacun correspondant à un plan séquentiel de n étapes. Le nombre minimum d'ensembles indispensables d'actions et alors de n^m qui correspond au nombre de possibilités de choisir pour chaque route une étape parmi les n différentes.

D'autre part, le problème d'un choix optimal d'ensembles indispensables à utiliser est NP-difficile :

Problème : COEI (Choix Optimal d'Ensembles Indispensables)

Instance : Un problème de planification optimal $\Pi = \langle A, I, B\rangle$ avec C_{min} le coût minimum des actions de A, $mincout(X)$ le minimum des coûts des actions de X, un ensemble J d'ensembles indispensables ($\forall X \in J$, $X \subseteq A$ et X est un ensemble indispensable d'actions) et une constante L.

Question : Existe-t-il un sous-ensemble $\mathcal{X} \subseteq J$ tel que $\forall X, Y \in \mathcal{X}$, $X \cap Y = \emptyset$ et

$$\sum_{X \in \mathcal{X}} (mincout(X) - C_{min}) \geq L$$

Théorème 5.2.23 *COEI est NP-difficile.*

Preuve : Soit $\phi : \{0,1\}^2 \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ la fonction

$$\phi(x,y) \;=\; \begin{cases} \infty & \text{si } x = y = 1 \\ 0 & \text{sinon} \end{cases}$$

et $\psi : \{0,1\} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ la fonction $\psi(0) = 1$, $\psi(1) = 0$.

WCSP(Γ) est l'ensemble des instances du problème de satisfaction de contraintes pondérées dans lesquelles toutes les fonctions de coût appartiennent à Γ. Dans le cas de domaines booléens, il n'existe que huit classes de contraintes valuées traitables [Cohen *et al.*, 2006]. L'ensemble $\{\phi, \psi\}$ est un sous-ensemble d'aucune de ces huit classes traitables, ce qui implique que WCSP($\{\phi, \psi\}$) est NP-difficile.

Soit P une instance de WCSP($\{\phi, \psi\}$) sur les variables $V = \{v_1, \ldots, v_n\}$, représentée par un graphe $G_P = \langle V, E_P \rangle$, un sous-ensemble $V_P \subseteq V$ et une constante K. La question est de déterminer s'il existe une affectation $s : V \rightarrow \{0,1\}$ telle que

$$\sum_{\{v_i,v_j\} \in E_P} \phi(s(v_i), s(v_j)) \;+\; \sum_{v_i \in V_P} \psi(s(v_i)) \;\leq\; K$$

Pour chaque variable $v_i \notin V_P$, on peut affecter la valeur 0 à v_i sans engendrer le moindre coût. Ainsi, on peut éliminer $v_i \notin V_P$ pendant une phase de prétraitement. Par la suite et sans perte de généralité, on suppose que $V_P = V = \{v_1, \ldots, v_n\}$.

Pour démontrer le théorème, il suffit de donner une réduction polynomiale de WCSP($\{\phi, \psi\}$) vers COEI. Pour chaque variable v_i, on crée un fluent B_i. Pour chaque arête $\{v_i, v_j\} \in E_P$, on crée une action $a_{\{i,j\}}$ de coût 2 telle que $add(a_{\{i,j\}}) = \{B_i, B_j\}$. On crée aussi une action a_0 de coût 1. Soit $A = \{a_0\} \cup \{a_{\{i,j\}} : \{v_i, v_j\} \in E_P\}$. Pour chaque variable $v_i \in V_P$, soit $X_i = \{a_{\{i,j\}} : \{v_i, v_j\} \in E_P\}$ l'ensemble des actions qui réalisent le fluent B_i. Soit $J = \{X_i : 1 \leq i \leq n\}$ et

$$B = \bigwedge_{1 \leq i \leq n} B_i$$

Soit Π le problème de planification optimal $\langle A, \emptyset, B \rangle$. Par le Lemme 5.2.10, les ensembles X_i $(1 \leq i \leq n)$ sont des ensembles indispensables de Π.

Il existe une bijection entre les affectations $s : V \rightarrow \{0,1\}$ telles que

$$\forall \{v_i, v_j\} \in E, \phi(s(v_i), s(v_j)) = 0$$

et les choix $\mathcal{X} = \{X_i : s(v_i) = 1,\ 1 \leq i \leq n\}$ d'ensembles indispensables tels que $\forall X_i, X_j \in \mathcal{X}$, $X_i \cap X_j = \emptyset$, car X_i, X_j s'intersectent ssi $\{v_i, v_j\} \in E_P$. Soit $L = n - K$. Comme $C_{min} = 1$ et $\forall X \in \mathcal{X}$, $mincout(X) = 2$,

$$\sum_{X \in \mathcal{X}} (mincout(X) - C_{min}) = |\mathcal{X}|$$

Par définition de la fonction ψ,

$$\sum_{v_i \in V_P} \psi(s(v_i)) \;=\; n \;-\; |\mathcal{X}|$$

Donc $|\mathcal{X}| \geq L$ ssi

$$\sum_{\{v_i,v_j\}\in E_P} \phi(s(v_i), s(v_j)) + \sum_{v_i \in V_P} \psi(s(v_i)) = \sum_{v_i \in V_P} \psi(s(v_i)) \leq n - L = K$$

Donc, les solutions $\mathcal{X}$ à l'instance de COEI définie par Π correspondent exactement aux solutions s au problème de satisfaction de contraintes pondérées P. Cette réduction est clairement polynomiale. ∎

Pour construire un ensemble d'ensembles indispensables d'actions, tout en évitant une complexité exponentielle, nous utiliserons donc un algorithme gourmand.

5.2.3 Ensemble optimalement indispensable

Nous pouvons également détecter, au cours de la recherche, des actions devenues trop onéreuses (dans un plan de coût optimal) car tous les plans-solutions contenant ces actions ont un coût supérieur au coût du plan optimal trouvé jusqu'à présent. L'élimination de ces actions permet de réduire l'espace de recherche et de détecter des paires indispensables à l'obtention d'une solution de coût optimal.

Action trop onéreuse

De nombreux travaux permettent d'éliminer des actions qui ne permettent pas d'atteindre les buts ou qui sont devenues inutiles au cours de la recherche. [Weld, 1999] propose, par exemple, pour instancier efficacement des domaines non typés, d'utiliser les prédicats statiques présents dans l'état initial d'un problème et absents des effets des actions. Les fluents formés avec ces prédicats seront présents dans tous les états et contraindront l'instanciation des opérateurs ; on peut donc ne calculer que les instanciations d'opérateurs compatibles avec ces fluents statiques. Ils ne jouent pas non plus de rôle dans la construction du graphe de planification et peuvent donc être supprimés de l'état initial et des préconditions des actions qu'ils ont permis d'instancier. [Koehler et Hoffmann, 1999] généralisent cette méthode au langage ADL [Pednault, 1989] pour l'inertie de fluents apparaissant après l'instanciation : quand on instancie les actions partiellement instanciées, l'inertie des prédicats permet de trouver d'autres actions non pertinentes ou des effets conditionnels non pertinents qui peuvent être supprimées suivant un principe identique (après étude de l'inertie des fluents constitutifs). [Do *et al.*, 2000] proposent une méthode qui permet de déterminer, à partir du graphe de planification, des actions et des fluents non pertinents pour obtenir le but du problème. Ils calculent des "bmutex" (Backward mutex) qui sont, pour un niveau donné, des exclusions mutuelles binaires entre des fluents et des actions : x, y sont bmutex lorqu'ils ne sont satisfaits simultanément dans aucun état visité par un plan de longueur minimale. Ces bmutex sont propagés en arrière dans le graphe à partir des buts : en employant une des deux actions, on exclut l'autre. Testée avec une approche SAT, cette méthode améliore de façon modérée la résolution des problèmes de planification. [Wehrle *et al.*, 2008] introduisent une méthode générale pour évaluer l'utilité des actions. Cette méthode identifie des actions inutiles (useless) pour l'obtention d'un plan-solution séquentiel de longueur minimale. A partir

d'une heuristique estimant la distance d'un état E à l'état satisfaisant les buts le plus proche, une action applicable a est "useless" dans cet état E si la valeur heuristique de l'état obtenu après l'application de a est supérieure à celle de E. Ils montrent que leur méthode peut être utilisée avec plusieurs heuristiques classiques et combinée avec l'utilisation des actions "helpful". Implémentée dans le planificateur FAST-DOWNARD, qui a remporté la compétition IPC'2004 dans la catégorie non-optimale, elle permet de résoudre davantage de problèmes.

Souhaitant obtenir un plan-solution de coût optimal et de longueur minimale en nombre de niveaux, nous présentons maintenant une méthode qui permet d'éliminer des actions devenues trop onéreuses au cours de la recherche d'une solution de coût optimal et de longueur minimal en niveaux. Nous considérerons également que le coût minimum des actions C_{min} est strictement positif.

Définition 5.2.24 (Action trop onéreuse (1)) *Une action a est trop onéreuse au niveau k et pour tous les niveaux $k' \geq k$ si*

$$(k-1) * C_{min} + cout(a) \geq C^*_{k-1}. \tag{5.1}$$

Cette relation peut être améliorée en tenant compte du coût minimum des ensembles d'actions indispensables. Pour cela, nous utilisons un ensemble $\mathcal{X}$ d'ensembles indispensables d'actions tel que $\forall X, Y \in \mathcal{X}, X \cap Y = \emptyset$.

Définition 5.2.25 (Action trop onéreuse (2)) *Une action a est trop onéreuse à partir d'un niveau k si :*

$$\sum_{X \in \mathcal{X}, a \notin X} mincout(X) + (k - |\mathcal{X}|) * C_{min} + cout(a) \geq C^*_{k-1} \tag{5.2}$$

Exemple 5.2.26 (Action trop onéreuse) *Dans l'exemple 4.1, page 55, le plan-solution P^*_3 a un coût de 108, l'action $trajet_{FB}$ a un coût de 100, elle est donc une action trop onéreuse au niveau 4 et pour tous les niveaux $k' \geq 4$.*

Ensemble optimalement indispensable

Nous avons défini, dans la sous-section précédente, des ensembles indispensables pour des plans-solutions de qualité quelconque. Nous proposons maintenant de définir des ensembles indispensables pour des plans-solutions de coût optimal :

Définition 5.2.27 (Action optimalement indispensable) *Une action a est définie comme optimalement indispensable dans un problème Π ssi elle appartient à tous les plans-solutions de coût optimal du problème Π.*

Cette définition ne nous permet pas de déterminer toutes les actions optimalement indispensables sans connaître l'ensemble des plans-solutions optimaux $\mathcal{P}^*$. Nous montrons ici comment déterminer les actions qui appartiennent à tous les plans-solutions de $\mathcal{P}^*_{k'}$ ($\forall\ k' \geq k$) en utilisant les actions trop onéreuses à partir du niveau k. Nous noterons O_k un ensemble composé de toutes les actions trop onéreuses à partir du niveau k.

Définition 5.2.28 (Action optimalement indispensable à partir du niveau k) *Une action a est définie comme optimalement indispensable à partir du niveau k dans un problème Π ssi elle appartient à tous les plans-solutions $P^*_{k'}$ ($\forall\ k' \geq k$) du problème Π.*

Nous étendrons la définition d'une action optimalement indispensable à une paire optimalement indispensable. La preuve du lemme 5.2.29 sera alors donnée en considérant le cas plus général d'une paire optimalement indispensable à partir du niveau k.

Lemme 5.2.29 *Si une action a est indispensable dans le problème $\Pi_{-(O_k,\emptyset)}$ alors a est une action optimalement indispensable à partir du niveau k pour le problème Π.*

Exemple 5.2.30 (Action optimalement indispensable à partir du niveau k) *Dans l'exemple 4.1, page 55, le plan-solution P^*_3 de coût 108 et l'équation (5.1) permettent de montrer que l'action $trajet_{FB}$ de coût 100 est trop onéreuse à partir du niveau 4. $trajet_{FC}$ est alors une action indispensable pour le problème $\Pi_{-(\{trajet_{FB}\},\emptyset)}$ et elle est donc également une action optimalement indispensable à partir du niveau 4. Le plan P^*_4 et l'équation (5.2) nous permettent de déduire que toutes les actions non contenues dans P^*_4 sont trop onéreuses à partir du niveau 5.*

Lemme 5.2.31 *Une action indispensable est une action optimalement indispensable.*

Preuve : Si une action est indispensable pour un problème Π, elle appartient à tous les plans-solutions de Π et en particulier aux plans-solutions de coût optimal.

■

Définition 5.2.32 (Fluent optimalement indispensable) *Un fluent f est défini comme optimalement indispensable dans un problème Π ssi il n'existe aucun plan-solution de coût optimal pour le problème $\Pi_{-(\emptyset,\{f\})}$.*

Définition 5.2.33 (Fluent optimalement indispensable à partir du niveau k) *Un fluent f est défini comme optimalement indispensable à partir du niveau k dans un problème Π ssi il n'existe aucun plan-solution $P^*_{k'}$ ($\forall\ k' \geq k$) pour le problème $\Pi_{-(\emptyset,\{f\})}$.*

Lemme 5.2.34 *Si f est un fluent indispensable pour le problème $\Pi_{-(O_k,\emptyset)}$ alors f est un fluent optimalement indispensable à partir du niveau k pour le problème Π.*

Nous étendons la définition d'un fluent optimalement indispensable à une paire optimalement indispensable. La preuve du lemme 5.2.34 est donc donnée en considérant le cas plus général d'une paire optimalement indispensable.

Définition 5.2.35 (Paire optimalement indispensable) *Une paire $\langle X, U\rangle$ est définie comme optimalement indispensable dans un problème $\Pi = \langle A, I, B\rangle$ ssi le problème $\Pi_{-(X,U)}$ n'admet aucun plan-solution de coût optimal.*

Définition 5.2.36 (Paire optimalement indispensable à partir du niveau k) *Une paire $\langle X, U\rangle$ est définie comme optimalement indispensable à partir du niveau k dans un problème $\Pi = \langle A, I, B\rangle$ ssi le problème $\Pi_{-(X,U)}$ n'admet aucun plan-solution $P^*_{k'}$ ($\forall\ k' \geq k$).*

Lemme 5.2.37 *Si $\langle X, U\rangle$ est une paire indispensable pour le problème $\Pi_{-(O_k,\emptyset)}$ alors $\langle X, U\rangle$ est une paire optimalement indispensable à partir d'un niveau k pour le problème Π.*

Preuve : Supposons, par contradiction, une paire $\langle X, U\rangle$ qui est indispensable au problème $\Pi_{-(O_k,\emptyset)}$ et qui n'est pas optimalement indispensable à partir d'un niveau k pour le problème Π. Il existe alors un plan-solution P^* de coût optimal et de longueur minimale en niveaux pour le problème $\Pi_{-(X,U)}$. Par définition des actions trop onéreuses, P^* ne contient aucune action de O_k. Ainsi, P^* est aussi un plan-solution au problème $\Pi_{-(O_k\cup X,U)}$. Or, comme $\langle X, U\rangle$ est une paire indispensable, $\Pi_{-(O_k\cup X,U)}$ n'admet aucun plan-solution et contredit l'hypothèse.

■

De la même manière que nous avons défini des paires indispensables à partir d'autres paires indispensables (lemme 5.2.22), nous pouvons construire d'autres paires optimalement indispensables à partir de paires optimalement indispensables obtenues précédemment. Le lemme 5.2.38 reprend le lemme 5.2.22 dans le cadre de paires optimalement indispensables et le lemme 5.2.39 est applicable lorsque l'on cherche des plans qui ne comportent pas d'actions inutiles. La preuve de ces deux lemmes sera donnée dans la section 5.3, page 98.

Lemme 5.2.38 *Si $\langle X, U\rangle$ est une paire optimalement indispensable, alors*

1. *si $a \in X$ et $f \in prec(a)$, alors $\langle X - \{a\}, U \cup \{f\}\rangle$ est une paire optimalement indispensable*
2. *si $f \in U$ et $f \notin I$, alors $\langle X \cup eff_f, U - \{f\}\rangle$ est une paire optimalement indispensable.*

Lemme 5.2.39 *Si $\langle X, U\rangle$ est une paire optimalement indispensable, alors*

1. *si $a \in X$ et $cout(a) \geq 0$, alors $\langle X - \{a\}, U \cup eff(a)\rangle$ est une paire optimalement indispensable,*
2. *si $f \in U$ et $f \notin B$, alors $\langle X \cup prec_f, U - \{f\}\rangle$ est une paire optimalement indispensable.*

5.2.4 Ensemble indispensable dans les WCSP issus de la planification

Le nombre de contraintes a une forte influence sur le temps de résolution des WCSP. Rajouter les notions d'actions et de fluents indispensables dans le wcsp peut améliorer la résolution des wcsp issus des graphes. Nous montrerons dans cette sous-section comment coder une paire indispensable dans un WCSP issu d'un problème de planification et construit selon la méthode du chapitre précédent. Nous donnerons ensuite les résultats expérimentaux obtenus.

Codage d'une paire indispensable dans un WCSP

Le codage d'un graphe de planification en WCSP est décrit dans la section 4.3, page 57. Les deux premières phases du codage font correspondre les fluents et les actions du graphe réduit avec les variables et les valeurs du WCSP. Pour chaque fluent n'appartenant pas à l'état initial, on crée ensuite une variable dont le domaine est l'ensemble des actions qui produisent ce fluent. Pour les fluents n'appartenant pas au but, on ajoute la valeur -1 pour représenter sa non-activation éventuelle. Les fluents indispensables appartenant à l'état initial ne seront donc pas codés dans les WCSP.

Soit une paire indispensable $\langle X, U\rangle$. Pour chaque action $a \in X$, nous notons f_{j_i} ($i \in \{1, \ldots, \Sigma_a\}$) les fluents produits par l'action a. Pour chaque fluent $f \in U$, nous notons f_i un représentant de f et Σ_f le nombre de représentants de f.

Une paire indispensable $\langle X, U\rangle$ est codée par l'ajout d'une variable V de domaine $\{1,\ldots,\Sigma_a,\ldots, \Sigma_a + \Sigma_f\}$ et les contraintes :
- $\forall i \in \{1, \ldots, \Sigma_a\} : V = i \Rightarrow f_{j_i} = a$ *et*
- $\forall i \in \{1 + \Sigma_a, \ldots, \Sigma_a + \Sigma_f\} : V = i \Rightarrow f_i \neq -1$

Exemple 5.2.40 (Codage d'une paire indispensable) *Considérons une paire indispensable $\langle\{a, b\}, \{f, g\}\rangle$ et le WCSP dans lequel l'action a est présente dans les domaines de f_5 et de f_7 ; l'action b est présente dans les domaines de f_7 et de f_9 ; le fluent f est représenté par les variables f_3 et f_5 ; le fluent g est représenté par la variable f_{12}. Pour coder le fait que $\langle\{a, b\}, \{f, g\}\rangle$ est une paire indispensable, nous ajoutons au WCSP la variable V de domaine $\{1, \ldots, 7\}$ et l'ensemble de contraintes : $\{V = 1 \Rightarrow f_5 = a; V = 2 \Rightarrow f_7 = a; V = 3 \Rightarrow f_7 = b; V = 4 \Rightarrow f_9 = b; V = 5 \Rightarrow f_3 \neq -1; V = 6 \Rightarrow f_5 \neq -1; V = 7 \Rightarrow f_{21} \neq -1\}$.*

Expérimentations

Nous avons testé l'efficacité de l'ajout des notions d'actions et de fluents indispensables dans les WCSP lors de la résolution des problèmes par GP-WCSP. Afin d'intégrer de gros WCSP dans ces tests, nous avons utilisé le planificateur GP-WCSP* qui est basé sur GP-WCSP. GP-WCSP* est détaillé dans le chapitre suivant, il permet d'obtenir une solution k-optimale tel que k est le nombre de niveaux minimum pour garantir l'obtention d'une solution de coût optimal. Comme dans les planificateurs GP-WCSP et GP-WCSP*, les WCSP ont été résolus en utilisant la version TOULBAR2.0.6 du solveur TOULBAR2. Nous présenterons dans les figures suivantes les temps de résolution de chacun des WCSP résolus par GP-WCSP* sans tenir compte du coût supplémentaire généré par le calcul des actions et des fluents indispensables ni du temps nécessaire à leur codage en WCSP.

Nous avons comparé la résolution des WCSP avec le codage de toutes les actions indispensables (noté *AvecAction*) par rapport à la résolution des WCSP sans ce codage (noté *SansAction*). Dans cette comparaison, nous avons pris le temps de résolution des WCSP avec le codage des actions indispensables comme indice de difficulté des problèmes. Le graphe 5.1 donne, avec une échelle logarithmique pour l'axe des abscisses, le rapport entre le nombre de contraintes des WCSP *AvecAction* par rapport aux WCSP *SansAction* en fonction du temps de résolution des WCSP *AvecAction*. Nous pouvons observer que ce rapport est compris entre 1 et $1,073$, ce qui signifie que la prise en compte des actions indispensables dans le codage des WCSP augmente peu le nombre de contraintes. L'ordonnée du point moyen, qui correspond à la moyenne des rapports, nous montre que la prise en compte des actions indispensables dans le codage des WCSP augmente de $1,1\%$ le nombre de contraintes sur les problèmes testés.

Le graphe de la figure 5.2 représente, avec une échelle logarithmique, le temps de résolution en utilisant les actions indispensables et sans son utilisation. Nous pouvons déduire de ce graphe que les temps de résolution en intégrant ou non ces actions indispensables évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé

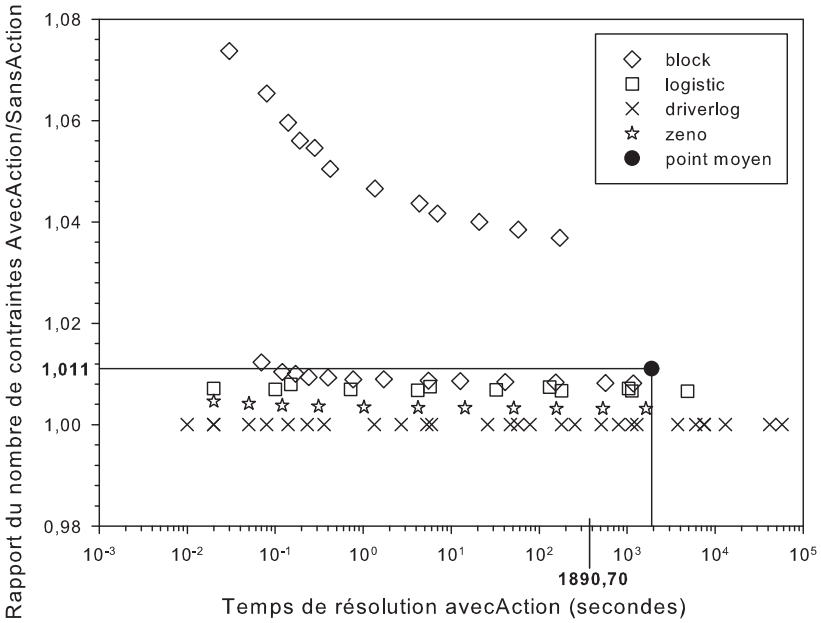


FIG. 5.1 – Rapport du nombre de contraintes en intégrant ou non les actions indispensables dans les WCSP en fonction du temps de résolution en intégrant cette notion.

($R^2 = 0,9965$) et son équation est $AvecAction = 1,0158 \times SansAction^{1,0038}$. Cette courbe nous montre que l'intégration des actions indispensables dans les WCSP ne diminue pas le temps de résolution pour la plupart des problèmes testés. La prise en compte de la notion d'action indispensable dans le codage n'aide donc pas le solveur TOULBAR2.

Nous avons ensuite comparé la résolution des WCSP avec le codage de tous les fluents indispensables (noté *AvecFluent*) par rapport à la résolution des WCSP sans ce codage (noté *SansFluent*). Dans cette comparaison, nous avons pris le temps de résolution des WCSP avec le codage des fluents indispensables comme indice de difficulté des problèmes. Le graphe 5.3 donne, avec une échelle logarithmique pour l'axe des abscisses, le rapport entre le nombre de contraintes des WCSP *AvecFluent* par rapport aux WCSP *SansFluent* en fonction du temps de résolution des WCSP *AvecFluent*. Nous pouvons observer que ce rapport est compris entre 1 et 1,072, ce qui signifie que la prise en compte des fluents indispensables dans le codage des WCSP augmente peu le nombre de contraintes. L'ordonnée du point moyen, qui correspond à la moyenne des rapports, nous montre que la prise en compte des fluents indispensables dans le codage des WCSP augmente de 1,6% le nombre de contraintes sur les problèmes testés.

Le graphe de la figure 5.4 représente, avec une échelle logarithmique, le temps de résolution en utilisant les fluents indispensables et sans son utilisation. Nous pouvons déduire de ce graphe que les temps de résolution en intégrant ou non ces fluents indispensables évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé ($R^2 = 0,9971$) et son équation est $AvecFluent = 1,036 \times SansFluent^{1,0111}$. Cette courbe nous montre que l'intégration des fluents indispensables dans les WCSP diminue faiblement le

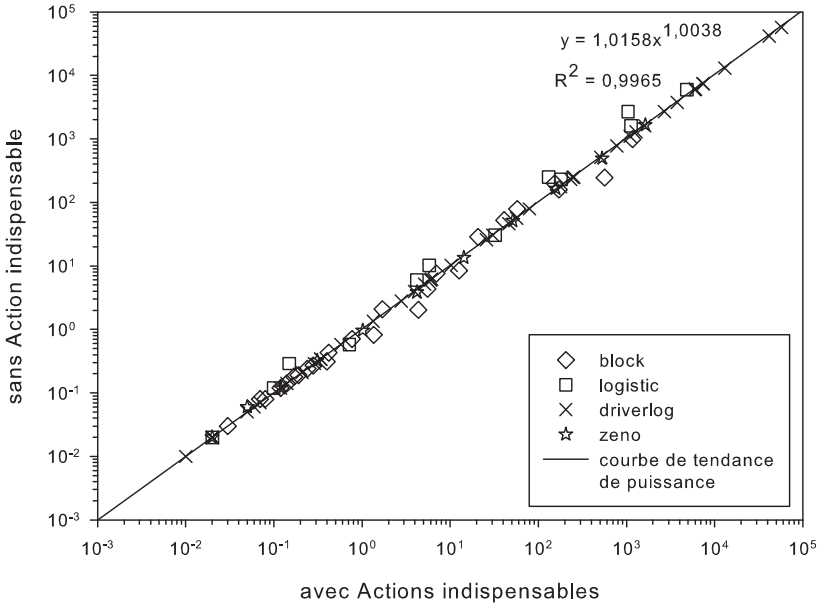


FIG. 5.2 – Comparaison des temps de résolution des WCSP en intégrant ou non les actions indispensables (en secondes).

temps de résolution des WCSP testés. La figure 5.5 montre qu'en moyenne le temps de résolution des WCSP comprenant l'information des fluents indispensables est diminué de 3%. Cependant, même si nous notons une faible amélioration du temps de résolution, elle n'est pas suffisante pour compenser le temps supplémentaire nécessaire à l'ajout de ces informations dans les WCSP.

Nous avons enfin comparé la résolution des WCSP avec le codage de toutes les actions et de tous les fluents indispensables (noté *AvecFluentAvecAction*) par rapport à la résolution des WCSP sans ce codage (noté *SansFluentSansAction*). Dans cette comparaison, nous avons pris le temps de résolution des WCSP avec le codage des fluents et des actions indispensables comme indice de difficulté des problèmes. Le graphe 5.6 donne, avec une échelle logarithmique pour l'axe des abscisses, le rapport entre le nombre de contraintes des WCSP *AvecFluentAvecAction* par rapport aux WCSP *SansFluentSansAction* en fonction du temps de résolution des WCSP *AvecFluentAvecAction*. Nous pouvons observer que ce rapport est compris entre 1 et $1,07$, ce qui signifie que la prise en compte des fluents et des actions indispensables dans le codage des WCSP augmente peu le nombre de contraintes. L'ordonnée du point moyen, qui correspond à la moyenne des rapports, nous montre que la prise en compte des fluents et des actions indispensables dans le codage des WCSP augmente de $1,3\%$ le nombre de contraintes sur les problèmes testés.

Le graphe de la figure 5.7 représente, avec une échelle logarithmique, le temps de résolution en codant les fluents et les actions indispensables et sans ce codage. Nous pouvons déduire de ce graphe que les temps de résolution en intégrant ou non les actions et les fluents indispensables évoluent de façon comparable. La courbe de tendance de puissance obtient le taux de corrélation le plus élevé ($R^2 = 0,9954$) et son équation est $AvecFluentAvecAction =$

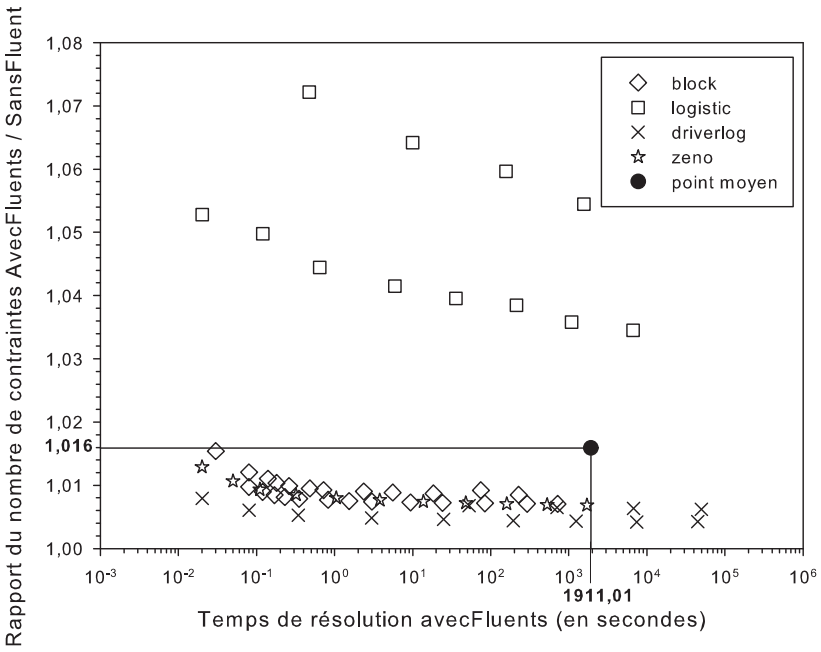


FIG. 5.3 – Rapport du nombre de contraintes en intégrant ou non les fluents indispensables dans les WCSP en fonction du temps de résolution en intégrant cette notion.

$1,017 \times SansFluentSansAction^{1,0071}$. Cette courbe nous montre que l'intégration des fluents et des actions indispensables dans les WCSP diminue faiblement le temps de résolution des WCSP testés. La figure 5.8 montre qu'en moyenne le temps de résolution des WCSP comprenant l'information des fluents et des actions indispensables est diminué de 0,8%. Ce test nous montre que la combinaison des actions et des fluents indispensables n'améliore pas non plus le temps de résolution.

Les tests sur des paires indispensables composées d'actions et de fluents donnent des résultats identiques. La prise en compte de la notion de paires indispensables dans les WCSP n'est donc pas bénéfique au solveur TOULBAR2 et ceci peut s'expliquer de deux manières : soit le solveur déduit déjà ces informations, soit ces nouvelles informations, au lieu d'ajouter de la connaissance, masquent les informations utilisées lors de la résolution. Notons également que la propagation des contraintes dans TOULBAR2 ne se fait qu'au niveau binaire. Les contraintes non binaires d'arité r sont propagées lorsque $r-1$ ou $r-2$ variables sont déjà instanciées. Cette phase de codage ne sera donc pas utilisée dans la suite des expérimentations. Nous montrerons cependant, dans le chapitre suivant, que la notion d'ensembles indispensables d'actions peut être utilisée avec succès pour obtenir une borne inférieure au coût du plan-solution optimal.

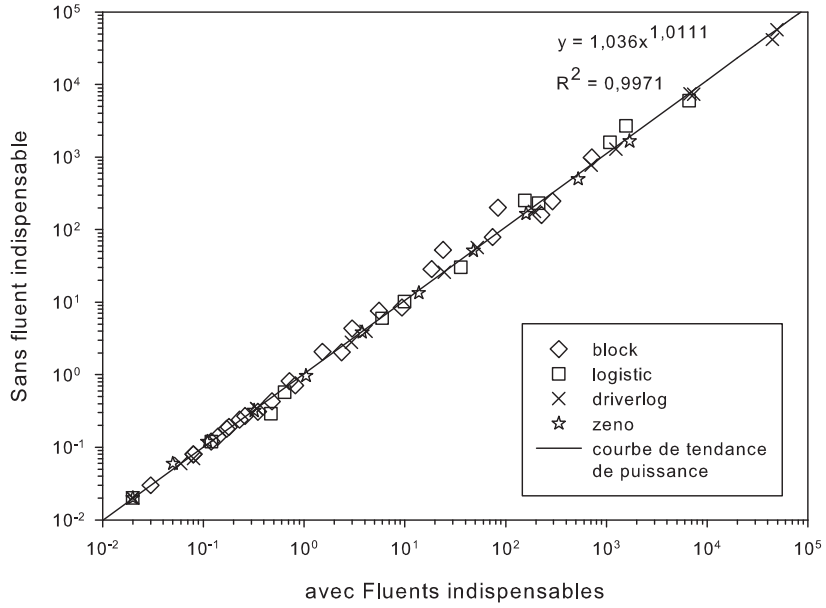


FIG. 5.4 – Comparaison des temps de résolution des WCSP en intégrant ou non les fluents indispensables (en secondes).

5.3 Occurence des actions et des fluents

Dans cette section nous allons montrer comment améliorer la notion d'actions, de fluents ou de paires indispensables en prenant en compte leur nombre d'occurences. Après avoir posé plusieurs définitions, nous montrerons comment obtenir un ensemble d'inégalités linéaires entre le nombre d'occurences des actions et des fluents. Nous montrerons ensuite que ce programme en nombre entier peut être utilisé pour prouver certains des lemmes énoncés précédemment mais aussi pour calculer une borne inférieure au coût d'un plan-solution optimal. En effet, résoudre une relaxation linéaire d'un programme en nombre entier fournit une méthode de complexité temporelle d'ordre polynomiale. Nous montrerons enfin qu'il fournit une méthode alternative pour détecter des problèmes de planification qui n'ont aucune solution et nous en déduirons une nouvelle méthode pour détecter des ensembles indispensables ou optimalement indispensables.

Dans cette section, nous supposerons que Π est un problème de planification qui peut également comporter des fluents négatifs dans les préconditions ou dans les buts. Nous dérivons des relations linéaires entre le nombre d'occurrences des actions et des fluents. En ignorant l'ordonnancement des actions dans un plan, nous évitons les problèmes causés par un horizon borné. Nous obtenons ainsi un programme en nombre entier similaire mais plus simple que dans la formulation de [Benton *et al.*, 2007b] puisque nous ne déterminons pas quelles actions établissent quels fluents. Notre but est de développer un ensemble d'outils qui peuvent être utilisés pour analyser ou transformer des problèmes de planification optimaux de la même manière que les techniques d'analyse ou de transformation de WCSP (cf. section 3.5, page 45, [Cooper et Schiex,

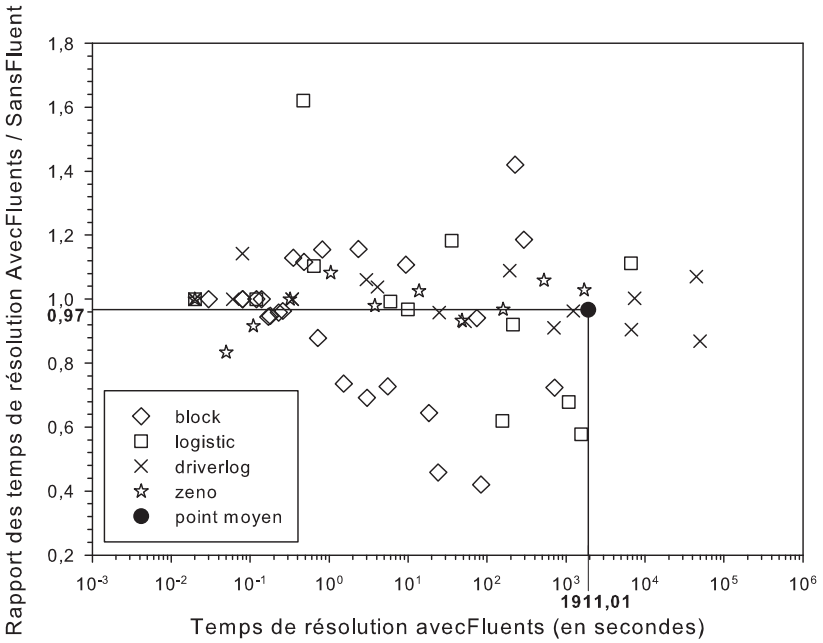


FIG. 5.5 – Rapport du temps de résolution des WCSP en intégrant ou non les fluents indispensables dans les WCSP en fonction du temps de résolution en intégrant cette notion.

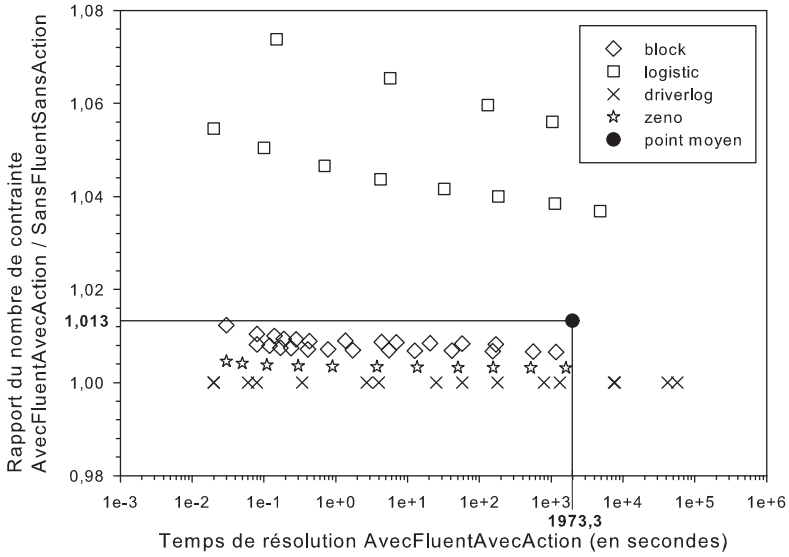


FIG. 5.6 – Rapport du nombre de contraintes en intégrant ou non les actions et les fluents indispensables dans les WCSP en fonction du temps de résolution en intégrant ces notions.

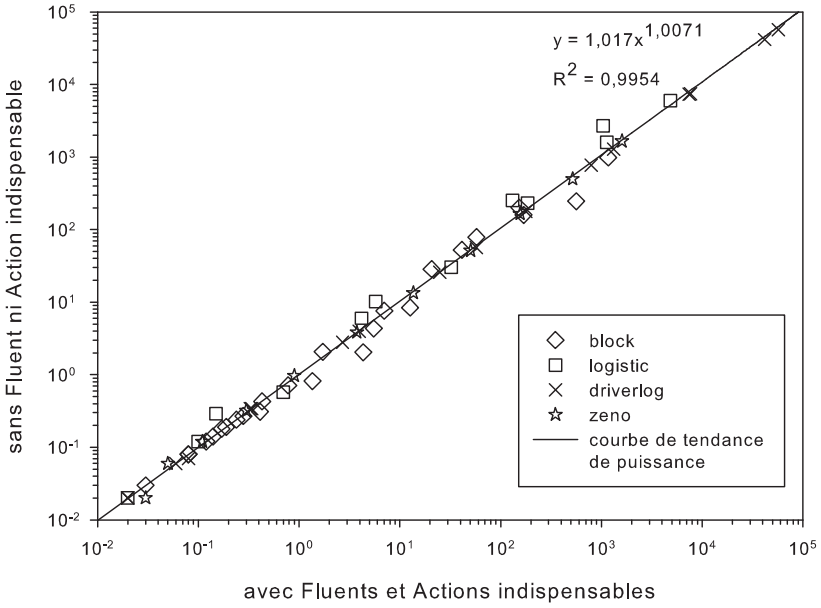


FIG. 5.7 – Comparaison des temps de résolution des WCSP en intégrant ou non les actions et les fluents indispensables (en secondes).

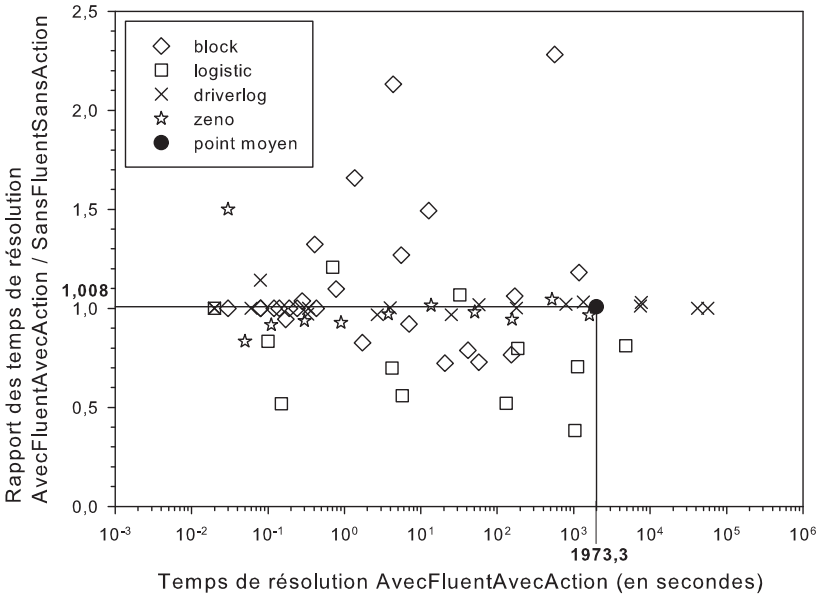


FIG. 5.8 – Rapport du temps de résolution des WCSP en intégrant ou non les actions et les fluents indispensables dans les WCSP en fonction du temps de résolution en intégrant ces notions.

2004 ; Cooper *et al.*, 2007a, 2008b]).

5.3.1 Définitions préliminaires

Nous définissons pour la suite de ce chapitre les notions de fluents et d'états en intégrant les fluents négatifs.

Définition 5.3.1 (Fluent positif / négatif) *Un fluent est positif (négatif) s'il est une variable propositionnelle non-négative (négative).*

Exemple 5.3.2 (Fluent positif / négatif) *Le fluent f est un fluent positif alors que le fluent $\neg f$ est un fluent négatif.*

Définition 5.3.3 (Etat) *Un état est un ensemble de fluents tel que : $\nexists f$ tel que $(f \in E) \wedge (\neg f \in E)$. Un ensemble de fluents positifs (négatifs) noté E^+ (E^-) est associé à un état E. Un ensemble de fluents T est satisfait dans un état E si $T \subseteq E$, i.e. $T^+ \subseteq E^+$ et $T^- \subseteq E^-$.*

Exemple 5.3.4 (Etat) *Considérons un état $E = \{a, b, c, \neg d, \neg e\}$. Associé à cet état E, nous avons l'ensemble des fluents positifs : $E^+ = \{a, b, c\}$ et l'ensemble des fluents négatifs : $E^- = \{\neg d, \neg e\}$.*

Définition 5.3.5 (Problème de planification positif) *Un problème de planification est positif si tous les fluents du but et toutes les préconditions de toutes les actions sont positifs.*

Exemple 5.3.6 (Problème de planification positif) *Considérons les actions suivantes :*

$$A1 = \langle\{a\}, \{b, \neg a\}, 1\rangle \qquad A1' = \langle\{a, \neg b\}, \{b, \neg a\}, 1\rangle$$
$$A2 = \langle\{c\}, \{a, \neg b, \neg c\}, 1\rangle \quad A2' = \langle\{b, c\}, \{a, \neg b, \neg c\}, 1\rangle \quad A2'' = \langle\{b, c, \neg a\}, \{a, \neg b, \neg c\}, 1\rangle$$

Soit l'état initial $I = \{c\}$ et le but $B = \{b\}$. Dans le problème de planification $\Pi = \langle\{A1, A2\}, I, B\rangle$, tous les fluents des préconditions et du but sont des fluents positifs, ainsi Π est un problème de planification positif. Au contraire, le problème de planification $\Pi' = \langle\{A1', A2'\}, I, B\rangle$ n'est pas un problème de planification positif car l'action $A1'$ contient le fluent négatif $\neg b$ dans ses préconditions.

Nous considérerons que dans un état E les fluents de E^+ sont vrais, les fluents de E^- sont faux, et que tous les autres fluents ont une valeur indéfinie. C'est une légère généralisation de l'hypothèse du monde clos du formalisme STRIPS dans laquelle un état correspond à une affectation de valeurs de vérité à tous les fluents. Il est ainsi possible que la valeur d'un fluent soit indéfinie dans l'état initial. Sa valeur sera établie plus tard par une action, mais une fois qu'un fluent est affecté, sa valeur ne peut pas redevenir indéfinie.

Nous noterons $\overline{T} = \{\neg f : f \in T\}$ pour représenter la négation de tous les fluents de T, $B - C = \{b \in B : b \notin C\}$ pour représenter la différence entre deux ensembles. Nous noterons également F_A^+ l'ensemble des fluents dans leur forme positive associés à l'ensemble des actions A, i.e. l'ensemble des littéraux non-négatifs f tel que f ou $\neg f$ appartiennent à une précondition ou un effet d'au moins une action de A. $F_A = F_A^+ \cup \overline{F_A^+}$ représente les versions positives et négatives de tous les fluents associés à l'ensemble d'actions A.

Définition 5.3.7 (Etat complet / partiel) *Un état E est complet pour l'ensemble des fluents F si pour chaque fluent $f \in F$ soit $f \in E$ soit $\neg f \in E$. Dans un contexte d'un problème de planification $\Pi = \langle A, I, B \rangle$, nous dirons qu'un état E est complet si I est complet pour l'ensemble des fluents F_A^+ ; sinon E est un état partiel.*

Exemple 5.3.8 (Etat complet / partiel) *Considérons le problème de planification $\Pi = \langle \{A1, A2\}, I, B \rangle$ de l'exemple 5.3.6. L'ensemble des fluents dans leur forme positive associés à $\{A1, A2\}$ correspond à $F^+_{\{A1,A2\}} = \{a, b, c\}$ et on a $\overline{F^+_{\{A1,A2\}}} = \{\neg a, \neg b, \neg c\}$. Dans ce problème Π, l'état initial $I = \{c\}$ est un état partiel car les fluents a et b ont une valeur non définie. Par contre, dans le problème de planification $\Pi = \langle \{A1, A2\}, \{\neg a, \neg b, c\}, B \rangle$ a un état initial complet.*

Différents problèmes de planification seront considérés en fonction de la présence des fluents dans les effets et les préconditions des actions :

Définition 5.3.9 (Fluent effet-strict) *Un fluent f est un effet-strict si pour toutes les actions a, on a $f \in eff(a) \Rightarrow \neg f \in prec(a)$.*

Exemple 5.3.10 (Fluent effet-strict) *Considérons le problème de planification $\Pi' = \langle \{A1', A2'\}, I, B \rangle$ de l'exemple 5.3.6. Dans ce problème, le fluent b est un fluent effet-strict car il appartient aux effets de l'action $A1'$ et la négation de b (qui correspond au fluent $\neg b$) appartient aux préconditions de cette action. Les fluents $\neg a$, $\neg b$ et $\neg c$ sont les autres fluents effets-stricts du problème.*

Définition 5.3.11 (Action retrait-strict) *Une action a est retrait-strict si pour tous les fluents positifs f, on a $\neg f \in eff(a) \Rightarrow f \in prec(a)$.*

Exemple 5.3.12 (Action retrait-strict) *Considérons les actions de l'exemple 5.3.6. L'action $A1 = \langle \{a\}, \{b, \neg a\}, 1 \rangle$ est une action retrait-strict car le fluent a, qui est le seul fluent positif dont la négation est présente dans les effets de $A1$, appartient au précondition de $A1$. L'action $A1'$ est également une action retrait-strict car elle est une copie de $A1$ excepté qu'elle contient en plus une précondition.*

L'action $A2 = \langle \{c\}, \{a, \neg b, \neg c\}, 1 \rangle$ n'est pas une action retrait-strict car le fluent b, dont la négation est présente dans ses effets, n'appartient pas à ses préconditions. L'action $A2'$ est une copie de $A2$ exceptée qu'elle contient en plus ce fluent b en précondition. Ainsi $A2'$ est une action retrait-strict car tous les fluents positifs, dont la négation est présente dans ses effets, appartiennent à ses préconditions. L'action $A2''$ est également une action retrait-strict car elle est une copie de l'action $A2'$ excepté l'ajout d'une précondition.

Définition 5.3.13 (Problème retrait-strict) *Un problème de planification est retrait-strict si toutes ses actions sont retraits-stricts.*

Exemple 5.3.14 (Problème retrait-strict) *Considérons le problème de planification $\Pi' = \langle \{A1', A2'\}, I, B \rangle$ de l'exemple 5.3.6. Nous avons vu, dans l'exemple 5.3.12, que les actions $A1'$ et $A2'$ sont des actions retraits-stricts. Ainsi, le problème de planification Π' est un problème retrait-strict.*

Définition 5.3.15 (Action effet-strict) *Une action a est effet-strict si pour tous les fluents (positifs ou négatifs) f, on a $f \in eff(a) \Rightarrow \neg f \in prec(a)$.*

Exemple 5.3.16 (Action effet-strict) *Considérons les actions retraits-stricts $A1$, $A1'$, $A2'$ et $A2''$ de l'exemple 5.3.6. L'action $A1 = \langle\{a\}, \{b, \neg a\}, 1\rangle$ n'est pas une action effet-strict car le fluent b est un effet de $A1$ et la négation de b ($\neg b$) n'est pas une précondition de $A1$. Par contre, l'action $A1'$, qui est une copie de $A1$ exceptée qu'elle contient en plus en précondition ce fluent $\neg b$, a toutes les négations de ses effets présentes dans ses préconditions est $A1'$ est donc une action effet-strict.*

De manière similaire, l'action $A2' = \langle\{b, c\}, \{a, \neg b, \neg c\}, 1\rangle$ n'est pas une action effet-strict car elle ne contient pas en précondition la négation de son effet a et l'action $A2''$, qui est une copie de $A2'$ exceptée qu'elle contient en plus en précondition $\neg a$, est une action effet-strict.

Définition 5.3.17 (Problème effet-strict) *Un problème de planification est effet-strict si toutes ses actions sont effets-stricts.*

Exemple 5.3.18 (Problème effet-strict) *Considérons le problème de planification $\Pi'' = \langle\{A1', A2''\}, I, B\rangle$ à partir des actions et des états I et B de l'exemple 5.3.6. Nous avons vu, dans l'exemple 5.3.16, que les actions $A1'$ et $A2''$ sont des actions effets-stricts. Ainsi, le problème de planification Π'' est un problème effet-strict.*

Nous nous intéresserons à des plans-solutions qui ne comportent pas d'actions inutiles :

Définition 5.3.19 (Plan optimal minimal) *Etant donné un problème de planification Π, un plan optimal minimal $P = \langle a_1, \ldots, a_n\rangle$ est un plan de coût optimal tel que $\forall i \in \{1, \ldots\}$, le plan $\langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n\rangle$ n'est pas un plan de coût optimal.*

Exemple 5.3.20 (Plan optimal minimal) *Pour le problème de planification Π de l'exemple 5.3.6, le plan-solution de coût optimal $P = \langle A2, A1\rangle$ est un plan optimal minimal car aucune action ne peut être supprimée tout en conservant un plan-solution de coût optimal. Soit l'action $A3 = \langle\{a\}, \{c\}, 0\rangle$ et le plan-solution de coût optimal $P' = \langle A2, A3, A1\rangle$. P' n'est pas un plan optimal minimal car en supprimant l'action $A3$ de P' on obtient le plan P qui un plan de coût optimal.*

Définition 5.3.21 (Etablissement d'un fluent) *Un fluent f est établi par une action a de P si f passe de faux ou indéfini à vrai durant l'exécution de a.*

Soit e_f le nombre de fois que le fluent f est établi durant l'application de P. Notons que la définition 5.3.21 s'applique pour des fluents négatifs aussi bien que positifs. $e_{\neg f}$ est le nombre de fois que le fluent f passe de la valeur vrai ou indéfini à la valeur faux. Soit b_f (i_f) le nombre de fois que le fluent f est vrai dans le but (l'état initial). b_f et i_f valent soit 0 soit 1.

Exemple 5.3.22 (Etablissement d'un fluent) *Considérons le problème de planification $\Pi = \langle\{A1, A2\}, I, B\rangle$ de l'exemple 5.3.6 et le plan-solution $P = \langle A2, A1\rangle$. Après l'application de P, le fluent a a été établi une fois par l'action $A2$ et le fluent $\neg a$ a été établi une fois par l'action $A1$. Nous obtenons ainsi $e_a = 1$ et $e_{\neg a} = 1$. Le fluent b est un fluent du but, nous avons ainsi $b_b = 1$. Le fluent c est un fluent de l'état initial, nous avons ainsi $i_c = 1$.*

5.3.2 Inéquations linéaires

Nous montrons dans cette sous-section plusieurs lemmes qui établissent des inéquations linéaires. Nous illustrons ces lemmes à partir de l'exemple de problème de planification suivant :

Exemple 5.3.23 (Problème de planification) *Soit le problème de planification* Π *comportant les quatres actions suivantes :*

$A1 = \langle\{a\}, \{b, \neg a\}, 1\rangle$
$A2 = \langle\{c\}, \{a, \neg c\}, 1\rangle$
$A3 = \langle\{d\}, \{b, \neg d\}, 1\rangle$
$A4 = \langle\{b\}, \{c, d, \neg b\}, 1\rangle$

L'état initial est $I = \{c\}$ *et l'ensemble des buts* $B = \{a, b\}$.

Le lemme 5.3.24 et le corollaire 5.3.26 évaluent la différence du nombre d'établissement entre un fluent et sa négation en tenant compte de la présence de ces fluents dans l'état initial et dans les buts :

Lemme 5.3.24 *Si* f *est un fluent, alors*

$$b_f + i_{\neg f} - 1 \leq e_f - e_{\neg f}$$

Si $\neg f$ *est effet-strict et* $f \notin I$*, alors*

$$b_f \leq e_f - e_{\neg f}$$

Preuve : Pour qu'un fluent soit établi par une action a, il doit être faux (ou indéfini) quand l'action a est appliquée. Ceci implique que durant l'exécution d'un plan, les établissements de f et de $\neg f$ alternent. Par conséquent, pour tous les fluents f, $-1 \leq (e_f - e_{\neg f})$. Si $b_f = 1$ (i.e. f doit être vrai à la fin de l'application du plan) ou $i_{\neg f} = 1$ (i.e. f est faux avant l'application du plan), alors $0 \leq (e_f - e_{\neg f})$. De plus, si nous avons à la fois $b_f = 1$ et $i_{\neg f} = 1$, alors $1 \leq (e_f - e_{\neg f})$. On en déduit immédiatement que $b_f + i_{\neg f} - 1 \leq e_f - e_{\neg f}$.

Lorsque $\neg f$ est effet-strict, pour toutes les actions a, $\neg f \in eff(a) \Rightarrow f \in prec(a)$. Si $f \notin I$, alors nous avons l'inégalité plus forte $b_f \leq e_f - e_{\neg f}$ car, dans ce cas, f doit être établi avant $\neg f$. ∎

Exemple 5.3.25 (Application du lemme 5.3.24) *Considérons le problème de planification* Π *de l'exemple 5.3.23. L'application de la première partie du lemme 5.3.24 au fluent a produit l'inégalité* $b_a + i_{\neg a} - 1 \leq e_a - e_{\neg a}$*. Sachant que* $b_a = 1$ *car a est un but et* $i_{\neg a} = 0$ *car* $\neg a$ *n'est pas dans l'état initial, nous obtenons :* $0 \leq e_a - e_{\neg a}$ *qui signifie que le fluent a est établi au moins autant de fois que sa négation dans les plans-solutions.*

De plus, comme le fluent $\neg a$ *est un fluent effet-strict et que a n'appartient pas à I, nous avons en appliquant la deuxième partie du lemme 5.3.24 :* $b_a \leq e_a - e_{\neg a}$*. Nous pouvons en déduire* $1 \leq e_a - e_{\neg a}$ *qui signifie que le fluent a est établi plus de fois que sa négation dans les plans-solutions.*

Corollaire 5.3.26 *Pour tous les fluents* f*,*

$$b_f + i_{\neg f} - 1 \leq e_f - e_{\neg f} \leq 1 - b_{\neg f} - i_f$$

Preuve : L'application du Lemme 5.3.24 à $\neg f$ produit l'inégalité $b_{\neg f} + i_f - 1 \leq e_{\neg f} - e_f$. En le réarrangeant et en le combinant avec l'application du Lemme 5.3.24 à f, nous obtenons $b_f + i_{\neg f} - 1 \leq e_f - e_{\neg f} \leq 1 - b_{\neg f} - i_f$. ∎

Exemple 5.3.27 (Application du corollaire 5.3.26) *Considérons le problème de planification de l'exemple 5.3.23. L'application du corollaire 5.3.26 au fluent a produit l'inégalité :* $b_a + i_{\neg a} - 1 \leq e_a - e_{\neg a} \leq 1 - b_{\neg a} - i_a$. *Sachant que* $b_a = 1$, $i_{\neg a} = 0$, $b_{\neg a} = 0$ *et* $i_a = 0$, *nous obtenons* $0 \leq e_a - e_{\neg a} \leq 1$ *qui signifie que le fluent a est établi autant de fois ou au maximum une fois de plus que sa négation.*

Soit o_a le nombre d'occurences de l'action a dans le plan P. Le nombre d'établissement d'un fluent peut être borné par le nombre d'occurences des actions qui ont ce fluent dans leur effets :

Lemme 5.3.28 *Si f est un fluent, alors*

$$e_f \leq \sum_{a \in eff_f} o_a$$

et de plus, si f est effet-strict, alors nous avons une égalité.

Preuve : L'inégalité provient du fait que chaque établissement d'un fluent f doit être effectué par une action a pour laquelle $f \in eff(a)$. Si f est effet-strict, alors toutes les actions de eff_f permettent réellement d'établir f, et nous avons alors l'égalité. ∎

Exemple 5.3.29 (Application du lemme 5.3.28) *Considérons le problème de planification de l'exemple 5.3.23. L'application du lemme 5.3.28 au fluent b produit l'inégalité* $e_b \leq \sum_{a \in eff_b} o_a$. *Comme le fluent b est présent dans les effets des actions* $A1$ *et* $A3$, *nous avons* $e_b \leq o_{A1} + o_{A3}$. *Cette inégalité signifie que, dans un plan-solution P, le fluent b n'est pas établi plus de fois que la somme des occurences des actions* $A1$ *et* $A3$ *dans P.*

L'application du lemme 5.3.28 au fluent $\neg a$, sachant que $\neg a$ est effet-strict et appartient uniquement aux effets de l'action $A1$, *produit l'égalité* $e_{\neg a} = \sum_{a \in eff_{\neg a}} o_a = o_{A1}$. *Cette égalité signifie que, dans un plan-solution P, le fluent $\neg a$ est établi autant de fois que l'action* $A1$ *est utilisé dans P.*

Tous les établissements d'un fluent pris en compte dans la définition 5.3.21 ne sont pas utiles à la résolution du problème. En effet, un fluent peut être un effet d'une action et ne pas être utilisé lors de la résolution du problème. Pour obtenir des inégalités plus précises, nous définissons la notion d'établissement utile d'un fluent :

Définition 5.3.30 (Etablissement utile d'un fluent) *Soit* $P = \langle a_1, \ldots, a_n \rangle$ *un plan-solution du problème* $\Pi = \langle A, I, B \rangle$, *tel qu'une action* a_i *établisse le fluent f durant l'application de P. L'établissement de f par* a_i *est dit utile si on ne peut pas l'ignorer et obtenir une solution, i.e.* $\langle a_1, \ldots, a_{i-1}, a_i^{-f}, a_{i+1}, \ldots, a_n \rangle$ *n'est pas un plan-solution pour* Π, *où* a_i^{-f} *est l'action* $\langle prec(a_i), eff(a_i) - \{f\}, cout(a) \rangle$. *La présence d'un fluent f dans l'état initial I est dite utile si on ne peut pas l'ignorer et obtenir un plan-solution, i.e. si P n'est pas un plan-solution pour le problème* $\langle A, I - \{f\}, B \rangle$.

Exemple 5.3.31 (Etablissement utile d'un fluent) *Considérons le problème de planification de l'exemple 5.3.23 et le plan-solution $P = \langle A2, A1, A4, A3, A2\rangle$. Par exemple, durant l'exécution de P, l'établissement du fluent b, par l'action $A1$ puis par l'action $A3$, est utile car sans celui-ci les actions $A4$ et $A2$ ne sont plus applicables et P n'est plus un plan-solution. Par contre, l'établissement de $\neg a$ par l'action $A1$ n'est pas utile car le plan P reste un plan-solution même en ignorant l'ajout de $\neg a$ dans l'action $A1$. La présence du fluent c dans l'état initial est utile car elle permet d'appliquer l'action $A2$ qui appartient au plan P.*

Soit e_f^{utile} le nombre de fois que le fluent f est utilement établi durant l'exécution de P. Soit i_f^{utile} le nombre de fois que le fluent f est utilement présent dans l'état initial I. Le lemme suivant découle de la définition d'un établissement utile et de la présence utile d'un fluent dans I :

Lemme 5.3.32 *Pour tous les fluents f, $e_f^{utile} \leq e_f$ et $i_f^{utile} \leq i_f$.*

Exemple 5.3.33 (Application du lemme 5.3.32) *Considérons le problème de planification de l'exemple 5.3.23 et le plan-solution $P = \langle A2, A1, A4, A3, A2\rangle$. L'application du lemme 5.3.32 au fluent b produit l'inégalité : $e_b^{utile} \leq e_b$. Nous pouvons vérifier cette inégalité en considérant l'exécution du plan-solution P : nous avons vu, dans l'exemple 5.3.31, que durant l'exécution de P l'établissement du fluent b est utile deux fois, noté $e_b^{utile} = 2$, et nous pouvons calculer que $e_b = 2$ durant l'exécution de P.*

Nous pouvons également vérifier la deuxième inégalité avec le fluent c qui produit $i_c^{utile} \leq i_c$. Comme c est présent dans I, nous avons $i_c = 1$ et nous avons vu dans l'exemple 5.3.31 que le fluent c était utilement présent dans I et que l'on note $i_c^{utile} = 1$.

Nous pouvons évaluer le nombre d'établissement utile d'un fluent et la présence utile dans l'état initial d'un fluent en sachant que les actions des plans-solutions ne sont applicables que si toutes leurs préconditions sont satisfaites :

Lemme 5.3.34 *Pour chaque $f \in prec(a)$,*

$$o_a > 0 \Rightarrow e_f^{utile} + i_f^{utile} > 0$$

Preuve : Une action a est applicable seulement si toutes ses préconditions sont satisfaites. De plus, si a est appliquée alors chaque précondition $f \in prec(a)$ est utilement présente dans l'état initial I ou a été utilement établie. ∎

Exemple 5.3.35 (Application du lemme 5.3.34) *Considérons le problème de planification de l'exemple 5.3.23. L'application du lemme 5.3.34 à la précondition b de l'action $A4$ produit l'inégalité $o_{A4} > 0 \Rightarrow e_b^{utile} + i_b^{utile} > 0$. Et nous pouvons vérifier cette inégalité lors de l'exécution du plan-solution $P = \langle A2, A1, A4, A3, A2\rangle$ avec $e_b^{utile} = 2$ et $i_b^{utile} = 0$.*

Le nombre d'établissement utile d'un fluent et la présence utile d'un fluent dans I peuvent être bornés par le fait que le fluent soit un but et par le nombre d'occurences des actions qui requiert ce fluent en précondition :

Lemme 5.3.36 *Pour tous les fluents f,*

$$e_f^{utile} + i_f^{utile} \leq b_f + \sum_{a \in prec_f} o_a$$

Preuve : D'après la définition 5.3.30, chaque fois qu'un fluent f est utilement présent dans l'état initial ou utilement établi, f doit être soit un fluent du but soit une précondition d'une action de P. ∎

Exemple 5.3.37 (Application du lemme 5.3.36) *Considérons le problème de planification de l'exemple 5.3.23. L'application du lemme 5.3.36 au fluent b produit l'inégalité $e_b^{utile} + i_b^{utile} \leq b_b + \sum_{a \in prec_b} o_a$. Comme b n'appartient pas à I : $i_b^{utile} = 0$, comme b appartient à B : $b_b = 1$ et comme seule l'action A4 a b en précondition $\sum_{a \in prec_b} o_a = o_{A4}$, nous obtenons : $e_b^{utile} \leq 1 + o_{A4}$. Et nous pouvons vérifier cette inégalité lors de l'exécution du plan-solution $P = \langle A2, A1, A4, A3, A2 \rangle$ avec $e_b^{utile} = 2$ et $o_{A4} = 1$.*

Nous avons défini la notion de paire indispensable dans la sous-section 5.2.2, page 83, et la notion de paire optimalement indispensable, dans la sous-section 5.2.3, page 90. Ces définitions considèrent que les problèmes admettent au moins un plan-solution et que toutes les préconditions d'actions ne contiennent aucune négation. En considérant les mêmes conditions d'utilisation, nous pouvons traduire ces deux notions en inégalités avec les deux lemmes suivants.

Lemme 5.3.38 *$\langle X, U \rangle$ est une paire indispensable pour Π si et seulement si pour tous les plans-solutions P de Π*

$$\sum_{a \in X} o_a + \sum_{f \in U} (e_f^{utile} + i_f^{utile}) \geq 1$$

Preuve : Soit $\langle X, U \rangle$ une paire indispensable. Alors $\Pi_{-(X,U)} = \langle A_{-(X,U)}, I - U, B \rangle$ n'a pas de plan-solution. P est un plan-solution de Π mais il n'est pas un plan-solution pour $\Pi_{-(X,U)}$. De plus, soit une action $a \in X$ appartient à P, soit un fluent $f \in U$ est un but ou une précondition d'une action de P :

- Si $a \in X$ appartient à P, alors $\sum_{a \in X} o_a \geq 1$.
- Si $f \in U$ est un but ou une précondition d'une action de P, alors f doit être utilement établi dans I ou utilement présent par une action de P et ainsi $e_f^{utile} + i_f^{utile} \geq 1$.

Supposons que $\sum_{a \in X} o_a + \sum_{f \in U} (e_f^{utile} + i_f^{utile}) \geq 1$ pour tous les plans-solutions de Π. Supposons, par contradiction, que $\langle X, U \rangle$ n'est pas une paire indispensable. $\Pi_{-(X,U)}$ a donc un plan-solution P. Par définition, aucune action a de P n'appartient à X. De plus, durant l'application de P depuis l'état initial $I - U$, tous les fluents $f \in U$ restent indéfinis. L'application de P à partir de l'état initial I produit le but B sans utiliser les actions de X ni les fluents de U. P est donc un plan-solution pour Π et nous avons

$$\sum_{a \in X} o_a + \sum_{f \in U} (e_f^{utile} + i_f^{utile}) = 0$$

Cette contradiction termine la preuve. ∎

Exemple 5.3.39 (Application du lemme 5.3.38) *Considérons le problème de planification* Π *de l'exemple 5.3.23 et le plan-solution* $P = \langle A2, A1, A4, A3, A2\rangle$. *Nous pouvons déterminer les paires indispensables* $\langle\{A1, A3\}, \emptyset\rangle$ *et* $\langle\emptyset, \{b\}\rangle$. *L'application du lemme 5.3.38 à la paire* $\langle\{A1, A3\}, \emptyset\rangle$ *produit l'inégalité* $o_{A1} + o_{A3} \geq 1$ *que l'on peut vérifier avec le plan-solution* P. *L'application du lemme 5.3.38 à la paire* $\langle\emptyset, \{b\}\rangle$ *produit l'inégalité* $e_b^{utile} + i_b^{utile} \geq 1$ *que l'on peut vérifier lors de l'exécution du plan-solution* P.

Lemme 5.3.40 $\langle X, U\rangle$ *est une paire optimalement indispensable si et seulement si dans tous les plans de coût optimal*

$$\sum_{a\in X} o_a + \sum_{f\in U}(e_f^{utile} + i_f^{utile}) \geq 1$$

Preuve : Supposons que $\langle X, U\rangle$ est une paire optimalement indispensable et supposons, par contradiction, qu'il existe un plan de coût optimal P pour le problème Π tel que

$$\sum_{a\in X} o_a + \sum_{f\in U}(e_f^{utile} + i_f^{utile}) = 0 \tag{5.3}$$

Alors P est également un plan-solution pour $\Pi_{-(X,U)}$. Soit un plan optimal minimal $P' \subseteq P$. L'équation (5.3) est nécessairement vérifiée pour $P' \subseteq P$. Par conséquent P' est un plan optimal minimal pour Π et un plan-solution pour $\Pi_{-(X,U)}$, ce qui contredit la supposition que $\langle X, U\rangle$ est une paire indispensable dans des plans optimaux. Supposons que

$$\sum_{a\in X} o_a + \sum_{f\in U}(e_f^{utile} + i_f^{utile}) \geq 1 \tag{5.4}$$

dans tous les plans optimaux. Supposons par contradiction que $\Pi_{-(X,U)}$ a un plan-solution P qui est un plan optimal minimal pour Π. Alors l'inégalité (5.4) est vérifiée pour P. Mais ceci est en contradiction avec le fait que P ne doit contenir aucune action de X et que tous les fluents $f \in U$ restent indéfinis durant l'application de P à partir de l'état initial $I - U$.

■

Exemple 5.3.41 (Application du lemme 5.3.40) *Considérons le problème de planification* Π *de l'exemple 5.3.23 et le plan-solution de coût optimal* $P = \langle A2, A1, A4, A3, A2\rangle$. *Comme* Π *est un problème simple, les paires optimalement indispensables sont également des paires indispensables. Ainsi l'application du lemme 5.3.40 à la paire optimalement indispensable* $\langle\{A1, A3\}, \emptyset\rangle$ *produit la même inégalité que le lemme 5.3.38 :* $o_{A1} + o_{A3} \geq 1$. *De manière générale, les paires optimalement indispensables ne sont pas toujours indispensables et le lemme 5.3.40 permet d'obtenir des inégalités supplémentaires par rapport au lemme 5.3.38.*

Dans le lemme 5.3.42, nous supposons un plan optimal minimal P dans un problème de planification Π. Puisque P est minimal, aucune action a de P tel que $cout(a) \geq 0$ ne peut être supprimée de P en conservant un plan-solution. Ceci implique que chaque occurence de l'action a dans P établi utilement un fluent (sinon a pourrait être supprimée).

Lemme 5.3.42 *Dans un plan optimal minimal pour un problème de planification, pour toutes les actions a telle que cout(a) ≥ 0, on a*

$$o_a \leq \sum_{f \in eff(a)} e_f^{utile}$$

Exemple 5.3.43 (Application du lemme 5.3.42) *Considérons le problème de planification Π de l'exemple 5.3.23 et le plan-solution optimal optimal $P = \langle A2, A1, A4, A3, A2\rangle$. L'application du lemme 5.3.42 à l'action A1 produit l'inégalité : $o_{A1} \leq \sum_{f \in eff(A1)} e_f^{utile}$ et donc $o_{A1} \leq e_b^{utile} + e_{\neg a}^{utile}$. Cette inégalité peut être vérifié dans le plan P avec $o_{A1} = 1$, $e_b^{utile} = 2$ et $e_{\neg a}^{utile} = 0$.*

Comme chaque occurence d'une action a dans un plan optimal minimal P établi utilement au moins un fluent, le nombre d'occurences de a dans P peut être borné par le nombre de fois que les effets de a sont utilisés (fluent présent dans les buts ou en précondition des autres actions). Nous posons que b_U est égale à 1 si un fluent de U est un fluent du but et 0 sinon et obtenons le lemme suivant :

Lemme 5.3.44 *Dans un plan optimal minimal pour un problème de planification, pour toutes les actions a telle que cout(a) ≥ 0, on a*

$$o_a \leq b_{eff(a)} + \sum_{a' \text{ tel que } prec(a') \cap eff(a) \neq \emptyset} o_{a'}$$

Exemple 5.3.45 (Application du lemme 5.3.44) *Considérons le problème de planification Π de l'exemple 5.3.23 et le plan-solution optimal optimal $P = \langle A2, A1, A4, A3, A2\rangle$. L'application du lemme 5.3.44 à l'action A1 produit l'inégalité : $o_{A1} \leq b_{eff(A1)} + \sum_{a' \text{ tel que } prec(a') \cap eff(A1) \neq \emptyset} o_{a'}$ Comme l'action A1 produit un but, et que seul l'effet b apparaît dans une autre précondition d'action (A4), nous obtenons : $o_{A1} \leq 1 + o_{A4}$. Cette inégalité peut être vérifiée dans le plan P avec $o_{A1} = 1$ et $o_{A4} = 1$.*

5.3.3 Preuve des lemmes de construction de paires indispensables

Nous donnons, dans cette sous-section, les preuves des lemmes 5.2.22, 5.2.38 et 5.2.39 qui nous ont permis de construire des paires (optimalement) indispensables à partir d'autres paires (optimalement) indispensables. Les preuves s'obtiennent directement à partir des lemmes donnés dans la sous-section précédente et en considèrant que les problèmes admettent au moins un plan-solution et que toutes les préconditions d'actions ne contiennent aucune négation.

Preuve du lemme 5.2.22 : Soit $\langle X, U\rangle$ une paire indispensable. Par le lemme 5.3.38,

$$\sum_{a \in X} o_a + \sum_{f \in U} (e_f^{utile} + i_f^{utile}) \geq 1. \tag{5.5}$$

Soit $a0 \in X$ et $f0 \in prec(a0)$. Par le lemme 5.3.34, $o_{a0} > 0 \Rightarrow e_{f0}^{utile} + i_{f0}^{utile} > 0$. Par conséquent,

$$\sum_{a \in X - \{a0\}} o_a + \sum_{f \in U \cup \{f0\}} (e_f^{utile} + i_f^{utile}) \geq 1$$

et donc $\langle X - \{a0\}, U \cup \{f0\}\rangle$ est une paire indispensable par le lemme 5.3.38. Maintenant, pour une même paire indispensable $\langle X, U\rangle$, considérons $f1 \in U$ où $f1 \notin I$. Par les lemmes 5.3.28 et 5.3.32, nous avons

$$e_{f1}^{utile} \leq e_{f1} \leq \sum_{a \in eff_{f1}} o_a$$

En utilisant le fait que $f1 \notin I$ et par conséquent que $i_{f1}^{utile} = 0$, nous pouvons déduire de l'inégalité (5.6) que

$$\sum_{a \in X \cup\ eff_{f1}} o_a + \sum_{f \in U - \{f1\}} (e_f^{utile} + i_f^{utile}) \geq 1$$

et donc $\langle X \cup\ eff_{f1}, U - \{f1\}\rangle$ est une paire indispensable par le lemme 5.3.38.

∎

Preuve du lemme 5.2.38 : Soit $\langle X, U\rangle$ une paire optimalement indispensable. Par le lemme 5.3.40 dans tous les plans de coût optimal

$$\sum_{a \in X} o_a + \sum_{f \in U} (e_f^{utile} + i_f^{utile}) \geq 1 \tag{5.6}$$

Soit $a0 \in X$ et $f0 \in prec(a0)$. Par le lemme 5.3.34 $o_{a0} > 0 \Rightarrow e_{f0}^{utile} + i_{f0}^{utile} > 0$. Par conséquent, dans tous les plans de coût optimal

$$\sum_{a \in X - \{a0\}} o_a + \sum_{f \in U \cup \{f0\}} (e_f^{utile} + i_f^{utile}) \geq 1$$

et donc $\langle X - \{a0\}, U \cup \{f0\}\rangle$ est une paire optimalement indispensable par le lemme 5.3.40.

Maintenant, pour une même paire optimalement indispensable $\langle X, U\rangle$, considérons $f1 \in U$ où $f1 \notin I$. Par les lemmes 5.3.28 et 5.3.32, nous avons

$$e_{f1}^{utile} \leq e_{f1} \leq \sum_{a \in eff_{f1}} o_a$$

En utilisant le fait que $i_{f1}^{utile} = 0$ (car $f1 \notin I$), nous pouvons déduire de l'inégalité (5.6) que, dans tous les plans-solutions de coût optimal

$$\sum_{a \in X \cup\ eff_{f1}} o_a + \sum_{f \in U - \{f1\}} (e_f^{utile} + i_f^{utile}) \geq 1$$

et donc $\langle X \cup\ eff_{f1}, U - \{f1\}\rangle$ est une paire optimalement indispensable par le lemme 5.3.40.

∎

Preuve du lemme 5.2.39 : Soit $\langle X, U\rangle$ une paire optimalement indispensable. Par le lemme 5.3.40, dans tous les plans de coût optimal,

$$\sum_{a \in X} o_a + \sum_{f \in U} (e_f^{utile} + i_f^{utile}) \geq 1 \tag{5.7}$$

Soit $a0 \in X$ tel que $cout(a0) \geq 0$. Par le lemme 5.3.42, dans un plan optimal minimal

$$o_{a0} \leq \sum_{f \in eff(a0)} e_f^{utile}$$

Par conséquent, dans un plan optimal minimal,

$$\sum_{a \in X - \{a0\}} o_a + \sum_{f \in U \cup eff(a0)} (e_f^{utile} + i_f^{utile}) \geq 1 \tag{5.8}$$

Pour chaque plan optimal P, il existe un plan optimal minimal $P' \subseteq P$ qui vérifie l'inégalité (5.8). Clairement, l'inégalité (5.8) est aussi vérifiée pour P. Donc $\langle X - \{a0\}, U \cup eff(a0)\rangle$ est une paire optimalement indispensable.

Soit $f0 \in U$ tel que $f0 \notin B$. Par le lemme 5.3.36 où $f0 \notin B$,

$$e_{f0}^{utile} + i_{f0}^{utile} \leq \sum_{a \in prec_{f0}} o_a$$

Par conséquent, dans un plan optimal,

$$\sum_{a \in X \cup prec_{f0}} o_a + \sum_{f \in U - \{f0\}} (e_f^{utile} + i_f^{utile}) \geq 1$$

et donc $\langle X \cup prec_{f0}, U - \{f0\}\rangle$ est une paire optimalement indispensable par le lemme 5.3.40.

■

5.3.4 Utilisations des inéquations linéaires

Les inéquations présentées dans la sous-section 5.3.2, page 104, nous ont permis de démontrer les lemmes générant des paires indispensables à partir de paires indispensables déjà connues. Nous montrerons dans cette sous-section comment ces inéquations peuvent également permettre d'obtenir une borne inférieure au coût optimal, de détecter des problèmes sans solution et de détecter des ensembles indispensables ou optimalement indispensables.

Trouver une borne inférieure

Une première utilisation des inéquations est qu'elles permettent de trouver une bonne borne inférieure au coût optimal comme nous le montre l'exemple ci-dessous (que nous généraliserons ensuite).

Exemple 5.3.46 *Considérons le problème de planification* $\Pi = \langle \{A1, A2, A3, A4\}, I = \{c\}, B = \{a, b\}\rangle$ *de l'exemple 5.3.23 et dont nous rappelons ici les quatres actions :*

$A1 = \langle \{a\}, \{b, \neg a\}, 1\rangle$
$A2 = \langle \{c\}, \{a, \neg c\}, 1\rangle$
$A3 = \langle \{d\}, \{b, \neg d\}, 1\rangle$
$A4 = \langle \{b\}, \{c, d, \neg b\}, 1\rangle$

Pour trouver une borne inférieure au coût d'une solution optimale, nous cherchons une borne inférieure au nombre d'occurences des actions dans tous les plans-solutions. L'application du lemme 5.3.28 aux fluents a, b et d (qui ne sont pas effets-stricts) produit les inégalités : $e_a \leq o_{A2}$, $e_b \leq o_{A1} + o_{A3}$ *et* $e_d \leq o_{A4}$. *Ces dernières nous permettent d'obtenir l'inégalité*

$$o_{A1} + o_{A2} + o_{A3} + o_{A4} \geq e_a + e_b + e_d$$

L'application du lemme 5.3.24 aux fluents a, b et d produit, après un réarrangement, les inégalités : $e_a \geq 1 + e_{\neg a}$, $e_b \geq 1 + e_{\neg b}$ *et* $e_d \geq e_{\neg d}$ *qui nous permettent d'obtenir que*

$$e_a + e_b + e_d \geq 2 + e_{\neg a} + e_{\neg b} + e_{\neg d}$$

Comme les fluents $\neg a$, $\neg b$ *et* $\neg d$ *sont effets-stricts, l'application du lemme 5.3.28 produit les égalités :* $e_{\neg a} = o_{A1}$, $e_{\neg b} = o_{A4}$ *et* $e_{\neg d} = o_{A3}$ *qui nous permettent d'obtenir que*

$$2 + e_{\neg a} + e_{\neg b} + e_{\neg d} = 2 + o_{A1} + o_{A4} + o_{A3}$$

Comme $e_b \leq o_{A1} + o_{A3}$ *et* $e_b \geq 1 + e_{\neg b}$, *nous obtenons*

$$2 + o_{A1} + o_{A4} + o_{A3} \geq 3 + e_{\neg b} + o_{A4}$$

Enfin, comme $e_{\neg b} = o_{A4}$, *nous obtenons l'inégalité*

$$o_{A1} + o_{A2} + o_{A3} + o_{A4} \geq 3 + 2o_{A4}$$

Le coût de chaque action étant de 1, nous pouvons en déduire une borne inférieure de 3 au coût d'un plan optimal P. Nous pouvons obtenir une meilleure borne en détectant au préalable les actions indispensables. L'action A4 est indispensable, ce qui signifie qu'elle appartient à tous les plans-solutions. Le lemme 5.3.38 peut être appliqué à cette action indispensable et nous dit que $o_{A4} \geq 1$. *Nous pouvons donc en déduire que*

$$o_{A1} + o_{A2} + o_{A3} + o_{A4} \ \geq \ 5$$

ce qui correspond exactement au coût du plan optimal $\langle A2, A1, A4, A3, A2 \rangle$.

De manière générale, nous cherchons une borne inférieure au coût d'un plan-solution optimal P. Ceci revient à résoudre le problème suivant :

$$\text{minimiser} \sum_{a \in A} o_a cout(a)$$

tel que les inégalités obtenues par les lemmes précédents soient satisfaites et que chaque variable $o_a, e_f, e_f^{utile}, i_f^{utile}$ soit un entier non-négatif. Ce problème est quasiment un programme linéaire en nombre entier. Tous les lemmes sont des inégalités linéaires entre les variables $o_a, e_f, e_f^{utile}, i_f^{utile}$, excepté pour le lemme 5.3.34. Cependant, nous pouvons transformer $o_a > 0 \ \Rightarrow \ e_f + i_f^{utile} > 0$ en l'inégalité linéaire $M(e_f + i_f^{utile}) > o_a$ pour une constante suffisamment grande M. Si nous résolvons la relaxation linéaire résultant du programme en nombre entier, nous pouvons obtenir des valeurs non entières pour les variables o_a mais cette résolution fournira néanmoins une borne inférieure utile de $cout(P)$. Si le nombre d'actions et de fluents n'est pas excessif, nous pouvons

également envisager une recherche exhaustive pour résoudre le programme linéaire en nombre entier.

Nous pouvons envisager de généraliser la planification optimale en associant des coûts aux fluents : $cout(f)$ est le coût de chaque établissement du fluent f. Dans ce cas, la somme à minimiser afin d'obtenir une borne inférieure du coût d'un plan-solution est

$$\sum_{a \in A} o_a \times cout(a) + \sum_{f} e_f \times cout(f)$$

Nous montrons maintenant un exemple de problème avec des coûts non-uniformes.

Exemple 5.3.47 *Considérons comme exemple de problème avec des coûts non-uniformes celui qui consiste à passer d'un état a à un état d en passant par un des deux états intermédiaires possibles que sont les états b et c. Le problème de planification peut être modélisé par quatre actions :*

$C1 = \langle \{a\}, \{b, \neg a\}, 1 \rangle$
$C2 = \langle \{b\}, \{d, \neg b\}, 100 \rangle$
$C3 = \langle \{a\}, \{c, \neg a\}, 10 \rangle$
$C4 = \langle \{c\}, \{d, \neg c\}, 50 \rangle$

L'état initial est $I = \{a\}$ et le but $B = \{d\}$. Les fluents $\neg a, \neg b, \neg c, \neg d$ sont tous effets-stricts ($\neg d$ trivialement car $\neg d$ n'appartient à aucun effet d'action). Ainsi, nous obtenons les inégalités suivantes à partir du lemme 5.3.24

$$-1 \leq e_a - e_{\neg a} \leq 0, \quad 0 \leq e_b - e_{\neg b}, \quad 0 \leq e_c - e_{\neg c}, \quad 1 \leq e_d - e_{\neg d}$$

et les inégalités et les équations suivantes à partir du lemme 5.3.28

$$e_a \leq 0, \quad e_b \leq o_{C1}, \quad e_c \leq o_{C3,} \quad e_d \leq o_{C2} + o_{C4},$$
$$e_{\neg a} = o_{C1} + o_{C3}, \quad e_{\neg b} = o_{C2}, \quad e_{\neg c} = o_{C4}, \quad e_{\neg d} = 0$$

Dans un plan optimal minimal P, nous avons également les inégalités suivantes à partir du lemme 5.3.44

$$o_{C1} \leq o_{C2}, \quad o_{C3} \leq o_{C4}$$

En éliminant les variables e_f, pour tous les fluents f, nous obtenons

$$o_{C1} = o_{C2}, \quad o_{C3} = o_{C4}, \quad o_{C1} + o_{C3} = o_{C2} + o_{C4} = 1$$

Ainsi, pour un plan optimal minimal P

$$\begin{aligned} cout(P) &= o_{C1} + 100o_{C2} + 10o_{C3} + 50o_{C4} \\ &= 101o_{C1} + 60o_{C3} \\ &= 60 + 41o_{C1} \\ &\geq 60 \end{aligned}$$

En effet, 60 est le coût du plan optimal $\langle C3, C4 \rangle$.

Détection des problèmes sans solution

Les approches par programmation linéaire peuvent parfois détecter des problèmes de planification qui n'ont pas de solution quand la relaxation classique ne le détecte pas. Nous illustrons cette idée avec l'exemple suivant.

Exemple 5.3.48 *Considérons un problème de planification $\Pi = \langle A, I, B\rangle$, où $I = \{a, b\}$, $B = \{a, b, c\}$ et A contient deux actions :*

$$B1 = \langle\{a\}, \{c, \neg a\}, 1\rangle$$
$$B2 = \langle\{b\}, \{c, \neg b\}, 1\rangle$$

Ce problème n'a pas de solution, mais ce fait n'est pas détecté en utilisant la relaxation classique car le niveau 1 du graphe de planification relaxé contient le but : $F^+[1] = B$.

Cependant, comme $\neg a$, $\neg b$ et $\neg c$ sont des fluents effets-stricts, nous déduisons des lemmes précédents :

$$1 \leq e_c - e_{\neg c} \quad 0 \leq e_{\neg a} - e_a \quad 0 \leq e_b - e_b \quad \textit{(lemme 5.3.24)}$$
$$e_a \leq 0 \quad e_b \leq 0 \quad e_c \leq o_{B1} + o_{B2} \quad \textit{(lemme 5.3.28)}$$
$$e_{\neg a} = o_{B1} \quad e_{\neg b} = o_{B2} \quad e_{\neg c} = 0 \quad \textit{(lemme 5.3.28)}$$

Ce système d'inéquations n'a pas de solution, ce qui permet de prouver que le problème de planification n'a pas de solution.

Considérons un deuxième exemple.

Exemple 5.3.49 *Considérons un simple problème de planification contenant deux actions :*

$$D1 = \langle\{a\}, \{b\}, 1\rangle$$
$$D2 = \langle\{b\}, \{a\}, 1\rangle$$

L'état initial est $I = \emptyset$ et le but est $B = \{a, b\}$. Clairement, ce problème n'a pas de solution, ce qui est déterminé immédiatement par l'analyse du problème relaxé. Cependant, le programme linéaire obtenu trouve une solution faisable : $o_{D1} = o_{D2} = e_a = e_b = 1$. Cet exemple démontre que l'utilisation d'une relaxation du problème et l'approche par programmation linéaire sont complémentaires.

Détection d'ensembles indispensables et élimination d'actions

L'approche par programmation linéaire permet donc de détecter certains problèmes de planification qui n'ont pas de solution. Nous pouvons ainsi l'utiliser pour détecter des paires indispensables : si le programme linéaire obtenu à partir de $\Pi_{-(X,U)}$ n'a pas de solution, nous pouvons en déduire que $\langle X, U\rangle$ est une paire indispensable. Quand le programme linéaire a une solution, alors il fournit une borne inférieure au coût du plan-solution optimal. Ceci a une utilité évidente durant la recherche d'un plan optimal, mais peut également être utilisé pour détecter des paires optimalement indispensables. Si le programme linéaire obtenu à partir de $\Pi_{-(X,U)}$ fournit une borne inférieure qui est plus grande que le coût d'un plan-solution déjà obtenu pour Π, nous pouvons en déduire que $\langle X, U\rangle$ est une paire optimalement indispensable.

L'approche par programmation linéaire peut également être utilisée pour éliminer des actions de A pendant le prétraitement. Si le programme linéaire obtenu à partir de Π augmenté de l'inégalité $o_a \geq 1$ n'a pas de solution, ou fournit une borne inférieure qui est plus grande que le coût d'un plan-solution déjà obtenu pour Π, alors nous pouvons déduire que a ne peut pas faire partie d'un plan optimal minimal.

5.4 Transformation d'un problème de planification

La transformation en temps polynomial d'une instance d'un problème combinatoire en un autre problème équivalent mais plus facile à résoudre est une approche classique en Intelligence Artificielle. Par exemple, la notion d'arc consistance est omniprésente dans les techniques de satisfaction de contraintes comme nous l'avons vu dans le chapitre 3 avec les différentes cohérences d'arc pour les WCSP. Nous montrerons dans cette section comment un problème de planification peut être transformé en un autre problème équivalent mais plus facile à résoudre. Ces techniques sont fortement inspirées des techniques de cohérences d'arc pour les WCSP.

Nous définirons une transformation qui conserve les coûts des plans-solutions que nous illustrerons sur un exemple. L'opération de tranformation locale que nous proposerons nécessite que les problèmes de planification appartiennent à une certaine classe de problèmes et nous montrerons comment la plupart des problèmes de planification peuvent être réduits à cette classe. Nous montrerons également comment transformer un problème de planification optimal effet-strict et normalisé en un problème équivalent en coût. Nous montrerons enfin comment obtenir une transformation optimale.

5.4.1 Problème de planification équivalent en coût

La transformation d'un problème de planification en un problème de planification plus facile à résoudre doit conserver les mêmes solutions avec les mêmes coûts :

Définition 5.4.1 (Transformation préservant le coût, CPT) *Une transformation qui préserve le coût (CPT, Cost-Preserving Transformation) d'un problème de planification optimal* Π *est une opération qui transforme* Π *en un problème de planification optimal* Π' *tel que* Π *et* Π' *ont le même ensemble de plans-solutions et tel que pour chaque plan-solution* P*,*

$$cout_{\Pi}(P) = cout_{\Pi'}(P)$$

Nous dirons que Π *et* Π' *sont équivalents en coût.*

Il est important de noter qu'une CPT ne préserve pas, en général, le coût des actions ou des plans partiels. En effet, par la définition 5.4.1, tous les problèmes Π qui n'ont aucun plan-solution sont considérés comme équivalents en coût.

Nous utiliserons ici principalement les CPT pour transformer un problème Π en un autre problème Π' dans lequel la borne inférieure du coût d'un plan-solution optimal est plus explicite. Mais elles peuvent être utilisées pour d'autres applications : égaliser le coût des actions afin de trouver une meilleure borne supérieure au nombre d'actions dans un plan optimal ou, au

contraire, augmenter le coût d'une action particulière afin de montrer qu'elle est trop onéreuse pour faire partie d'un plan optimal. Nous illustrons ces idées par un exemple. Soit un problème de transport Π dans lequel, pour aller de A à B, nous avons deux choix :

1. prendre un taxi en A jusqu'à l'aéroport le plus proche (coût 20 euros) ; prendre un avion jusqu'à l'aéroport le plus proche de B (coût 140 euros) ; prendre un taxi de cet aéroport jusqu'à B (coût 20 euros).
2. prendre un bus en A jusqu'à la gare la plus proche (coût 2 euros) ; prendre un train jusqu'à la gare la plus proche de B (coût 152 euros) ; prendre un bus de cette gare jusqu'à B (coût 2 euros).

Ce problème est clairement équivalent en coût à un problème identique Π' dans lequel le coût d'un trajet en taxi est de 60 euros, un vol en avion est de 60 euros, un trajet en bus est de 52 euros et un trajet en train coûte 52 euros. Dans cette version, le coût minimum d'une action est de 52 euros alors qu'il est de 2 euros dans le problème original. Une fois que le plan-solution (bus-train-bus) de coût total 156 euros a été trouvé, nous pouvons déduire qu'il n'existe pas de meilleur plan comprenant plus de deux actions, car dans Π' chaque action à un coût d'au moins 52 euros. Dans une autre version Π'' équivalente en coût au même problème, les taxis sont gratuits mais un trajet en avion coûte 180 euros, nous pouvons en déduire immédiatement que cette action ne peut pas faire partie d'un plan optimal, car nous avons déjà un plan coûtant seulement 156 euros. Π est également équivalent en coût au problème Π_{opt} qui est identique à Π excepté que le coût d'un taxi à partir de A vers l'aéroport est de 24 euros, toutes les autres actions ont un coût de 0, mais il y a un coût de 156 euros à payer en arrivant à B (quel que soit le moyen de transport utilisé).

5.4.2 Réduction des problèmes de planification

Nous montrerons dans cette sous-section comment réduire des problèmes de planification à des problèmes de la classe des problèmes de planification effets-stricts et normalisés. Nous commençerons par présenter deux réductions vers la classe des problèmes de planification optimaux effets-stricts. Puis, après avoir défini un problème normalisé, nous montrerons comment réduire un problème effet-strict à un problème de la classe des problèmes de planification optimaux effets-stricts et normalisés.

Réductions vers la classe des problèmes de planification effets-stricts

Les deux lemmes de cette sous-section décrivent deux méthodes distinctes pour rendre un problème de planification optimal effet-strict. Nous étudierons également leurs possibles utilisations sur les problèmes issus des compétitions de planification IPC.

Définition 5.4.2 (Arité d'une action, Problème de planification k-local) *L'arité d'une action a est $|eff(a)|$, le nombre de fluents positifs ou négatifs appartenant à $eff(a)$. Un problème de planification optimal $\Pi = \langle A, I, B\rangle$ est k-local si l'arité de chaque action $a \in A$ est bornée par k.*

Lemme 5.4.3 *Il existe une réduction polynomiale de la classe des problèmes de planification optimaux k-locaux ayant un état initial complet, où k est une constante, vers la classe des problèmes de planification optimaux effets-stricts.*

Preuve : Soit $\Pi = \langle A, I, B\rangle$ un problème de planification optimal dans lequel I est un état complet et tel que, $\forall a \in A, |eff(a)| \leq k$. Pour chaque fluent $f \in F_A$ et une action $a \in A$ tels que $\neg f \in eff(a)$ et $f \notin prec(a)$, nous pouvons remplacer a par deux nouvelles actions a_1 et a_2 telles que

$$\begin{array}{lll} prec(a_1) = prec(a) \cup \{f\} & eff(a_1) = eff(a) & cout(a_1) = cout(a) \\ prec(a_2) = prec(a) \cup \{\neg f\} & eff(a_2) = eff(a) - \{\neg f\} & cout(a_2) = cout(a) \end{array}$$

L'action a_1 (a_2) est une version de a qui peut être appliquée dans un état E si f est vrai (faux) dans E ; a_1 supprime f tandis que a_2 laisse la valeur de f inchangée. Nous devons prouver que le problème résultant $\Pi_{(f,a)}$ est équivalent à Π. Soit P un plan-solution pour Π qui contient l'action a. Soit E un état qui est atteint durant l'application de P juste avant l'application de l'action a. Comme l'état I est complet, l'état E l'est aussi. Nous pouvons alors simplement remplacer a par a_1 ou a_2 selon si $f \in E$ ou $\neg f \in E$. Ainsi, un plan-solution pour Π peut être converti en temps polynomial en un plan-solution pour $\Pi_{(f,a)}$. Inversement, si P' est un plan-solution pour $\Pi_{(f,a)}$, alors il est trivial d'obtenir un plan-solution P pour Π, en remplaçant toutes les actions a_1 ou a_2 appartenant à P' par a.

Nous pouvons clairement appliquer la transformation ci-dessus pour chaque fluent f (positif ou négatif) et pour chaque action a tels que $\neg f \in eff(a)$ et $f \notin prec(a)$. Le problème de planification résultant Π' est effet-strict. La transformation de Π à Π' qui en résulte introduit 2^{k_a} copies de l'action a, où k_a est le nombre de fluents f pour lesquels $\neg f \in eff(a)$ et $f \notin prec(a)$. Comme $k_a \leq k$ et k est une constante, la transformation est une réduction polynomiale. ∎

Le lemme 5.4.3 fournit seulement une réduction polynomiale de la classe des problèmes de planification k-locaux à la classe des problèmes de planification effets-stricts. Le lemme suivant nous montre que si le problème original est retrait-strict et qu'il ne contient que des fluents positifs, nous n'avons pas besoin de borner l'arité des actions.

Lemme 5.4.4 *Il existe une réduction polynomiale de la classe des problèmes de planification optimaux retraits-stricts dans lesquels les préconditions d'actions ne contiennent que des fluents positifs vers la classe des problèmes de planification optimaux effets-stricts.*

Preuve : Soit $\Pi = \langle A, I, B\rangle$ un problème de planification optimal retrait-strict. Par hypothèse, toutes les actions $a \in A$ sont retraits-stricts. Pour chaque action $a \in A$, nous définissons l'action a' ainsi :

$$\begin{array}{rcl} prec(a') & = & prec(a) \cup \{\neg f | f \in add(a)\} \\ eff(a') & = & eff(a) \\ cout(a') & = & cout(a) \end{array}$$

Nous définissons également, pour chaque fluent positif $f \in F_A^+$, une nouvelle action factice d_f qui supprime simplement le fluent f, i.e.

$$\begin{array}{rcl} prec(d_f) & = & \{f\} \\ eff(d_f) & = & \{\neg f\} \\ cout(d_f) & = & 0 \end{array}$$

Toutes les actions a' (pour $a \in A$) et d_f (pour $f \in F_A^+$) sont effets-stricts. Soit $\Pi' = \langle A', I, B \rangle$, où

$$A' = \{a' | a \in A\} \cup \{d_f | f \in F_A^+\}$$

Π' est effet-strict. Il nous reste à montrer qu'il est équivalent à Π. Un plan-solution P pour Π peut être transformé en un plan-solution pour Π'

1. en remplaçant chaque action a appartenant à P par l'action a',
2. et en ajoutant une action factice d_f juste avant l'action a' quand $\neg f$ appartient aux préconditions de a' et qu'il n'est pas satisfait.

Inversement, un plan-solution P' pour Π' peut être transformé en un plan-solution pour Π par la suppression des actions factices d_f de P', et en remplaçant chaque action a' dans P' par l'action correspondante a. Comme les préconditions des actions de Π ne contiennent aucun fluent négatif, le plan résultant est valide. Dans les deux sens, les plans-solutions ont le même coût puisque les actions factices ont un coût nul. De plus, la réduction est clairement polynomiale. ∎

Nous avons étudié tous les problèmes des 26 domaines non-temporels STRIPS proposés par les compétitions de IPC'2000[1] [Bacchus, 2001], IPC'2002[2] [Long et Fox, 2003], IPC'2004[3] [Hoffmann *et al.*, 2006], IPC'2006[4] [Dimopoulos *et al.*, 2006] et IPC'2008[5]. Les 26 domaines étudiés sont : *Schedule*, *Logistics*, *Freecell*, *Elevator*, *Blocks*, *Depots*, *Driver*, *Zeno*, *Rovers*, *Satellite*, *Airport*, *Pipesworld*, *Promela optical telegraph*, *Promela philosophers*, *Psr*, *Pathways*, *Storage*, *Tpp*, *Trucks*, *Openstacks*, *ParcPrinter*, *Pegsol*, *Scanalyser*, *Sokoban*, *Transport* et *Woodworking*.

- Pour 19 de ces domaines (qui sont *Schedule*, *Logistics*, *Freecell*, *Elevator*, *Blocks*, *Depots*, *Driver*, *Zeno*, *Rovers*, *Psr*, *Pathways*, *Tpp*, *Trucks*, *ParcPrinter*, *Openstacks*, *Pegsol*, *Scanalyser*, *Sokoban*, *Transport*), toutes les instances sont à la fois positives et retraits-stricts, le Lemme 5.4.4 peut donc être appliqué.
- Dans 4 des 7 domaines restants, la valeur maximum de k_a (pour $a \in A$) est relativement petite : 2 pour *Satellite* ; 3 pour *Storage* ; 6 pour *Pipesworld* ; 7 pour *Woodworking*. Nous pouvons donc envisager, pour ces 4 domaines, d'appliquer la transformation décrite dans la preuve du Lemme 5.4.3.
- La valeur maximum de k_a est atteinte dans les domaines *Airport*, *Promela optical telegraph* et *Promela philosophers* avec des valeurs comprises entre 12 (pour *Airport*) et 178 (pour le problème $pfile7$ de *Promela optical telegraph*).

Nous pouvons donc en conclure que, dans la majorité des benchmarks, nous pouvons transformer le problème en un problème équivalent et effet-strict sans augmenter de manière excessive la taille de l'ensemble des actions.

Réduction vers la classe des problèmes de planification effets-stricts normalisés

Nous montrons dans cette sous-section qu'il est toujours possible de convertir un problème de planification en une version normalisée.

[1] http ://www.cs.toronto.edu/aips2000/
[2] http ://planning.cis.strath.ac.uk/competition/
[3] http ://ls5-web.cs.uni-dortmund.de/ edelkamp/ipc-4/
[4] http ://zeus.ing.unibs.it/ipc-5/
[5] http ://ipc.informatik.uni-freiburg.de/

Définition 5.4.5 (Problème de planification normalisé) *Un problème de planification optimal $\Pi = \langle A, I, B\rangle$ est normalisé si l'état initial est $\{init\} \cup \{\neg f | f \in F_A^+ - \{init\}\}$ et l'état but est $\{but\} \cup \{\neg f | f \in F_A^+ - \{but\}\}$.*

Lemme 5.4.6 *Il existe une réduction polynomiale de la classe des problèmes de planification optimaux effets-stricts avec un état initial complet vers la classe des problèmes de planification optimaux effets-stricts normalisés.*

Preuve : Soit $\Pi = \langle A, I, B\rangle$ un problème de planification optimal effet-strict, avec I un état complet. Nous introduisons deux nouveaux fluents *init* et *but* pour lesquels on peut considérer, sans perte de généralité, qu'ils ne sont pas présents dans F_A^+. Soit $F = F_A^+ \cup \{init, but\}$. Nous définissons les actions action-initialisation a_I et action-but a_B ainsi :

$$\begin{aligned} prec(a_I) &= \{init\} \cup \{\neg f | f \in F - \{init\}\} \\ eff(a_I) &= I^+ \cup \{\neg init\} \\ cout(a_I) &= 0 \end{aligned}$$

$$\begin{aligned} prec(a_B) &= B \cup \{\neg f | f \in F - B^+\} \\ eff(a_B) &= \{\neg f | f \in B^+\} \cup \{but\} \\ cout(a_B) &= 0 \end{aligned}$$

Le résultat de l'application de l'action a_I est un état complet $I \cup \{\neg init, \neg but\}$. La précondition de l'action a_B est également un état complet dans lequel tous les fluents n'appartenant pas à B prennent la valeur faux. Afin de pouvoir affecter les fluents de F_A^+ qui n'appartiennent pas au but à faux, nous définissons, pour chaque fluent $f \in F_A^+ - B^+$, une action factice d_f^B telle que :

$$\begin{aligned} prec(d_f^B) &= B \cup \{f\} \\ eff(d_f^B) &= \{\neg f\} \\ cout(d_f^B) &= 0 \end{aligned}$$

Les actions a_I, a_B et d_f^B (pour $f \in F_A^+ - B^+$) sont toutes effets-stricts. L'effet résultant de l'action d_f^B est simplement de retirer f ; cette action joue le même rôle que l'action factice d_f dans la preuve du Lemme 5.4.4 (sauf que d_f^B ne peut être appliquée qu'après la satisfaction de B).

Soit $\Pi' = \langle A', I', B'\rangle$, tel que

$$\begin{aligned} A' &= A \cup \{a_I, a_B\} \cup \{d_f^B | f \in F_A^+ - B^+\} \\ I' &= \{init\} \cup \{\neg f | f \in F - \{init\}\} \\ B' &= \{but\} \cup \{\neg f | f \in F - \{but\}\} \end{aligned}$$

Π' est effet-strict et normalisé. Il reste à prouver qu'il est équivalent à Π. Un plan-solution P de Π peut être transformé en un plan-solution de Π' en préfixant P par une action-initialisation a_I, en terminant P par une action-but a_B et en insérant une action d_f^B juste avant a_B pour tous

les fluents positifs $f \in F_A^+ - B^+$ qui sont vrais à la fin de l'application de P. Inversement, un plan-solution P' pour Π' peut être transformé en un plan-solution pour Π en supprimant dans P' toutes les actions a_I, a_B et d_f^B (pour $f \in F_A^+ - B^+$). Dans les deux sens, les plans-solutions ont les mêmes coûts car l'action-initialisation, l'action-but et les actions factices ont un coût nul. Cette réduction est clairement polynomiale. ∎

5.4.3 Transformation en un problème équivalent en coût

Nous décrirons dans cette sous-section l'opération **Arc-Mouvement**(f, δ) qui permet de transformer localement un problème de planification en un problème plus facile à résoudre. Alors que cette transformation sera locale, nous montrerons comment obtenir une transformation optimale d'un problème en un problème équivalent en coût et qui maximise la valeur de la borne inférieure au coût d'une solution optimale. Nous terminerons par un exemple et nous montrerons les implications de cette transformation sur les résultats précédents.

Dans la suite de cette section, nous considérerons des problèmes de planification optimaux effets-stricts et normalisés. Nous supposerons que l'une des transformations décrites dans les lemmes 5.4.3 ou 5.4.4 a été appliquée, suivi par la transformation décrite dans le lemme 5.4.6. Notons que si le problème original est positif et retrait-strict, il n'est pas nécessaire d'introduire les actions fictives d_f^B car les actions fictives d_f (décrites dans la preuve du lemme 5.4.4) remplissent déjà ce rôle.

Opération de transformation locale

L'algorithme 6 décrit la transformation **Arc-Mouvement**(f, δ) qui permet de déplacer un coût δ entre les actions liées à un fluent f. Sans perte de généralité, nous supposerons que f est un fluent positif et $\delta \in \mathbb{R}$. Comme le montre le lemme suivant, cette transformation préserve les coûts dans un problème de planification optimal Π effet-strict et normalisé.

Algorithme 6 Arc-Mouvement$(f \in F_A^+, \delta \in \mathbb{R})$

pour tout $a \in eff_f$ **faire**
  $cout(a) \leftarrow cout(a) - \delta$;
fin pour
pour tout $a \in eff_{\neg f}$ **faire**
  $cout(a) \leftarrow cout(a) + \delta$;
fin pour

Lemme 5.4.7 *Soit Π un problème de planification optimal normalisé dans lequel f est un fluent positif tel que $f \notin \{init, but\}$. Si f et $\neg f$ sont des fluents effets-stricts, alors* ***Arc-Mouvement****(f, δ) est une transformation qui préserve les coûts.*

Preuve : Soit P un plan-solution pour Π. Soit e_f ($e_{\neg f}$) le nombre de fois que f (respectivement $\neg f$) est établi durant l'application de P et o_a le nombre de fois que l'action a apparaît dans P.

Comme Π est normalisé et f est un fluent positif tel que $f \notin \{init, but\}$, $\neg f$ apparaît à la fois dans l'état initial et dans l'état but. En appliquant le Corollaire 5.3.26, nous obtenons

$$0 \ \leq \ e_f - e_{\neg f} \ \leq \ 0$$

et donc $e_f = e_{\neg f}$. Si f et $\neg f$ sont effets-stricts, par le Lemme 5.3.28, nous avons

$$e_f \ = \ \sum_{a \in eff_f} o_a, \qquad e_{\neg f} \ = \ \sum_{a \in eff_{\neg f}} o_a$$

Par conséquent, le coût de P dans le problème transformé Π' est

$$cout_{\Pi'}(P) \ = \ cout_{\Pi}(P) \ - \ e_f\delta \ + \ e_{\neg f}\delta \ = \ cout_{\Pi}(P)$$

■

Transformation optimale en un problème équivalent en coût

L'opération **Arc-Mouvement**(f, δ) permet de transformer un problème de planification effet-strict et normalisé en un problème équivalent en coût. Mais cette opération est juste une opération locale et il peut être beaucoup plus productif d'appliquer simultanément un ensemble d'opérations de transformations par arc-mouvement. En effet, nous avons vu qu'un but commun aux problèmes combinatoires de minimisation consiste à trouver une borne inférieure à la valeur d'une solution optimale. De plus, l'obtention d'une borne inférieure est essentielle dans des algorithmes de recherche intelligents impliquant l'élagage de l'arbre de recherche. Par conséquent, nous cherchons à maximiser cette borne inférieure. Si a_B est l'action-but, alors nous devons maximiser $cout(a_B)$. Bien sûr, ceci est une borne inférieure au coût du plan-solution optimal seulement si tous les autres coûts sont non-négatifs.

Un ensemble d'arc-mouvements est clairement équivalent à un ensemble d'appels de l'opération **Arc-Mouvement**(f, δ_f) où f varie entre tous les fluents de $F_A^+ - \{init, but\}$. Pour simplifier la notation dans cette section, toutes les sommes où les quantifications de fluents sont supposées être sur tous les fluents positifs de $F_A^+ - \{init, but\}$.

Pour des coûts réels finis, les appels à **Arc-Mouvement** commutent. Notons que les coûts originaux des actions peuvent être négatifs (représentant un bénéfice gagné par l'application de l'action), à condition que dans la version transformée Π', le coût de chaque action a soit non négatif :

$$cout(a) \ - \ \sum_{f \in eff(a)} \delta_f \ + \ \sum_{\neg f \in eff(a)} \delta_f \ \geq \ 0$$

Par conséquent, le problème qui consiste à trouver la meilleur borne inférieure peut être exprimé par le programme linéaire suivant :

LP1 : maximiser $\sum_{f \in B} \delta_f$ tel que
pour toutes les actions $a \in A - \{a_B\}$,
$\sum_{f \in eff(a)} \delta_f \ - \ \sum_{\neg f \in eff(a)} \delta_f \ \leq \ cout(a)$

Nous n'imposons pas la contrainte que $cout(a_B)$ soit non négatif dans Π', car c'est exactement cette valeur que nous essayons de maximiser. En effet, si on permet des profits, la valeur maximum de $\sum_{f \in B} \delta_f$ peut être négative.

Définition 5.4.8 (Transformation arc-optimale) *Un problème de planification optimal effet-strict et normalisé avec une action-but a_B est transformé de manière arc-optimale si aucun ensemble d'opérations d'arc-mouvement ne peut incrémenter $cout(a_B)$ tout en conservant des coûts non-négatifs pour toutes les autres actions.*

Théorème 5.4.9 *Si Π est un problème de planification optimal effet-strict et normalisé, alors un problème Π', transformé de manière arc-optimale et équivalent en côut à Π, peut être trouvé en un temps polynomial.*

Preuve : Ceci suit immédiatement de la discussion ci-dessus et à partir du fait que la programmation linéaire peut être résolue en un temps polynomial [Karmarkar, 1984 ; Schrijver, 1998].

∎

Nous illustrons cette transformation par un exemple.

Exemple 5.4.10 *Considérons un problème de planification contenant les dix actions suivantes :*

$$
\begin{aligned}
&A1 = \langle \{a, \neg b\}, \{b, \neg a\}, 1 \rangle \\
&A2 = \langle \{c, \neg a\}, \{a, \neg c\}, 1 \rangle \\
&A3 = \langle \{d, \neg b\}, \{b, \neg d\}, 1 \rangle \\
&A4 = \langle \{b, \neg c, \neg d\}, \{c, d, \neg b\}, 1 \rangle \\
&a_I = \langle \{\neg a, \neg b, \neg c, \neg d, \neg but, init\}, \{c, \neg init\}, 0 \rangle \\
&a_B = \langle \{a, b, \neg c, \neg d, \neg init, \neg but\}, \{\neg a, \neg b, but\}, 0 \rangle \\
&d_f = \langle \{f\}, \{\neg f\}, 0 \rangle \quad \textit{pour chaque } f \in \{a, b, c, d\}
\end{aligned}
$$

Ce problème Π est équivalent au problème de l'exemple 5.3.46. Il a été transformé selon les preuves des lemmes 5.4.4 et 5.4.6. Il est effet-strict et normalisé.

En appliquant l'approche précédente pour trouver la transformation de Π qui maximise la somme des coûts sur les fluents de but, nous obtenons le programme linéaire suivant :

$$
\begin{aligned}
&\textit{maximiser} \quad \delta_a + \delta_b \\
&\textit{tel que} \quad \delta_b - \delta_a \leq 1 \\
&\qquad \delta_a - \delta_c \leq 1 \\
&\qquad \delta_b - \delta_d \leq 1 \\
&\qquad \delta_c + \delta_d - \delta_b \leq 1 \\
&\qquad \delta_c \leq 0 \\
&\qquad -\delta_f \leq 0 \quad \textit{pour chaque } f \in \{a, b, c, d\}
\end{aligned}
$$

Ce programme linéaire a une solution optimale $\delta_a = 1$, $\delta_b = 2$, $\delta_c = 0$, $\delta_d = 1$. Nous pouvons en déduire que Π est équivalent en coût à un problème identique dans lequel les coûts des actions

sont :

$$\begin{array}{lll} cout(\text{A1}) = 0 & cout(a_I) = 0 & cout(d_a^B) = 1 \\ cout(\text{A2}) = 0 & cout(a_B) = 3 & cout(d_b^B) = 2 \\ cout(\text{A3}) = 0 & & cout(d_c^B) = 0 \\ cout(\text{A4}) = 2 & & cout(d_d^B) = 1 \end{array}$$

Ainsi la borne inférieure au coût de tous les plans-solutions optimaux a une valeur de 3, car le coût de l'action action-but a_B a un coût de 3.

Considérons un problème de planification Π effet-strict et normalisé qui contient une action-but a_B comme donnée dans la preuve du Lemme 5.4.6. Par les arguments donnés dans la preuve du Lemme 5.4.7, pour chaque fluent positif $f \notin \{init, but\}$,

$$\sum_{a \in eff_f} o_a \;=\; \sum_{a \in eff_{\neg f}} o_a$$

que nous pouvons réécrire :

$$\sum_{a \in eff_f - \{a_B\}} o_a - \sum_{a \in eff_{\neg f} - \{a_B\}} o_a = b_f$$

où $b_f = 1$ si $f \in B$ (et 0 sinon), comme le fluent positif f appartient à B ssi $\neg f \in eff(a_B)$, il n'y a aucun fluent positif $f \notin \{init, but\}$ dans $eff(a_B)$ et nous avons clairement dans le plan-solution $o_{a_B} = 1$. Nous pouvons alors obtenir une borne inférieure au coût de la solution optimale à partir des techniques de la section 5.3, page 98 et en résolvant le programme linéaire suivant :

LP2 : minimiser $\sum_{a \in A - \{a_B\}} o_a cout(a)$
tel que $\forall f \in F_A^+ - \{init, but\}$,
$\sum_{a \in eff_f - \{a_B\}} o_a \;-\; \sum_{a \in eff_{\neg f} - \{a_B\}} o_a = b_f$
et $\forall a,\; o_a \geq 0$

C'est exactement le dual de **LP1**. Par le théorème fondamental de dualité [Trustrum, 1971], les bornes inférieures obtenues en résolvant les programmes linéaires **LP1** et **LP2** sont identiques à condition qu'au moins un des problèmes soit faisable. L'exemple 5.3.46 a démontré que **LP2** peut nous permettre d'obtenir une meilleure borne inférieure quand nous prenons en compte des informations supplémentaires telles que la connaissance des ensembles indispensables d'actions. Cependant, la résolution de **LP1** possède l'avantage de fournir, en plus d'une borne inférieure, un problème de planification optimal équivalent en coût.

Nous pouvons clairement résoudre un programme linéaire similaire à **LP1** pour déterminer si, par exemple, il existe un ensemble d'opérations d'**Arc-Mouvement** qui transforment Π en un problème équivalent dans lequel le coût de toutes les actions soit borné par une certaine constante C. En particulier, dans le cas où le problème original contient des actions avec un coût négatif (i.e. des actions de profits), nous pouvons chercher un problème équivalent avec des coûts strictement positifs. De même, nous pouvons résoudre un programme linéaire pour déterminer s'il existe une version de Π, transformée par arc-mouvement, dans laquelle $cout(a) > M$ où M est le coût de la meilleure solution trouvée jusque là. Si tel est le cas, alors nous pouvons en déduire que l'action a ne peut appartenir à aucune solution optimale.

5.5 Synthèse

Nous avons présenté dans ce chapitre différentes techniques permettant d'extraire des informations à partir d'une analyse des domaines des problèmes.

Nous avons commencé par décrire la technique de relaxation la plus couramment utilisée dans le cadre non valué. Nous avons montré que la résolution de coût optimal des problèmes qui résultent de cette relaxation était NP-Difficile. Nous avons alors proposé différentes méthodes qui utilisent cette relaxation pour obtenir des informations plus précises que celles fournies par d'autres stratégies :

- Détermination, lors d'une phase de prétraitement, des ensembles d'actions, de fluents (ou une combinaison des deux) qui sont indispensables à la résolution du problème.
- Détermination, au cours de la recherche, des ensembles d'actions, de fluents (ou une combinaison des deux) qui sont indispensables à l'obtention de solutions de coût optimal.

Nous avons montré comment encoder ces informations dans un WCSP issu d'un problème de planification. Les expérimentations démontrent cependant que l'ajout de ces informations dans le WCSP n'améliore pas les performances. Nous verrons dans le chapitre suivant comment ces ensembles peuvent néanmoins être utilisés avec succès.

Nous avons montré comment construire des ensembles d'inégalités linéaires à partir du nombre d'occurences des actions et des fluents contenus dans les solutions. Ce programme en nombre entier nous a permis de calculer une borne inférieure au coût d'un plan-solution optimal. Il peut également fournir une méthode alternative pour détecter des problèmes de planification qui n'ont aucune solution et permettre ainsi de détecter des ensembles indispensables ou optimalement indispensables.

Nous avons aussi proposé une transformation d'un problème de planification en un autre problème équivalent en coût mais plus facile à résoudre. Nous avons ensuite montré comment réduire les problèmes de planification vers la classe des problèmes de planification effets-stricts et normalisés. Nous avons alors présenté une opération de transformation et montré comment obtenir un ensemble de transformations tel que le problème équivalent obtenu maximise la borne inférieure du coût de la solution optimale.

Nous avons ainsi introduit plusieurs méthodes pouvant être utilisées dans différentes approches, par exemple pour détecter des problèmes sans solution, pour simplifier le problème mais également pour connaître l'importance des actions ou des ensembles d'actions dans les plans-solutions non optimaux et optimaux. Dans le chapitre suivant, nous modifions le planificateur GP-WCSP pour lui permettre d'obtenir une solution de coût optimal. L'algorithme ainsi obtenu n'est pas utilisable en pratique et nous montrons comment les méthodes introduites précédemment peuvent nous permettre de l'améliorer.

Chapitre 6

Solution optimale globale

Nous avons vu dans le chapitre 4 comment obtenir une solution k-optimale qui est une solution de coût optimal parmi les solutions de longueur inférieure ou égale à k niveaux. Cependant, dans de nombreux problèmes, cette solution n'est pas une solution de coût optimal de manière globale. Par exemple, dans des problèmes de transports, si le plan le plus court implique un vol direct onéreux, il existe souvent un plan plus long mais moins coûteux qui est composé d'un itinéraire complexe impliquant plusieurs trains ou autobus.

Dans ce chapitre, nous présenterons une nouvelle approche permettant d'obtenir une solution de coût optimal (et qui minimise le nombre de niveaux) [Cooper *et al.*, 2007c,b, 2008c]. Cette approche sera implémentée dans le planificateur GP-WCSP* et sera présentée dans la section 6.1. GP-WCSP* sera basé sur le planificateur GP-WCSP et poursuivra le développement du graphe après l'obtention d'une première solution jusqu'à garantir l'obtention d'une solution de coût optimal. Nous comparerons ensuite, dans la section 6.2, page 134, les résultats obtenus avec des planificateurs non-optimaux. Nous terminerons en décrivant, dans la section 6.3, page 135, une comparaison de GP-WCSP* avec des planificateurs de coût optimaux.

6.1 GP-WCSP*

Cette section présentera le planificateur GP-WCSP* en commençant par son algorithme. Il garantira un plan-solution P^* de coût optimal lorsqu'il atteindra une borne supérieure au nombre de niveaux à développer. Nous montrerons, dans la sous-section 6.1.2, comment cette borne peut être largement améliorée en utilisant des informations extraites des domaines et des problèmes qui ont été présentées dans le chapitre précédent. Nous illustrerons ensuite cette méthode et ces améliorations par un exemple dans la sous-section 6.1.3. Nous présenterons enfin dans la sous-section 6.1.4, les résultats expérimentaux de GP-WCSP*.

6.1.1 Algorithme de GP-WCSP*

L'algorithme anytime GP-WCSP* que nous avons développé est basé sur celui de GP-WCSP. Une fois la phase d'expansion du graphe de planification terminée, le graphe est codé en un WCSP équivalent puis résolu (cf. chapitre 4, page 55). Le coût du plan-solution k-optimal trouvé à un niveau k du graphe nous sert ensuite de majorant pour relancer la recherche d'une meilleure solution au niveau suivant.

L'utilisation du majorant nous permet d'améliorer la recherche d'une solution de coût optimal. En effet, nous avons vu, dans le chapitre précédent, qu'un plan-solution k-optimal permet de déterminer des actions trop onéreuses à partir du niveau k (cf. sous-section 5.2.3, page 90). Ces actions sont trop coûteuses pour appartenir à une solution de coût optimal et peuvent être supprimées de l'espace de recherche. Cette opération est effectuée dans notre algorithme lors de la construction du graphe qui utilise uniquement les actions appartenant à l'ensemble d'actions donné en paramètre.

Le majorant permet également d'améliorer la résolution des WCSP. Ainsi, le solveur de WCSP ne cherche plus une solution de coût optimal mais une solution de coût optimal strictement inférieur à C_k^*. L'utilisation de ce majorant permet d'élaguer de nombreuses branches de l'arbre de recherche lors de la résolution du WCSP.

L'algorithme de GP-WCSP* garantit un plan-solution P^* de coût optimal lorsqu'il atteint une borne supérieure au nombre de niveaux à développer (borne appelée $NivMax$). Une première estimation de cette borne peut-être facilement calculée à partir du premier plan-solution P_k^* obtenu au niveau k. Seules les actions de A_r^+ sont susceptibles de mener au but. Soit C_{min} le coût minimum de ces actions :

$$C_{min} = \min_{a \in A_r^+} cout(a) \tag{6.1}$$

Dans le pire des cas, un plan optimal P^* aurait un coût $C^* = C_k^* - \epsilon$ (où $0 < \epsilon < C_{min}$) et serait un plan séquentiel composé de $(C_k^* - \epsilon)/C_{min}$ actions dont le coût ne pourrait être inférieur à C_{min}. Ainsi, le nombre de niveaux à développer nécessaires pour obtenir un plan optimal est au plus égal à :

$$NivMax = \left\lfloor \frac{C_k^* - \epsilon}{C_{min}} \right\rfloor = \left\lceil \frac{C_k^*}{C_{min}} \right\rceil - 1 \tag{6.2}$$

Le fait d'obtenir une valeur de $NivMax$ inférieure ou égale au niveau du graphe courant signifie que le plan-solution déjà trouvé est optimal. Nous verrons dans la section suivante comment améliorer la valeur de cette borne.

Exemple 6.1.1 *Dans l'exemple 4.1, page 55, l'action de plus petit coût qui est potentiellement utile à l'obtention du but est* $poser_B$ *de coût* 3*, la première estimation de la valeur de* $NivMax$ *faite avec le premier plan-solution obtenu* P_3^* *donne* $NivMax = 35$*.*

L'algorithme de GP-WCSP* (algorithme 7) donne un planificateur semi-complet car si un plan-solution existe alors l'algorithme s'arrête en retournant un plan-solution P^* de coût optimal et de nombre de niveaux minimum, sous l'hypothèse que tous les coûts soient des entiers positifs. Par contre, comme pour tous les planificateurs de ce type (par exemple SATPLAN [Kautz et Walser, 1999 ; Kautz *et al.*, 2006], GP-CSP [Do et Kambhampati, 2000] ou MAXPLAN [Xing *et al.*, 2006]) le problème de l'arrêt lorsqu'il n'y a pas de solution n'est pas résolu.

6.1.2 Amélioration de la borne supérieure au nombre de niveaux

L'efficacité de GP-WCSP* dépend, pour une grande partie, d'une bonne évaluation de la borne $NivMax$. Nous avons montré comment $NivMax$ peut être calculée à partir du premier plan-solution obtenu et du minimum des coûts des actions susceptibles de mener aux buts. Nous

Algorithme 7 Résolution optimale d'un problème de planification valué $\Pi\langle A, I, B\rangle$

Fonction :

- *constructionGraphe* : construit le graphe jusqu'à l'obtention de tous les buts sans mutex entre eux dans le niveau courant. Si le graphe est stabilisé avant de l'obtenir, alors il retourne un échec.
- $constructionGrapheNiveauSuivant(G_k, A)$: retourne le graphe de $k+1$ niveaux contenant uniquement des actions de A.
- $codageWCSP(G)$: retourne le WCSP correspondant au codage du graphe G réduit.
- $resolutionWCSP(wcsp, C)$: retourne une solution optimale au WCSP et son coût. Si ce coût n'est pas strictement inférieur à C alors il retourne un échec.
- $calculNivMax(C, A)$: met-à-jour l'ensemble des actions A et calcule $NivMax$.

Algorithme GP-WCSP* :

// initialisation, recherche d'une première solution
$G_k \leftarrow constructionGraphe(A, I, B)$
si la construction du graphe G_k est un échec **alors**
  echec // problème sans solution
finsi
$wcsp \leftarrow codageWCSP(G_k)$
$(P_k^*, C_k^*) \leftarrow resolutionWCSP(wcsp, \infty)$
tantque $P_k^* = echec$ **faire**
  $k \leftarrow k+1$
  $G_k \leftarrow constructionGrapheNiveauSuivant(G_{k-1}, A)$
  $wcsp \leftarrow codageWCSP(G_k)$
  $(P_k^*, C_k^*) \leftarrow resolutionWCSP(wcsp, \infty)$
fin tantque
$NivMax, A \leftarrow calculNiveauMax(C_k^*, A)$

// boucle anytime
tantque $k < NivMax$ **faire**
  $k \leftarrow k+1$
  $G_k \leftarrow constructionGrapheNiveauSuivant(G_{k-1}, A)$
  $wcsp \leftarrow codageWCSP(G_k)$
  $(P_k^*, C_k^*) \leftarrow resolutionWCSP(wcsp, C_{k-1}^*)$
  si $P_k^* = echec$ **alors**
    $P_k^*, C_k^* \leftarrow P_{k-1}^*, C_{k-1}^*$
  sinon
    $NivMax, A \leftarrow calculNiveauMax(C_k^*, A)$
  finsi
fin tantque
return (P_k^*, C_k^*).

montrerons dans cette sous-section que cette borne peut être largement améliorée, dès l'obtention d'une première solution puis au cours de la recherche, en utilisant les ensembles indispensables et optimalement indispensables qui ont été présentés dans le chapitre précédent.

La valeur de $NivMax$ peut être améliorée en estimant de manière plus précise le coût des actions réellement utilisées dans le plan optimal. En effet, l'équation (6.2) considère l'hypothèse la moins favorable dans laquelle toutes les actions du plan optimal ont un même coût C_{min}. Nous avons, dans le chapitre précédent, défini une action indispensable comme une action présente dans tous les plans-solutions (cf. définition (5.2.6), page 84). Ces actions peuvent être déterminées en utilisant une méthode qui est de complexité temporelle d'ordre polynomiale. Nous noterons $\mathcal{Z}$ l'ensemble des actions indispensables tel qu'une action i-indispensable soit représentée dans $\mathcal{Z}$ par i copies. En remplaçant le coût de chaque action indispensable, qui était évalué à C_{min} dans l'équation (6.2), par son coût réel, nous pouvons améliorer l'estimation de $NivMax$:

$$NivMax = |\mathcal{Z}| + \left\lceil \frac{C_k^* - \sum_{a \in \mathcal{Z}} cout(a)}{C_{min}} \right\rceil - 1 \tag{6.3}$$

Nous avons également défini, dans le chapitre précédent, un ensemble indispensable d'actions comme un ensemble d'actions tel que tous les plans-solutions contiennent au moins une action de cet ensemble (cf. définition (5.2.9), page 85). Un ensemble d'ensembles indispensables d'actions qui n'ont aucune action en commun peut être utilisé pour améliorer le calcul de la borne $NivMax$. Cependant le théorème (5.2.23) a montré que le problème d'un choix optimal d'ensembles indispensables à utiliser est NP-difficile. Nous construisons donc un ensemble d'ensembles indispensables $\mathcal{X}$ de façon gourmande :

1. Nous générons tous les ensembles indispensables d'actions à partir des lemmes 5.2.7, 5.2.10, 5.2.11, 5.2.12, 5.2.14, 5.2.15, 5.2.17 et 5.2.19. Le nombre d'ensembles ainsi construits est au maximum de $n_a + n_f + n_l$ où n_a est le nombre d'actions, n_f est le nombre de fluent, n_l est le nombre de niveaux du graphe de planification relaxé et stabilisé du problème.
2. En commençant par les ensembles composés uniquement d'une action indispensable, puis en ordonnant les ensembles restants selon l'ordre inverse de leur coût minimum, nous ajoutons un ensemble indispensable X à $\mathcal{X}$ ssi X n'a pas d'intersection avec les ensembles déjà ajoutés à $\mathcal{X}$: $\forall\, Y \in \mathcal{X},\, X \cap Y = \emptyset$.

Pour rappel, nous notons $mincout(X)$ le minimum des coûts d'un ensemble d'actions X :

$$mincout(X) = \min_{a \in X} cout(a)$$

La valeur de $NivMax$ peut ainsi être améliorée en remplaçant le coût d'une action appartenant à un ensemble indispensable $X \in \mathcal{X}$, qui était évalué à C_{min} dans l'équation (6.3), par le minimum des coûts des actions de X :

$$NivMax = |\mathcal{X}| + \left\lceil \frac{C_k^* - \sum_{X \in \mathcal{X}} mincout(X)}{C_{min}} \right\rceil - 1 \tag{6.4}$$

La valeur de la borne $NivMax$ obtenue avec un choix optimal d'ensembles indispensables peut parfois être encore améliorée. Considérons, dans l'exemple 4.1, page 55, deux ensembles indispensables qui se recouvrent : $X_1 = \{trajet_{FB}, trajet_{CB}, trajet_{EB}\}$ et $X_2 = \{trajet_{FB}, prendre_F, trajet_{FC}\}$. Le choix optimal d'un ensemble d'ensembles indispensables est $\mathcal{X} = \{X_1\}$, avec

$$\Sigma_{X \in \mathcal{X}}(mincout(X) - C_{min}) = mincout(X_1) - 3 = 12$$

Cependant, il est facile de prouver par une recherche exhaustive que l'ensemble des actions S qui contient au moins une action de chaque élement de X_1 et X_2 satisfait

$$\Sigma_{a \in S}(cout(a) - C_{min}) \geq 14$$

Nous présentons maintenant le problème qui consiste à trouver la meilleure borne $NivMax$ en considérant les actions indispensables comme des ensembles indispensables formés de singletons.

Problème : COEA (Choix Optimal d'Ensemble d'Actions)

Instance : Un problème de planification optimal $\Pi = \langle A, I, B \rangle$ avec C_{min} le coût minimum des actions de A, un ensemble J d'ensembles indispensables et une constante M.

Question : Existe-t-il un sous-ensemble $S \subseteq A$ tel que $\forall X \in J$, $S \cap X \neq \emptyset$ et

$$\sum_{a \in S}(cout(a) - C_{min}) \leq M$$

Si S est un choix optimal d'ensembles d'actions, i.e. un ensemble qui minimise

$$M_S = \Sigma_{a \in S}(cout(a) - C_{min})$$

alors nous pouvons en déduire une borne supérieure

$$NivMax = \left\lceil \frac{C_k^* - M_S}{C_{min}} \right\rceil - 1$$

Sans surprise, COEA est intraitable comme nous allons le prouver.

Théorème 6.1.2 *COEA est NP-Difficile.*

Preuve : Soit $\phi : \{0,1\}^2 \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ la fonction

$$\phi(x,y) \ = \ \begin{cases} \infty & \text{si } x = y = 1 \\ 0 & \text{sinon} \end{cases}$$

et $\psi : \{0,1\} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ la fonction $\psi(0) = 1$, $\psi(1) = 0$.

Nous avons vu dans la preuve du théorème (5.2.23) que WCSP(Γ) est l'ensemble des instances du problème de satisfaction de contraintes pondérées dans lesquelles toutes les fonctions de coût appartiennent à Γ et que WCSP($\{\phi, \psi\}$) est NP-difficile.

Soit P une instance de WCSP($\{\phi, \psi\}$) sur les variables $V = \{v_1, \ldots, v_n\}$, représentée par un graphe $G_P = \langle V, E_P \rangle$, un sous-ensemble $V_P \subseteq V$ et une constante K. La question est de déterminer s'il existe une affectation $s : V \rightarrow \{0,1\}$ telle que

$$\sum_{\{v_i, v_j\} \in E_P} \phi(s(v_i), s(v_j)) + \sum_{v_i \in V_P} \psi(s(v_i)) \leq K$$

Comme dans la preuve du théorème (5.2.23), nous supposons que $V_P = V = \{v_1, \ldots, v_n\}$. Pour démontrer le théorème, il suffit de donner une réduction polynomiale de WCSP($\{\phi, \psi\}$) vers COEA. Pour chaque arc $\{v_i, v_j\} \in E_P$, on crée un fluent $B_{\{i,j\}}$; pour chaque variable v_i, on

crée une action a_i de coût 2 telle que $add(a_i) = \{B_{\{i,j\}} : \{v_i, v_j\} \in E_P\}$. On crée également une action a_0 de coût 1. Soit $A = \{a_0, a_1, \ldots, a_n\}$, soit $J = \{\{a_i, a_j\} : \{v_i, v_j\} \in E_P\}$ et

$$B = \bigwedge_{\{v_i,v_j\}\in E_P} B_{\{i,j\}}$$

Soit Π un problème de planification optimal $\langle A, \emptyset, B\rangle$. Par le lemme 5.2.10, les ensembles indispensables de Π sont les élements de J car un fluent du but $B_{\{i,j\}}$ ne peut être ajouté que par a_i ou a_j.

Il existe une bijection entre les affectations $s : V \to \{0, 1\}$ telles que $\forall\{v_i, v_j\} \in E_P$, $\phi(s(v_i), s(v_j)) = 0$ et les choix $S = \{a_i : s(v_i) = 0, 1 \leq i \leq n\}$ d'ensembles d'actions tels que $\forall\{a_i, a_j\} \in J$, $\{a_i, a_j\} \cap S \neq \emptyset$.

Soit $M = K$. Comme $C_{min} = 1$ et $\forall i \in \{1, \ldots, n\}, cout(a_i) = 2$, nous avons

$$M_S \;=\; \sum_{a\in S}(cout(a) - C_{min}) \;=\; |S|$$

Comme $a_i \in S \Leftrightarrow s(v_i) = 0 \Leftrightarrow \psi(s(v_i)) = 1$, nous avons

$$\sum_{v_i\in V_P} \psi(s(v_i)) \;=\; |S|$$

Donc $M_S \leq M$ si et seulement si

$$\sum_{\{v_i,v_j\}\in E_P} \phi(s(v_i), s(v_j)) + \sum_{v_i\in V_P} \psi(s(v_i))$$
$$= \sum_{v_i\in V_P} \psi(s(v_i)) \leq\; M = K$$

Donc, les solutions S à l'instance de COEA définie par Π correspondent exactement aux solutions s du problème de satisfaction de contraintes pondérées P. Cette réduction est clairement polynomiale. ∎

Nous avons également défini précédemment des ensembles optimalement indispensables d'actions, c'est-à-dire des ensembles indispensables à l'obtention d'une solution de coût optimal (cf. sous-section 5.2.3, page 90). Ils peuvent être utilisés pour obtenir une estimation de $NivMax$:

- Dans l'équation (6.3) en construisant $\mathcal{Z}$ avec les actions optimalement indispensables à la place des actions indispensables.
- Dans l'équation (6.4) en construisant $\mathcal{X}$ à partir d'ensembles optimalement indispensables et en respectant les mêmes conditions que celles données pour les ensembles indispensables d'actions.

6.1.3 Exemple de résolution optimale d'un problème

Considérons l'exemple 4.1, page 55. Pour rappel, une phase de prétraitement détermine que l'action de plus petit coût qui est potentiellement utile à l'obtention du but est $poser_B$ de coût 3. Nous calculons également l'ensemble des actions indispensables $\mathcal{Z} = \{prendre_F, poser_B\}$ et un ensemble d'ensembles indispensables d'actions $\mathcal{X} = \{\{prendre_F\}, \{poser_B\}, \{trajet_{FB}, trajet_{BF}, trajet_{DE}, trajet_{ED}, trajet_{BC}, trajet_{CB}\}\}$. Le premier plan-solution trouvé P_3^* a un coût $C_3^* = 108$. Puis, au cours de la recherche, différentes actions trop onéreuses sont détectées et modifient les ensembles d'ensembles optimalement indispensables :

1. Jusqu'au niveau 3, les actions optimalement indispensables sont les actions indispensables :
   - $\mathcal{Z} = \{prendre_F, poser_B\}$,
   - $\mathcal{X} = \{\{prendre_F\}, \{poser_B\}, \{trajet_{FB}\}, \{trajet_{BF}, trajet_{DE}, trajet_{ED}, trajet_{BC}, trajet_{CB}\}\}$
2. A partir du niveau 4, l'action $trajet_{FB}$ est trop onéreuse et $trajet_{FC}$ est détectée comme une action optimalement indispensable :
   - $\mathcal{Z} = \{prendre_F, poser_B, trajet_{FC}\}$,
   - $\mathcal{X} = \{\{prendre_F\}, \{poser_B\}, \{trajet_{FC}\}, \{trajet_{BF}, trajet_{DE}, trajet_{ED}, trajet_{BC}, trajet_{CB}\}\}$
3. A partir du niveau 5, l'action $trajet_{DE}$ est trop onéreuse et $trajet_{BC}$ est détectée comme une action optimalement indispensable :
   - $\mathcal{Z} = \{prendre_F, poser_B, trajet_{FC}, trajet_{BC}\}$,
   - $\mathcal{X} = \{\{prendre_F\}, \{poser_B\}, \{trajet_{FC}\}, \{trajet_{BC}\}\}$

Le nombre maximum de niveaux à construire pour garantir l'obtention d'un plan-solution optimal, selon les différentes équations et la prise en compte des ensembles indispensables ou optimalement indispensables, est présenté dans le tableau 6.1. Comme aucune valeur de $NivMax$ ne permet au niveau 3 de garantir l'obtention d'une solution optimale, on construit le niveau suivant du graphe et on en extrait $P_4^* = \langle prendre_A, trajet_{AC}, trajet_{CB}, poser_B \rangle$ de coût $C_4^* = 58$. La valeur $NivMax = 3$ trouvée au 4ème niveau, nous permet d'arrêter la recherche à ce niveau. En effet, dès que $NivMax \leq k$, nous pouvons déduire que P_k^* est un plan optimal. La valeur $NivMax = 3$ signifie que si un plan strictement meilleur que P_4^* existe alors il n'aurait pas plus de 3 niveaux. L'algorithme se termine donc après le développement du graphe de niveau 4 en garantissant que P_4^* est un plan de coût optimal.

équation	C_3^*	C_4^*		C_5^*	
	indis.	indis.	optimalement indis.	indis.	optimalement indis.
(6.2)	35	20	20	20	20
(6.3)	35	18	12	18	3
(6.4)	26	9	3	9	3

TAB. 6.1 – Valeurs de $NivMax$, dans l'exemple 4.1, calculées selon les équations (6.2)-(6.4) et les coûts des plans-solutions obtenus.**Indis.** : utilisation des ensembles indispensables. **Optimalement indis.** : utilisation des ensembles optimalement indispensables.

Cet exemple met en évidence l'intérêt de continuer la recherche dans les niveaux suivants du graphe après l'apparition d'une première solution. L'utilisation d'une borne majorante du

coût du plan-solution et une analyse de l'importance des actions indispensable et des ensembles indispensables permettent d'élaguer efficacement la recherche, tout en garantissant l'obtention d'un plan-solution optimal.

6.1.4 Résultats expérimentaux de GP-WCSP*

Les résultats expérimentaux de GP-WCSP* sont donnés dans le tableau 6.2 :

- La valeur initiale de $NivMax$ est calculée en utilisant la première équation (6.2) dans laquelle le coût de toutes les actions est estimé à C_{min} et la valeur finale est estimée par l'équation (6.4) après détection des ensembles optimalement indispensables. La valeur de $NivMax$ est calculée à chaque niveau et peut être diminuée d'un niveau à l'autre. Nous donnons la valeur finale qui correspond à la plus petite valeur obtenue.
- N_{ai} est le nombre d'actions optimalement indispensables détectées.
- $|\mathcal{X}|$ est le nombre d'ensembles optimalement indispensables d'actions qui n'ont pas d'actions en commun, construit selon la méthode introduite dans la sous-section 6.1.2.
- N_{to} est le nombre d'actions trop onéreuses détectées.
- Les solutions obtenues sont décrites par le meilleur plan-solution obtenu : son optimalité ($plan$), son nombre de niveaux (k) et son nombre d'actions (N_a). Le temps de résolution est donné avec la notation $h : m : s$ qui représente respectivement les heures, les minutes et les secondes, sinon le temps est exprimé en secondes. Si le meilleur plan trouvé est P^* alors nous avons une garantie d'optimalité. Si le meilleur plan trouvé est P_k^* alors nous n'avons pas de garantie d'optimalité globale mais le plan obtenu est un plan de coût optimal parmi les plans-solutions contenant au maximum k niveaux.

Comme nous l'avions observé dans le chapitre 4, les WCSP résolus ont une taille très largement supérieure à ceux récemment résolus dans la littérature [Larrosa et Schiex, 2003 ; de Givry *et al.*, 2005]. Nous pouvons ainsi résoudre des WCSP provenant de graphes comportants jusqu'à 20 niveaux. L'usage du coût du plan-solution obtenu au niveau $k-1$ diminue fortement le temps de résolution du WCSP issu du graphe de niveau k. Par exemple, le WCSP issu du graphe de planification réduit de 20 niveaux dans *blocks*05 a une taille d'espace de recherche de $5E + 566$ (moyenne géométrique de taille de domaine : 2,79 ; variables : 1273 ; contraintes : 19005). Il est résolu en 0,58 secondes en passant en paramètre le coût du plan déjà obtenu et en 12885 secondes sans utiliser ce dernier. L'utilisation des WCSP pour résoudre de manière optimale des problèmes de planification avec actions valuées est donc envisageable.

GP-WCSP* s'avère cependant plus performant dans certains domaines que dans d'autres. Dans le domaine *storage* qui contient peu d'actions, il détermine rapidement une solution optimale. Les domaines qui contiennent beaucoup d'actions indispensables, telles que *blocks* ou *logistic*, représentent le cas le plus favorable même si le dernier WCSP est souvent trop gros pour être résolu en moins de 100000 secondes, et que, par conséquent, aucune garantie d'optimalité ne peut être fournie. Dans les domaines *driver*, *zenotravel*, *satellite* et *depots*, GP-WCSP* retourne un plan optimal mais la valeur de $NivMax$ est rarement suffisamment faible pour être atteinte. Pour les benchmarks testés, les plans-solutions optimaux correspondent souvent au premier plan-solution trouvé par GP-WCSP*.

Problèmes	NivMax					Solution			Temps de
	init.	final	N_{ai}	$\|\mathcal{X}\|$	N_{to}	plan	k	N_a	résolution
blocks01	51	5	6	6	0	P^*	6	6	11,17
blocks02	61	31	6	6	10	P^*_{20}	10	10	0 : 10 : 12
blocks03	37	5	6	6	0	P^*	6	6	3,15
blocks04	96	50	8	8	0	P^*_{20}	12	12	0 : 47 : 36
blocks05	93	21	7	7	37	P^*_{20}	10	10	1 : 21 : 09
blocks06	111	73	9	9	0	P^*_{20}	16	16	0 : 25 : 19
blocks07	84	26	11	11	34	P^*_{20}	12	12	2 : 57 : 45
blocks08	132	9	10	10	0	P^*	10	10	0 : 19 : 44
blocks09	139	88	11	11	0	P^*_{20}	20	20	0 : 51 : 54
blocks10	145	71	13	13	0	P^*_{20}	20	20	1 : 23 : 08
logistics01	148	21	19	19	27	P^*_{13}	9	20	0 : 56 : 52
logistics02	138	27	17	17	13	P^*_{12}	9	19	0 : 59 : 01
logistics03	145	34	13	13	5	P^*_{15}	9	15	2 : 26 : 05
logistics04	234	46	25	25	0	P^*_{12}	9	27	2 : 23 : 47
logistics05	128	21	15	15	26	P^*_{12}	9	17	2 : 03 : 15
logistics06	51	7	8	8	70	P^*	3	8	0,92
storage01	15	2	3	3	0	P^*	3	3	0,01
storage02	38	2	1	3	0	P^*	3	3	0,03
storage03	32	4	1	3	53	P^*	3	3	1,54
driverlog01	60	41	0	3	0	P^*_{16}	6	8	18 : 54 : 12
driverlog02	141	121	0	9	0	P^*_{12}	7	18	3 : 21 : 26
driverlog03	110	86	0	6	0	P^*_{13}	7	12	3 : 55 : 33
driverlog04	179	112	0	11	0	P^*_{11}	10	21	2 : 48 : 22
driverlog07	146	88	0	6	0	P^*_9	8	18	4 : 45 : 11
driverlog08	224	165	0	14	0	P^*_9	9	27	6 : 41 : 08
zenotravel01	18	6	0	1	0	P^*	3	3	4,93
zenotravel02	52	30	2	3	11	P^*_{17}	6	7	6 : 32 : 49
zenotravel03	49	26	0	4	0	P^*_{11}	6	7	6 : 55 : 47
satellite01	60	23	5	8	22	P^*_{20}	8	9	1 : 18 : 41
satellite02	80	28	4	13	17	P^*_{20}	15	16	1 : 36 : 32
satellite03	128	61	0	4	0	P^*_{16}	11	14	2 : 03 : 20
satellite04	124	89	0	6	0	P^*_{13}	11	24	3 : 08 : 22
depot01	93	33	4	8	0	P^*_{16}	8	10	11 : 06 : 02
depot02	123	73	5	6	0	P^*_{13}	11	17	9 : 07 : 34

TAB. 6.2 – Résultats expérimentaux de GP-WCSP*.

Dans tous les problèmes testés, la valeur de $NivMax$ est diminuée de manière significative et en moyenne de 61% grâce à l'utilisation des ensembles indispensables trouvés dans tous les problèmes testés. Les actions indispensables et optimalement indispensables jouent donc un rôle déterminant dans la réduction de $NivMax$. En effet, en moyenne 48% des actions du meilleur plan-solution trouvé sont des actions optimalement indispensables qui sont détectées rapidement par une analyse d'un graphe de planification relaxé. Par exemple, dans le problème *logistics*04, nous détectons 25 actions indispensables pour un plan-solution P^*_{12} comportant 27 actions.

Lorsqu'il existe des actions trop onéreuses, celles-ci permettent de diminuer très efficacement l'espace de recherche. Par exemple, durant la résolution du problème *logistics*06, 70 actions trop onéreuses sont détectées. Le temps total nécessaire à la détection des ensembles indispensables d'actions et des actions trop onéreuses, durant la résolution des 33 problèmes testés dans le tableau 6.2, est en moyenne de 0,06 secondes (et jamais plus de 0,25 secondes). Ce très faible temps de calcul permet d'envisager l'utilisation de ces méthodes (détection des ensembles indispensables et des actions trop onéreuses) dans d'autres algorithmes pour la planification de coût optimal.

6.2 Comparaison avec des planificateurs de coût non optimal

Nous comparons, dans cette section, notre approche avec deux planificateurs numériques qui sont non-optimaux, c'est-à-dire qu'ils ne garantissent pas d'optimalité. Pour cela, nous utilisons deux versions de notre planificateur :

- GP-WCSP retourne le premier plan-solution trouvé qui est de coût optimal pour un nombre de niveaux minimum,
- et GP-WCSP* retourne un plan-solution de coût optimal ou un plan-solution optimal à un niveau donné (comme décrit dans la colonne "plan-solution" du tableau (6.2)).

LPG [Gerevini *et al.*, 2004] est un planificateur numérique basé sur une méthode de recherche locale. Nous l'avons testé sous deux versions :

- LPG-SPEED retourne la première solution trouvée
- LPG-QUALITY cherche à optimiser la qualité de la solution.

Comme l'heuristique de LPG est en partie aléatoire, nous avons relancé les tests sur la durée de résolution de notre planificateur. Nous donnons la moyenne des coûts des plans-solutions obtenus par LPG-SPEED et le meilleur plan-solution obtenu par LPG-QUALITY. SGPLAN [Chen *et al.*, 2004] utilise les landmarks qu'il ordonne pour diviser le problème en sous-problèmes. Il retourne le même plan-solution quel que soit le nombre d'itérations.

En moyenne, LPG et SGPLAN résolvent chaque problème testé en moins d'une seconde. Les figures 6.1 et 6.2 donnent les résultats obtenus pour la comparaison des coûts des plans-solutions retournés par ces planificateurs. Elles montrent que les plans retournés par LPG-SPEED et SGPLAN sont rarement optimaux. LPG-SPEED, même relancé un grand nombre de fois (plusieurs centaines), ne trouve pas la solution optimale pour un tiers des problèmes. Pour un même problème, les coûts des plans-solutions qu'il retourne sont très variables (du simple au triple). SGPLAN retourne rarement un plan de coût optimal et surévalue ce coût de 24% en moyenne (jusqu'à 89% pour *block*09). Bien que LPG-QUALITY retourne souvent un plan de coût optimal, aucune garantie d'optimalité n'est apportée. Pour certains problèmes (*blocks*07, *driver*02

et *driver*03), ni SGPLAN ni LPG-QUALITY ne trouvent une solution aussi bonne que GP-WCSP*. Dans les 33 problèmes testés, la solution retournée par GP-WCSP* est au moins aussi bonne (et souvent meilleure) que les plans retournés par LPG ou SGPLAN. C'est le cas même lorsque GP-WCSP* n'a pas pu fournir, faute de temps, une garantie complète et qu'il ne fournit une garantie d'optimalité que pour un nombre de niveaux k.

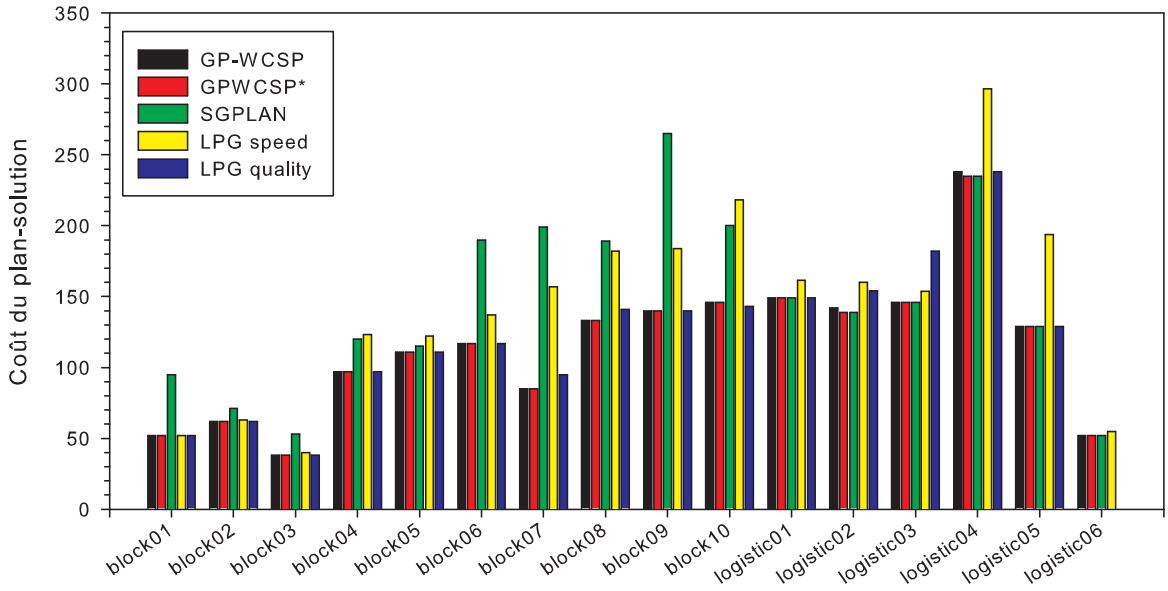


FIG. 6.1 – Comparaison du coût des plans-solutions retournés par GP-WCSP, GP-WCSP*, SGPLAN, LPG-SPEED et LPG-QUALITY dans les domaines *Block* et *Logistic*.

Les plans retournés par GP-WCSP ont un surcoût moyen de 5,7 % seulement par rapport au coût d'un plan optimal. Comme dans beaucoup de problèmes d'optimisation, l'obtention d'une preuve de l'optimalité est bien plus longue que l'obtention d'une bonne solution. Nous avons néanmoins démontré que notre approche peut produire une preuve de l'optimalité pour certains problèmes non triviaux de planification optimale. Les planificateurs non-optimaux, d'autre part, sont essentiels pour des problèmes beaucoup plus grands pour lesquels une recherche exhaustive n'est pas possible en pratique.

6.3 Comparaison avec des planificateurs de coût optimal

Nous comparons, dans cette section, GP-WCSP* à deux planificateurs optimaux qui sont deux versions de HSP* [Haslum, 2006] qui effectue une recherche heuristique dans les espaces d'états. HSP^*_0 (également connu sous le nom de TP4) effectue une recherche par régression, en utilisant l'algorithme IDA*, tandis que HSP^*_a effectue une recherche par régression en utilisant l'algorithme IDAO*. Ces deux planificateurs garantissent l'optimalité des plans-solutions retournés. Ils ne sont

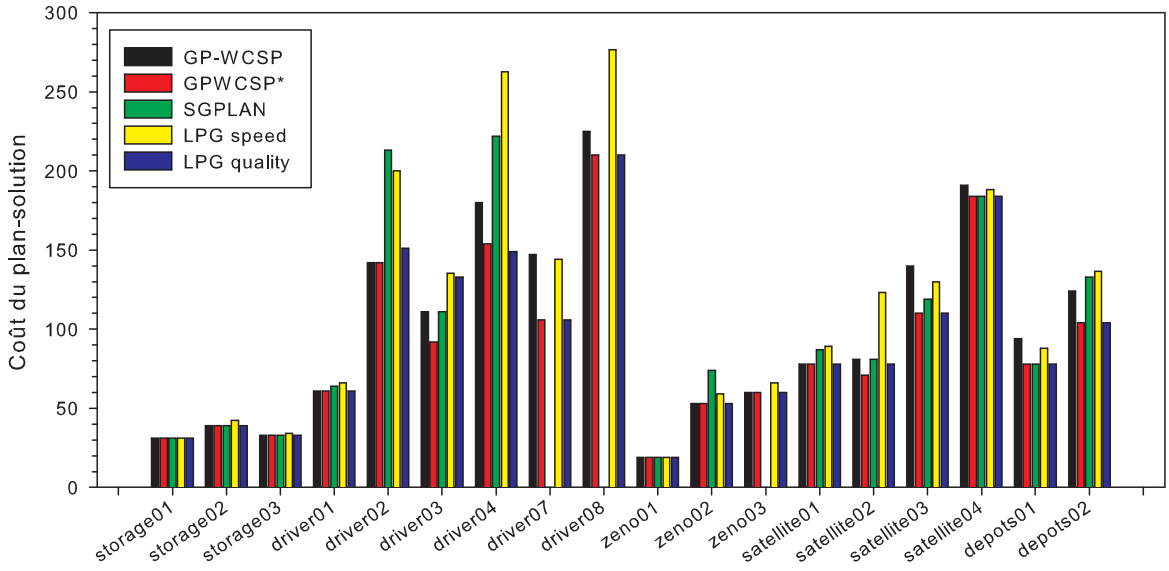


FIG. 6.2 – Comparaison du coût des plans-solutions retournés par GP-WCSP, GP-WCSP*, SGPLAN, LPG-SPEED et LPG-QUALITY dans les domaines *Storage*, *Driverlog*, *Zeno*, *Satellite* et *Depot*.

pas spécifiquement conçus pour avoir un comportement anytime : ils retournent un plan-solution optimal ou aucun plan-solution en cas d'arrêt du programme.

Le tableau 6.3 donne les résultats de nos expériences sur les mêmes problèmes que dans le tableau 6.2, excepté le domaine *storage*, car HSP* n'accepte pas les problèmes issus de ce domaine. Un tiret signifie que le programme a été arrêté après 100000 secondes et qu'aucune solution n'est retournée. Pour chaque problème, les plans retournés par les différents planificateurs étaient de coût identique excepté trois problèmes pour lesquels GP-WCSP* n'a pas trouvé le plan optimal. Dans ce cas, les solutions avaient des coûts surévalués de 2% pour *blocks*10 et *driver*02, et de 8% pour *driver*04. Comme indiqué précédemment dans le tableau 6.2, pour beaucoup de problèmes, GP-WCSP* n'a pas pu fournir une garantie complète de l'optimalité à cause du temps excessif de calcul, mais dans ce cas il fournit une garantie d'optimalité pour un nombre de niveaux k. Pour tous les problèmes résolus par GP-WCSP*, une version de HSP* a résolu le problème en garantissant l'optimalité (excepté le problème *driver*08). De plus, GP-WCSP* était, en moyenne, considérablement plus lent que les autres planificateurs.

6.4 Synthèse

Dans ce chapitre, nous avons présenté une nouvelle méthode pour obtenir une solution de coût optimal. Le planificateur GP-WCSP* implémente l'algorithme anytime que nous avons développé et qui est basé sur celui de GP-WCSP. Une fois la phase d'expansion du graphe de planification terminée, le graphe est codé en un WCSP équivalent puis résolu. Le coût du plan-solution k-

Problem	GP-WCSP*	HSP^*_0	HSP^*_a
blocks01	11,17	**0,01**	**0,01**
blocks02$^+$	0 : 10 : 12	**0,01**	**0,01**
blocks03	3,15	**0,01**	**0,01**
blocks04$^+$	0 : 47 : 36	**0,03**	0,08
blocks05$^+$	1 : 21 : 09	**0,06**	0,16
blocks06$^+$	0 : 25 : 19	**0,03**	0,35
blocks07$^+$	2 : 57 : 45	**0,15**	1,17
blocks08	0 : 19 : 44	**0,30**	1,13
blocks09$^+$	0 : 51 : 54	**0,37**	3,72
blocks10$^+$	1 : 23 : 08	**0,31**	0 : 00 : 42
logistics01$^+$	0 : 56 : 52	0 : 01 : 51	**0,87**
logistics02$^+$	0 : 59 : 01	0 : 07 : 11	**1,46**
logistics03$^+$	2 : 26 : 05	0,19	**0,06**
logistics04$^+$	2 : 23 : 47	-	**0 : 00 : 50**
logistics05$^+$	2 : 03 : 15	0 : 01 : 02	**0,91**
logistics06	0,92	0,06	**0,04**
driverlog01$^+$	18 : 54 : 12	**0,02**	0,03
driverlog02$^+$	3 : 21 : 26	11 : 58 : 24	**0 : 00 : 24**
driverlog03$^+$	3 : 55 : 33	7,10	**1,75**
driverlog04$^+$	2 : 48 : 22	-	**0 : 04 : 32**
driverlog07$^+$	4 : 45 : 11	-	**0 : 06 : 06**
driverlog08$^+$	**6 : 41 : 08**	-	-
zenotravel01	4,93	**0,01**	**0,01**
zenotravel02$^+$	6 : 32 : 49	0,04	**0,03**
zenotravel03$^+$	6 : 55 : 47	1,43	**1,39**
satellite01$^+$	1 : 18 : 41	**0,01**	**0,01**
satellite02$^+$	1 : 36 : 32	0,80	**0,14**
satellite03$^+$	2 : 03 : 20	0 : 39 : 54	**1,47**
satellite04$^+$	3 : 08 : 22	-	**0 : 02 : 29**
depot01$^+$	11 : 32 : 33	**0,05**	0,09
depot02$^+$	9 : 07 : 34	**0 : 00 : 57**	-

TAB. 6.3 – Comparaison des temps de résolution retournés par GP-WCSP*, HSP^*_0 et HSP^*_a. Pour les problèmes marqués d'un "+", GP-WCSP* fournit seulement une garantie d'optimalité pour k niveaux, la valeur de k étant donné dans la colonne "plan" du tableau 6.2. Le meilleur temps est écrit en gras.

optimal trouvé à un niveau k du graphe nous sert ensuite de majorant pour relancer la recherche d'une meilleure solution au niveau suivant.

Le nombre de niveaux à construire pour garantir l'obtention d'une solution de coût optimal est fini mais inconnu au départ. Nous avons montré comment une borne supérieure à ce nombre (appelée $NivMax$) peut être calculée à partir du premier plan-solution obtenu. L'utilisation des actions et des ensembles optimalement indispensables diminue de manière significative (61% en moyenne) la valeur de $NivMax$ dans tous les problèmes testés. Les actions trop onéreuses permettent également de diminuer la taille de l'espace de recherche.

La comparaison expérimentale avec d'autres planificateurs montre que l'utilisation du graphe de planification et des WCSP pour la planification optimale est possible en pratique même si elle n'est pas compétitive, en terme de temps de calcul, avec les planificateurs optimaux les plus récents.

Chapitre 7

Conclusion et Perspectives

Nous présentons dans ce chapitre une synthèse des contributions apportées par cette thèse puis nous proposerons plusieurs perspectives de recherche.

7.1 Synthèse

Dans cette thèse, nous nous sommes plaçés dans un cadre de planification classique non temporel avec des actions valuées. Notre objectif était de développer une méthode originale permettant d'obtenir une solution de coût optimal et pour y parvenir nous avons procédé en trois étapes :

- Nous avons commencé par développer une méthode permettant d'obtenir un plan-solution de coût optimal pour un nombre de niveaux donné.
- Nous avons ensuite proposé différentes méthodes d'extraction d'informations à partir d'une analyse de domaines et nous avons montré comment ces informations pouvaient améliorer la résolution optimale des problèmes de planification.
- Enfin, nous avons développé une méthode permettant d'obtenir une solution de coût optimal sans fixer de borne au nombre de niveaux des plans.

Nous récapitulons pour chacune de ces étapes, l'essentiel de nos contributions et ce que l'on peut en retenir.

7.1.1 Obtention d'une solution de coût optimal pour un nombre de niveaux donné

Pour obtenir une solution de coût optimal pour un nombre de niveaux donné, nous avons développé une méthode qui utilise le graphe de planification introduit par le planificateur GRAPHPLAN. Cette méthode consiste à coder un graphe de planification en un WCSP puis à extraire de ce WCSP une solution de coût optimal.

Le codage original que nous avons proposé représente chaque fluent du graphe par une variable dont le domaine est l'ensemble des actions qui ajoutent ce fluent dans le graphe. Les contraintes sont construites à partir des relations entre les fluents et les actions et à partir des contraintes d'exclusions mutuelles contenues dans le graphe. Elles relient donc uniquement des valeurs appartenant à un même niveau ou à un niveau adjacent. Ce codage a permis de générer des WCSP comportant des domaines de petite taille, de nombreuses contraintes, et qui sont

fortement structurés.

Nous avons ensuite implémenté cette méthode dans le planificateur GP-WCSP et nous avons comparé l'efficacité de plusieurs algorithmes de résolution sur les WCSP issus de problèmes de planification. Les expérimentations ont montré que l'algorithme FDAC était le plus efficace et qu'il permettait de résoudre des WCSP dont la taille était largement supérieure aux WCSP aléatoires actuellement résolus.

Nous pouvons retenir de cette première étape que le codage du graphe de planification donne des WCSP fortement structurés ce qui rend possible la résolution de WCSP de taille largement supérieure à celle que nous aurions pensé pouvoir résoudre. Notre méthode permet donc en pratique d'obtenir une solution de coût optimal pour un nombre de niveaux donné.

7.1.2 Extraction d'informations à partir d'une analyse des domaines et des problèmes

Pour tenter d'améliorer la résolution optimale des problèmes de planification, nous avons ensuite cherché à exploiter les informations contenues dans les domaines et les problèmes. Nous avons donc proposé plusieurs techniques pour extraire des informations à partir d'une analyse des domaines et des problèmes.

Les premières méthodes que nous avons présentées permettent de déterminer, avant de résoudre le problème, des ensembles d'actions, de fluents (ou une combinaison des deux) qui sont indispensables à la résolution du problème. Par exemple, nous pouvons déduire d'une paire indispensable, composée d'un ensemble d'actions X et d'un ensemble de fluents U, que tous les plans-solutions contiennent une action de X ou satisfont un fluent de U. Nous avons également montré comment déterminer, à partir d'un premier plan-solution, des ensembles d'actions, de fluents (ou une combinaison des deux) qui sont indispensables à l'obtention de solutions de coût optimal. Ces méthodes sont basées sur une relaxation du problème, elles nécessitent donc des temps de calculs très faible et les expérimentations ont montré qu'elles permettaient d'obtenir des informations utilisables dans tous les problèmes testés.

A partir du nombre d'occurences des actions et des fluents contenus dans les solutions, nous avons ensuite présenté un ensemble d'inégalités linéaires. Nous avons montré comment elles peuvent permettre, avant de résoudre le problème, d'améliorer la borne inférieure au coût d'une solution optimale et de détecter des problèmes sans solution.

Nous avons également montré comment modifier la répartition des coûts dans le problème en transformant le problème en un problème équivalent en coût. Nous avons défini une opération de transformation des problèmes et montré comment conserver l'équivalence entre le problème initial et le problème transformé. Une meilleure répartition des coûts peut ainsi permettre une meilleure résolution des problèmes dans des approches dépendantes, par exemple, de la répartition des coûts entre les actions .

Nous pouvons retenir de cette deuxième étape que toutes les méthodes que nous avons pré-

sentées sont utilisables en prétraitement et sont indépendantes de la stratégie de résolution des problèmes. Elles peuvent donc être utilisées dans de nombreuses approches comme par exemple les approches heuristiques. Les expérimentations effectuées montrent en particulier que la détermination des paires indispensables est très rapide et qu'elle permet d'obtenir des informations dans tous les problèmes testés.

7.1.3 Obtention d'une solution de coût optimal

Pour obtenir une solution de coût optimal (sans fixer de borne au nombre de niveaux), nous avons développé le planificateur GP-WCSP* qui est une extension de GP-WCSP. GP-WCSP* poursuit le développement du graphe après l'apparition d'une première solution ; le coût d'un plan-solution k-optimal trouvé à un niveau k du graphe sert alors de majorant pour la recherche d'une meilleure solution au niveau suivant.

Nous avons calculé une borne supérieure (appelée $NivMax$) au nombre de niveaux à construire pour garantir l'obtention d'une solution de coût optimal. Puis nous avons montré que, dans tous les problèmes testés, cette borne pouvait être grandement diminuée en utilisant les ensembles d'actions indispensables. Les expérimentations ont montré que GP-WCSP* obtient souvent le plan optimal mais sans pouvoir toujours garantir cette optimalité.

Nous avons ensuite comparé notre approche avec plusieurs versions du planificateur optimal HSP* qui est un planificateur séquentiel qui effectue une recherche heuristique dans les espaces d'états. Cette comparaison montre que, pour la plupart des problèmes, GP-WCSP* nécessite beaucoup plus de temps de calcul que les approches heuristiques. Mais pour un problème comportant beaucoup de parallélisme et de contraintes, GP-WCSP* obtient un plan-solution alors que toutes les versions de HSP* échouent.

Nous pouvons retenir de cette dernière étape que la poursuite de la construction du graphe de planification jusqu'au niveau $NivMax$ n'est pas compétitive en terme de temps de calcul, notamment par rapport à une approche heuristique. Mais, l'expérimentation avec GP-WCSP* démontre cependant que l'utilisation des ensembles d'actions indispensables permet de diminuer fortement la valeur de $NivMax$ dans tous les problèmes testés et ainsi d'en résoudre un plus grand nombre. Cette approche nous confirme donc que les informations que nous avons extraites d'une analyse des domaines et des problèmes s'avèrent efficaces et permettent d'améliorer la résolution du problème.

7.2 Perspectives

Plusieurs directions de recherche méritent encore d'être explorées. En premier lieu, il nous semble possible d'améliorer les techniques utilisées par GP-WCSP et GP-WCSP* :

- Les informations calculées dans un WCSP de niveau $i-1$ sont en grande partie recalculées de manière identique lors de la résolution du WCSP de niveau i. Actuellement la seule information conservée d'un niveau à l'autre est le majorant du coût d'une solution optimale qui permet d'améliorer la résolution des WCSP. Or, un certain nombre d'actions, de fluents et de mutex se retrouvent à la fois dans le graphe réduit de niveau $i-1$ et dans le graphe

réduit de niveau i. Les WCSP issus de ces graphes possèdent donc un bon nombre de similitudes qu'il pourrait être intéressant d'exploiter.

- L'efficacité de GP-WCSP* dépend essentiellement de la valeur de la borne $NivMax$. En effet, lors des expérimentations, le plan optimal est souvent obtenu sans garantie d'optimalité (faute de temps). Plusieurs améliorations peuvent être envisagées :
  1. Les estimations de $NivMax$ sont liées à une solution séquentielle mais utilisées avec une solution parallèle ; une meilleure estimation pourrait être obtenue en tenant compte du parallèlisme de certaines actions.
  2. D'autre part, les différentes estimations de la valeur de $NivMax$ sont basées sur l'action de plus petit coût ; l'estimation du nombre d'occurences de cette action pourrait permettre d'utiliser la deuxième action la moins coûteuse...
  3. Une autre piste de recherche plus générale pourrait consister à rééquilibrer les coûts des actions dans le problème. En effet, nous avons présenté une méthode permettant de transformer un problème de planification en un problème équivalent en coût ainsi qu'une opération de transformation qui préserve les coûts. Il serait donc pertinent d'expérimenter cette méthode afin d'obtenir un problème plus facile à résoudre et qui permettrait d'obtenir une meilleure borne au coût de la solution optimale.

D'autre part, les méthodes que nous avons proposées ont été utilisées pour améliorer l'efficacité d'algorithmes basés sur le graphe de planification. La plupart d'entres elles pourraient certainement être exploitées dans d'autres approches :

- La détection des actions, des fluents et des ensembles indispensables dans un plan-solution a montré de bons résultats et des ensembles optimalement indispensables ont été trouvés dans tous les problèmes testés. Il pourrait certainement être intéressant d'étendre leur utilisation à d'autres approches comme par exemple aux approches heuristiques. D'autres techniques, indépendantes du graphe de planification, peuvent également être étudiées pour trouver d'autres ensembles indispensables. Ainsi, le programme en nombre entier que nous avons introduit pourrait permettre de développer une autre approche pour obtenir une solution de coût optimal.
- Nous avons montré comment transformer un problème de planification en un autre problème mais plus facile à résoudre. Pour cela, nous avons défini une transformation d'un problème en un problème équivalent en coût. Nous avons également présenté une opération de transformation et l'obtention d'un ensemble de transformations tel que le problème équivalent obtenu maximise la borne inférieure du coût de la solution optimale. Cette nouvelle approche est inspirée de méthodes très performantes pour les WCSP et mériterait d'être approfondie.

Enfin, tous nos travaux se restreignent à un cadre dans lequel le coût d'un plan-solution est la somme des coûts de ses actions. De nombreux exemples de métriques additives sont en effet présents dans les applications réelles comme, par exemple, des coûts financiers, des consommations d'énergie, des émissions de CO_2, ou un temps total d'utilisation d'une ressource donnée. Mais d'autres applications nécessitent plus d'une métrique additive. Ce type de métrique pourrait être modélisé dans les WCSP puisqu'ils permettent d'intégrer des contraintes supplémentaires. Par exemple, considérons le cas dans lequel nous souhaitons minimiser une métrique *cout* et limiter

un deuxième critère $cout'$ par une borne K fixée. Cet exemple peut être traité en ajoutant une contrainte globale $cout'(P) \leq K$ pour chaque WCSP généré par GP-WCSP*. L'estimation de la valeur de $NivMax$ devrait pouvoir alors être améliorée en tenant compte de cette deuxième métrique. Il serait donc intéressant de poursuivre ces travaux en intégrant un objectif multi-critère.

Bibliographie

Bacchus F. (2001). The AIPS'00 planning competition. *AI Magazine*, **22**(3), 47–56.

Bacchus F. et Yang Q. (1994). Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, **71**(1), 43–100.

Backstrom C. et Nebel B. (1995). Complexity results for SAS^+ planning. *Computational Intelligence*, **11**, 625–655.

Baioletti M., Marcugini S. et Milani A. (2000). DPPlan : An algorithm for fast solutions extraction from a planning graph. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 13–21.

Baioletti M., Marcugini S., Milani A. et Vanvitelli V. (1998). CSATPlan : a SATPlan-based tool for planning with constraints. Dans *Proceedings of AIPS'98 Workshop on Planning as Combinatorial Search*.

Baptiste P., Laborie P., Pape C. L. et Nuijten W. (2006). *Constraint-Based Scheduling and Planning*, Dans *Handbook of Constraint Programming*, pages 761–799. Elsevier. ISBN : 0-444-52726-5.

Benton J., van den Briel M. et Kambhampati S. (2007a). Finding admissible bounds for over-subscription planning problems. Dans *Proceedings of ICAPS'07 Workshop on Heuristics for Domain independent Planning : Progress, Ideas, Limitations, Challenges*.

Benton J., van den Briel M. et Kambhampati S. (2007b). A hybrid linear programming and relaxed plan heuristic for partial satisfaction planning problems. Dans *Proceedings of 17th International Conference on Automated Planning and Scheduling, (ICAPS'07)*, pages 34–41.

Bienvenu M., Fritz C. et McIlraith S. A. (2006). Planning with qualitative temporal preferences. Dans *Proceedings of 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 134–144.

Bistarelli S., Montanari U. et Rossi F. (1995). Constraint solving over semirings. Dans *Proceedings of 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 624–630.

Bistarelli S., Montanari U. et Rossi F. (1997). Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, **44**(2), 201–236.

Bistarelli S., Montanari U., Rossi F., Schiex T., Verfaillie G. et Fargier H. (1999). Semiring-based CSPs and valued CSPs : Frameworks, properties and comparison. *Constraints*, **4**(3), 199–240.

Blum A. et Furst M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, **90**(1-2), 281–300.

Blum A. et Furst M. L. (1995). Fast planning through planning graph analysis. Dans *Proceedings of 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1636–1642.

Bonet B. et Geffner H. (1998). HSP : Heuristic search planner. Dans *Proceedings of International Planning Competition (IPC'1998)*.

Bonet B. et Geffner H. (1999). Planning as heuristic search : New results. Dans *Proceedings of 5th European Conference on Planning (ECP'99)*, pages 360–372.

Bonet B. et Geffner H. (2001). Planning as heuristic search. *Artificial Intelligence*, **129**(1-2), 5–33.

Bonet B. et Geffner H. (2006). Heuristics for planning with penalties and rewards using compiled knowledge. Dans *Proceedings of 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 452–462.

Bonet B., Loerincs G. et Geffner H. (1997). A robust and fast action selection mechanism for planning. Dans *Proceedings of 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 714–719.

Borowsky B. U. et Edelkamp S. (2008). Optimal metric planning with state sets in automata representation. Dans *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 874–879 : AAAI Press.

Brafman R. I. et Chernyavsky Y. (2005). Planning with goal preferences and constraints. Dans *Proceedings of 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, pages 182–191.

Brafman R. I. et Domshlak C. (2002). Introducing variable importance tradeoffs into CP-Nets. Dans *Proceedings of 18th Conference in Uncertainty in Artificial Intelligence (UAI'02)*, pages 69–76.

Bundy A., Giunchiglia F., Sebastiani R. et Walsh T. (1996). Computing abstraction hierarchies by numerical simulation. Dans *Proceedings of 13th National Conference on Artificial Intelligence (AAAI'96) - Volume 1*, pages 523–529.

Büttner M. et Rintanen J. (2005). Satisfiability planning with constraints on the number of actions. Dans *Proceedings of 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, pages 292–299.

Bylander T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, **69**(1-2), 165–204.

Bylander T. (1997). A linear programming heuristic for optimal planning. Dans *Proceedings of 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 694–699.

Cabon B., de Givry S., Lobjois L., Schiex T. et Warners J. P. (1999). Radio link frequency assignment. *Constraints*, **4**(1), 79–89.

Cayrol M., Régnier P. et Vidal V. (2000). New results about lcgp, a least committed graphplan. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 273–282.

Cayrol M., Régnier P. et Vidal V. (2001). Least commitment in graphplan. *Artificial Intelligence*, **130**(1), 85–118.

Chapman D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, **32**(3), 333–377.

Chen Y., Hsu C. et Wah B. (2004). SGPlan : subgoal partitioning and resolution in planning. Dans *Proceedings of 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 30–33.

Chen Y., Lv Q. et Huang R. (2008a). Optimizing action costs in planning by SAT-constrained optimization. Dans *Proceedings of 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, pages to appear.

Chen Y., Lv Q. et Huang R. (2008b). Plan-a : A cost-optimal planner based on SAT-constrained optimization. Dans *Proceedings of 6th International Planning Competition (IPC'2008)*.

Chen Y., Zhao X. et Zhang W. (2007). Long-distance mutual exclusion for propositional planning. Dans *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1840–1845.

Cheng J. et Irani K. B. (1989). Ordering problem subgoals. Dans *Proceedings of 11th International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 931–936.

Cohen D., Cooper M. C., Jeavons P. et Krokhin A. (2006). The complexity of soft constraint satisfaction. *Artificial Intelligence*, **170**(11), 983–1016.

Coles A., Fox M., Long D. et Smith A. (2008). Additive-disjunctive heuristics for optimal planning. Dans *Proceedings of 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*.

Cooper M. C. (2003). Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, **134**(3), 311–342.

Cooper M. C., Cussat-Blanc S., de Roquemaurel M. et Régnier P. (2006a). Résolution optimale d'un graphe de planification numérique à l'aide de CSP pondérés. Dans F. Garcia et G. Verfaillie, Eds., *Actes des 1ère Journées Francophones Planification, Décision, Apprentissage pour la conduite de systèmes (JFPDA'06)*, pages 15–21 : INRA - ONERA.

Cooper M. C., Cussat-Blanc S., de Roquemaurel M. et Régnier P. (2006b). Soft arc consistency applied to optimal planning. Dans *Proceedings of 12th International Conference Principles and Practice of Constraint Programming (CP'06)*, volume 4204 of *Lecture Notes in Computer Science*, pages 680–684 : Springer.

Cooper M. C., de Givry S., Sánchez M., Schiex T. et Zytnicki M. (2008a). Cohérence d'arc virtuelle pour les csp pondérés. Dans *Actes des 4ième Journées Francophones de Programmation par Contraintes (JFPC'08)*.

Cooper M. C., de Givry S., Sánchez M., Schiex T. et Zytnicki M. (2008b). Virtual arc consistency for weighted CSP. Dans *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 253–258 : AAAI Press.

Cooper M. C., de Givry S. et Schiex T. (2007a). Optimal soft arc consistency. Dans *Proceedings of 20th International Joint Conference on Artificial Intelligence, (IJCAI'07)*, pages 68–73.

Cooper M. C., de Roquemaurel M. et Régnier P. (2007b). Recherche d'une solution optimale dans des graphes de planification avec actions valuées. Dans *Actes des 2ème Journées Francophones Planification, Décision, Apprentissage pour la conduite de systèmes (JFPDA'07)*, pages 37–48 : Cépaduès Editions.

Cooper M. C., de Roquemaurel M. et Régnier P. (2007c). Un algorithme pour la résolution optimale de problèmes de planification valués. Dans *Journées d'Intelligence Artificielle Fondamentale (IAF'07)*, pages (en ligne) : GRD I3.

Cooper M. C., de Roquemaurel M. et Régnier P. (2008c). Un algorithme pour la résolution optimale de problèmes de planification valués. Dans *Congrès Francophone de Reconnaissance des Formes et Intelligence Artificielle (RFIA'08)*, pages 534–543 : Association Francaise d'Intelligence Artificielle (AFIA).

Cooper M. C. et Schiex T. (2004). Arc consistency for soft constraints. *Artificial Intelligence*, **154**(1-2), 199–227.

Davis M., Logemann G. et Loveland D. (1962). A machine program for theorem-proving. *Communications of the ACM*, **5**(7), 394–397.

Davis M. et Putnam H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, **7**(3), 201–215.

de Givry S., Heras F., Zytnicki M. et Larrosa J. (2005). Existential arc consistency : Getting closer to full arc consistency in weighted CSPs. Dans *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 84–89.

de Givry S., Larrosa J., Meseguer P. et Schiex T. (2003). Solving Max-SAT as weighted CSP. Dans *Proceedings of 9th Principles and Practice of Constraint Programming (CP'03)*, volume 2833 of *Lecture Notes in Computer Science*, pages 363–376 : Springer.

Dimopoulos Y. (2001). Improved integer programming models and heuristic search for AI planning. Dans *Proceedings of 6th European Conference on Planning (ECP'01)*, pages 301–313.

Dimopoulos Y., Gerevini A., Haslum P. et Saetti A. (2006). The benchmark domains of the deterministic part of IPC-5. Dans *Proceedings of 5th International Planning Competition (IPC'2006)*.

Dimopoulos Y., Nebel B. et Koehler J. (1997). Encoding planning problems in nonmonotonic logic programs. Dans *Proceedings of 4th European Conference on Planning (ECP'97)*, pages 169–181.

Do M. B., Benton J., van den Briel M. et Kambhampati S. (2007). Planning with goal utility dependencies. Dans *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1872–1878.

Do M. B. et Kambhampati S. (2000). Solving planning-graph by compiling it into CSP. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 82–91.

Do M. B. et Kambhampati S. (2001). Planning as constraint satisfaction : Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, **132**(2), 151–182.

Do M. B., Srivastava B. et Kambhampati S. (2000). Investigating the effect of relevance and reachability constraints on SAT encodings of planning. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 308–314.

Edelkamp S. (2001). Planning with pattern databases. Dans *Proceedings of 6th European Conference on Planning (ECP'01)*, pages 13–24.

Edelkamp S. (2003). Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, **20**, 195–238.

Edelkamp S. (2005). External symbolic heuristic search with pattern databases. Dans *Proceedings of 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, pages 51–60.

Edelkamp S. (2006). Optimal symbolic PDDL3 planning with MIPS-BDD. Dans *Proceedings of 5th International Planning Competition (IPC'2006)*.

Edelkamp S. et Helmert M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. Dans *Proceedings of 5th European Conference on Planning, Recent Advances in AI Planning (ECP'99)*, volume 1809 of *Lecture Notes in Computer Science*, pages 135–147 : Springer.

Edelkamp S. et Helmert M. (2001). MIPS : the model-checking integrated planning system. *AI Magazine*, **22**(3), 67–72.

Edelkamp S., Jabbar S., et Nazih M. (2006a). Large-scale optimal PDDL3 planning with MIPS-XXL. Dans *Proceedings of 5th International Planning Competition (IPC'2006)*.

Edelkamp S. et Jabbar S. (2006). Cost-optimal external planning. Dans *Proceedings of 21st AAAI Conference on Artificial Intelligence (AAAI'06)* : AAAI Press.

Edelkamp S., Jabbar S. et Nazih M. (2006b). Cost-optimal planning planning with constraints and preferences in large state spaces. Dans *Proceedings of ICAPS'06 Workshop on Preferences and Soft Constraints in Planning*.

Edelkamp S. et Kissmann P. (2008a). Limits and possibilities of BDDs in state space search. Dans D. Fox et C. P. Gomes, Eds., *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 1452–1453 : AAAI Press.

EDELKAMP S. ET KISSMANN P. (2008b). Optimal net-benefit with BDDs. Dans *Proceedings of ICAPS'08 Workshop on Oversubscribed Planning & Scheduling.*

EDELKAMP S. ET KISSMANN P. (2008c). Partial symbolic pattern databases for optimal sequential planning. Dans *Proceedings of 31st Annual German Conference on AI (KI'08)*, volume 5243 of *Lecture Notes in Computer Science*, pages 193–200 : Springer.

EITER T., FABER W., LEONE N., PFEIFER G. ET POLLERES A. (2003). Answer set planning under action costs. *Journal of Artificial Intelligence Research*, **19**, 25–71.

ERNST M., MILLSTEIN T. D. ET WELD D. S. (1997). Automatic SAT-compilation of planning problems. Dans *Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1169–1177.

FARGIER H. ET LANG J. (1993). Uncertainty in constraint satisfaction problems : a probalistic approach. Dans *Proceedings of European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'93)*, pages 97–104.

FIKES R. ET NILSSON N. J. (1971). STRIPS : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**, 189–208.

FOX M. ET LONG D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, **9**, 367–421.

FOX M. ET LONG D. (2000). Utilizing automatically inferred invariants in graph construction and search. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 102–111.

GAREY M. R. ET JOHNSON D. S. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness.* New York : W.H. Freeman and Company.

GEREVINI A. ET LONG D. (2006). Preferences and soft constraints in PDDL3. Dans *Proceedings of ICAPS'06 Workshop on Planning with Preferences and Soft Constraints*, pages 46–53.

GEREVINI A., SAETTI A. ET SERINA I. (2004). Planning with numerical expressions in LPG. Dans *Proceedings of 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 667–671.

GEREVINI A. ET SCHUBERT L. K. (1998). Inferring state constraints for domain-independent planning. Dans *Proceedings of 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 905–912.

GEREVINI A. ET SCHUBERT L. K. (2000). Discovering state constraints in DISCOPLAN : Some new results. Dans *Proceedings of 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 761–767 : AAAI Press / The MIT Press.

GIUNCHIGLIA E. ET MARATEA M. (2006). optsat : A tool for solving SAT related optimization problems. Dans *Proceedings of 10th European Conference on Logics in Artificial Intelligence (JELIA'06)*, pages 485–489.

GIUNCHIGLIA E. ET MARATEA M. (2007a). Planning as satisfiability with preferences. Dans *Proceedings of 22nd AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 987–992.

GIUNCHIGLIA E. ET MARATEA M. (2007b). SAT-Based planning with minimal-#actions plans and "soft" goals. Dans *Proceedings of 10th Congress of the Italian Association for Artificial Intelligence on Artificial Intelligence and Human-Oriented Computing (AI*IA'07)*, volume 4733 of *Lecture Notes in Computer Science*, pages 422–433 : Springer.

GOMES C. P., KAUTZ H., SABHARWAL A. ET SELMAN B. (2008). *Satisfiability Solvers*, Dans *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 89–134. Elsevier.

GRANDCOLAS S. ET PAIN-BARRE C. (2006a). FDP : Filtering and decomposition for planning. Dans *Proceedings of 5th International Planning Competition (IPC'2006)*.

GRANDCOLAS S. ET PAIN-BARRE C. (2006b). A filtering and decomposition approach to optimal sequential planning. Dans *Proceedings of ICAPS'06 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, pages 15–22.

GRANDCOLAS S. ET PAIN-BARRE C. (2007). Filtering, decomposition and search space reduction for optimal sequential planning. Dans *Proceedings of 22nd AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 993–998.

GREGORY P., LONG D. ET FOX M. (2007). A meta-CSP model for optimal planning. Dans *Proceedings of 7th International Symposium Abstraction, Reformulation, and Approximation (SARA'07)*, volume 4612 of *Lecture Notes in Computer Science*, pages 200–214 : Springer.

GUERE E. ET ALAMI R. (2001). One action is enough to plan. Dans *Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 439–444.

HASLUM P. (2004). TP4'04 and HSP^*_a. Dans *Proceedings of 4th International Planning Competition (IPC'2004)*.

HASLUM P. (2006). Improving heuristics through relaxed search - an analysis of TP4 and HSP*a in the 2004 planning competition. *Journal of Artificial Intelligence Research*, **25**, 233–267.

HASLUM P., BONET B. ET GEFFNER H. (2005). New admissible heuristics for domain-independent planning. Dans *Proceedings of 20th National Conference on Artificial Intelligence (AAAI'05)*, pages 1163–1168.

HASLUM P., BOTEA A., HELMERT M., BONET B. ET KOENIG S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. Dans *Proceedings of 22nd AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 1007–1012 : AAAI Press.

HASLUM P. ET GEFFNER H. (2000). Admissible heuristics for optimal planning. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 140–149.

HASLUM P. ET GEFFNER H. (2001). Heuristic planning with time and resources. Dans *Proceedings of 6th European Conference on Planning (ECP'01)*, pages 121–132.

HELMERT M. (2003). Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, **143**(2), 219–262.

HELMERT M., HASLUM P. ET HOFFMANN J. (2007). Flexible abstraction heuristics for optimal sequential planning. Dans *Proceedings of 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*.

HELMERT M., HASLUM P. ET HOFFMANN J. (2008). Explicit-state abstraction : A new method for generating heuristic functions. Dans *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 1547–1550 : AAAI Press.

HERAS F. ET LARROSA J. (2006). Intelligent variable orderings and re-orderings in dac-based solvers for wcsp. *Journal of Heuristics*, **12**(4-5), 287–306.

HICKMOTT S. L., RINTANEN J., THIÉBAUX S. ET WHITE L. B. (2007). Planning via petri net unfolding. Dans *Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1904–1911.

HOFFMANN J. (2001). Local search topology in planning benchmarks : An empirical analysis. Dans *Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 453–458.

HOFFMANN J. (2002). Local search topology in planning benchmarks : A theoretical analysis. Dans *Proceedings of 6th International Conference on Artificial Intelligence Planning Systems (AIPS'02)*, pages 92–100.

HOFFMANN J. (2003). The metric-FF planning system : Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, **20**, 291–341.

HOFFMANN J. (2005). Where 'ignoring delete lists' works : Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, **24**, 685–758.

HOFFMANN J., EDELKAMP S., THIÉBAUX S., ENGLERT R., DOS S. LIPORACE F. ET TRÜG S. (2006). Engineering benchmarks for planning : the domains used in the deterministic part of IPC-4. *Journal of Artificial Intelligence Research*, **26**, 453–541.

HOFFMANN J. ET GEFFNER H. (2003). Branching matters : Alternative branching in graphplan. Dans *Proceedings of 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pages 22–31.

HOFFMANN J. ET KOEHLER J. (1999). A new method to index and query sets. Dans *Proceedings of 6th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 462–467.

HOFFMANN J. ET NEBEL B. (2001). The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, **14**, 253–302.

HOFFMANN J., PORTEOUS J. ET SEBASTIA L. (2004). Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, **22**, 215–278.

Irani K. B. et Cheng J. (1987). Subgoal ordering and goal augmentation for heuristic problem solving. Dans *Proceedings of 10th International Joint Conference on Artificial Intelligence (IJCAI'87)*, pages 1018–1024.

Jacopin E. (1993). *Algorithmique de l'interaction : le cas de la planification : contribution à une classification des systèmes de planification en Intelligence Artificielle*. PhD thesis, Université de Paris 6.

Jacopin E. et Penon J. (2000). On the path from classical planning to arithmetic constraint satisfaction. Dans *Proceedings of AAAI'00 Workshop on Constraints and AI Planning*, pages 18–24.

Jacopin E. et Puget J. F. (1990). *From TWEAK to PWEAK : reducing the non-linear planning framework*. Rapport LAFORIA 29/90, Institut Blaise Pascal, Université Paris 6 et 7.

Jónsson A. K., Morris P. H., Muscettola N., Rajan K. et Smith B. D. (2000). Planning in interplanetary space : Theory and practice. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 177–186.

Joslin D. et Pollack M. E. (1996). Is "early commitment" in plan generation ever a good idea ? Dans *Proceedings of 13th National Conference on Artificial Intelligence (AAAI'96)*, volume 2, pages 1188–1193.

Joslin D. et Roach J. W. (1989). A theoretical analysis of conjunctive-goal problems. *Artificial Intelligence*, **41**(1), 97–106.

Kambhampati S. (1994). Multi-contributor causal structures for planning : A formalization and evaluation. *Artificial Intelligence*, **69**(1-2), 235–278.

Kambhampati S. (1997). Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, **18**(2), 67–97.

Kambhampati S. (1998). On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. *Artificial Intelligence*, **105**(1-2), 161–208.

Kambhampati S. (2000). Planning graph as a (dynamic) CSP : exploiting EBL, DDB and other CSP search techniques in graphplan. *Journal of Artificial Intelligence Research*, **12**, 1–34.

Kambhampati S., Katukam S. et Qu Y. (1996). Failure driven dynamic search control for partial order planners : An explanation based approach. *Artificial Intelligence*, **88**(1-2), 253–315.

Kambhampati S. et Nigenda R. S. (2000). Distance-based goal-ordering heuristics for graphplan. Dans *Proceedings of 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 315–322.

Kambhampati S., Parker E. et Lambrecht E. (1997). Understanding and extending graphplan. Dans *Proceedings of 4th European Conference on Planning (ECP'97)*, pages 260–272.

KARMARKAR N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, **4**(4), 373–396.

KATZ M. ET DOMSHLAK C. (2008a). Optimal additive composition of abstraction-based admissible heuristics. Dans *Proceedings of 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*.

KATZ M. ET DOMSHLAK C. (2008b). Structural pattern heuristics via fork decomposition. Dans *Proceedings of 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*.

KAUTZ H. A. (2004). SATPLAN04 : planning as satisfiability. Dans *Proceedings of 4th International Planning Competition (IPC'2004)*.

KAUTZ H. A. (2006). Deconstructing planning as satisfiability. Dans *Proceedings of 21st National Conference on Artificial Intelligence (AAAI'06)*.

KAUTZ H. A., MCALLESTER D. ET SELMAN B. (1996). Encoding plans in propositional logic. Dans *Proceedings of 5th International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, pages 374–384.

KAUTZ H. A. ET SELMAN B. (1992). Planning as satisfiability. Dans *Proceedings of 10th European Conference on Artificial Intelligence (ECP'92)*, pages 359–363.

KAUTZ H. A. ET SELMAN B. (1996). Pushing the envelope : Planning, propositional logic and stochastic search. Dans *Proceedings of 13th National Conference on Artificial Intelligence (AAAI'96)*, pages 1194–1201.

KAUTZ H. A. ET SELMAN B. (1998). BLACKBOX : A new approach to the application of theorem proving to problem solving. Dans *Working notes of (AIPS'98), Workshop on Planning as Combinatorial Search*.

KAUTZ H. A. ET SELMAN B. (1999). Unifying SAT-based and graph-based planning. Dans *Proceedings of 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325.

KAUTZ H. A., SELMAN B. ET HOFFMANN J. (2006). SATPLAN : planning as satisfiability. Dans *Proceedings of 5th International Planning Competition (IPC'2006)*.

KAUTZ H. A. ET WALSER J. P. (1999). State-space planning by integer optimization. Dans *Proceedings of 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 526–533.

KELLEHER G. ET COHN A. G. (1992). Automatically synthesising domain constraints from operator descriptions. Dans *Proceedings of 10th European Conference on Artificial Intelligence (ECAI'92)*, pages 653–655.

KNOBLOCK C. A. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, **68**(2), 243–302.

KNOBLOCK C. A. ET YANG Q. (1993). A comparison of the SNLP and TWEAK planning algorithms. Dans *Working notes of AAAI'93 Spring Symposium on Foundations of Automatic Planning*, pages 73–77.

KOEHLER J. (1998). Solving complex planning tasks through extraction of subproblems. Dans *Proceedings of 4th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 62–69.

KOEHLER J. ET HOFFMANN J. (1999). *Handling of Inertia in a Planning System.* Rapport interne 122/99, Albert-Ludwigs University.

KOEHLER J. ET HOFFMANN J. (2000). On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, **12**, 338–386.

KOEHLER J., NEBEL B., HOFFMANN J. ET DIMOPOULOS Y. (1997). Extending planning graphs to an ADL subset. Dans *Proceedings of 4th European Conference on Planning (ECP'97)*, pages 273–285.

KVARNSTRÖM J. (2002). Applying domain analysis techniques for domain-dependent control in TALplanner. Dans *Proceedings of 6th International Conference on Artificial Intelligence Planning Systems (AIPS'02)*, pages 101–111.

LABORIE P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling : existing approaches and new results. *Artificial Intelligence*, **143**(2), 151–188.

LABORIE P. ET GHALLAB M. (1995). Planning with sharable resource constraints. Dans *Proceedings of 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1643–1651.

LARROSA J. (2002). Node and arc consistency in weighted CSP. Dans *Proceedings of 18th National Conference on Artificial Intelligence (AAAI'02)*, pages 48–53.

LARROSA J. ET SCHIEX T. (2003). In the quest of the best form of local consistency for weighted CSP. Dans *Proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 239–244.

LONG D. ET FOX M. (1999). Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, **10**, 87–115.

LONG D. ET FOX M. (2003). The 3rd international planning competition : Results and analysis. *Journal of Artificial Intelligence Research*, **20**, 1–59.

LOPEZ A. ET BACCHUS F. (2003). Generalizing graphplan by formulating planning as a CSP. Dans *Proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 954–960.

MACKWORTH A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, **8**(1), 99–118.

Mali A. D. et Kambhampati S. (1998a). Encoding HTN planning in propositional logic. Dans *Proceedings of 4th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 190–198.

Mali A. D. et Kambhampati S. (1998b). Frugal propositional encodings for planning. Dans *Proceedings of AIPS'98 Workshop on Planning as Combinatorial Search.*

Mali A. D. et Kambhampati S. (1999). On the utility of plan-space (causal) encodings. Dans *Proceedings of 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 557–563.

Maris F., Régnier P. et Vidal V. (2004). Planification SAT : Amélioration des codages et traduction automatique. Dans *Proceedings du 14ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle (RFIA'04)*, pages 1019–1028.

McAllester D. A. et Rosenblitt D. (1991). Systematic nonlinear planning. Dans *Proceedings of 9th National Conference on Artificial Intelligence (AAAI'91)*, pages 634–639.

McCarthy J. et Hayes P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. Dans B. Meltzer et D. Michie, Eds., *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press.

Mccluskey T. L. (1988). *Experience-driven heuristic acquisition in general problem solvers.* PhD thesis, The City University (London) (United Kingdom).

McDermott D., Ghallab M., Howe A., Knoblock C., Ram A., Veloso M., Weld D. et Wilkins D. (1998). *PDDL - The Planning Domain Definition Language - Version 1.2.* Rapport interne, CVC TR-98-003, Yale Center for Computational Vision and Control.

Meseguer P., Rossi F. et Schiex T. (2006). *Soft Constraints*, Dans *Handbook of Constraint Programming*, pages 281–328. Elsevier. ISBN : 0-444-52726-5.

Miguel I., Jarvis P. et Shen Q. (2000). Flexible graphplan. Dans *Proceedings of 14th European Conference on Artificial Intelligence (ECAI'00)*, pages 506–510.

Miguel I., Shen Q. et Jarvis P. (2001). Efficient flexible planning via dynamic flexible constraint satisfaction. *Engineering Applications of Artificial Intelligence*, **14**(3), 301–327.

Mittal S. et Falkenhainer B. (1990). Dynamic constraint satisfaction problems. Dans *Proceedings of 8th National Conference on Artificial Intelligence (AAAI'90)*, pages 25–32.

Montanari U. (1974). Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences*, **7**, 95–132.

Nareyek A., Freuder E. C., Fourer R., Giunchiglia E., Goldman R. P., Kautz H., Rintanen J. et Tate A. (2005). Constraints and AI planning. *IEEE Intelligent Systems*, **20**(2), 62–72.

Nguyen X. et Kambhampati S. (2000). Extracting effective and admissible state space heuristics from the planning graph. Dans *Proceedings of 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 798–805.

Nguyen X. et Kambhampati S. (2001). Reviving partial order planning. Dans *Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 459–466 : Morgan Kaufmann.

Nguyen X., Kambhampati S. et Nigenda R. S. (2002). Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, **135**(1-2), 73–123.

Parker E. (2004). Combining backward-chaining with forward-chaining ai search. Dans *Proceedings of 4th International Planning Competition (IPC'2004)*.

Pednault E. P. D. (1989). ADL : Exploring the middle ground between STRIPS and the situation calculus. Dans *Proceedings of 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332.

Penberthy J. S. et Weld D. S. (1992). UCPOP : A sound, complete, partial order planner for ADL. Dans *Proceedings of 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 103–114.

Penberthy J. S. et Weld D. S. (1994). Temporal planning with continuous change. Dans *Proceedings of 12th National Conference on Artificial Intelligence (AAAI'94)*, pages 1010–1015.

Porteous J., Sebastia L. et Hoffmann J. (2001). On the extraction, ordering, and usage of landmarks in planning. Dans *Proceedings of 6th European Conference on Planning (ECP'01)*.

Richter S., Helmert M. et Westphal M. (2008). Landmarks revisited. Dans *Proceedings of 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 975–982 : AAAI Press.

Rintanen J. (1998). A planning algorithm not based on directional search. Dans *Proceedings of 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy*, pages 617–625.

Rintanen J. (2000). An iterative algorithm for synthesizing invariants. Dans *Proceedings of 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 806–811.

Rintanen J. (2003). Symmetry reduction for SAT representations of transition systems. Dans *Proceedings of 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, pages 32–41.

Rintanen J. (2004). Evaluation strategies for planning as satisfiability. Dans *Proceedings of 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 682–687.

Rintanen J. (2006). Unified definition of heuristics for classical planning. Dans *Proceedings of 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 600–604 : IOS Press.

Rintanen J., Heljanko K. et Niemelä I. (2004). Parallel encodings of classical planning as satisfiability. Dans *Proceedings of 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, pages 307–319.

Rintanen J., Heljanko K. et Niemelä I. (2006). Planning as satisfiability : parallel plans and algorithms for plan search. *Artificial Intelligence*, **170**(12-13), 1031–1080.

ROBINSON N., GRETTON C. ET PHAM D.-N. (2008a). CO-PLAN : combining SAT-based planning with forward-search. Dans *Proceedings of 6th International Planning Competition (IPC'2008)*.

ROBINSON N., GRETTON C., PHAM D.-N. ET SATTAR A. (2008b). A compact and efficient SAT encoding for planning. Dans *Proceedings of 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*.

ROSENFELD A., HUMMEL R. A. ET ZUCKER S. W. (1976). Scene labeling by relaxation operations. *IEEE Transactions on Systems, Man and Cybernetics*, **6**(6), 420–433.

SACERDOTI E. D. (1975). The nonlinear nature of plans. Dans *Proceedings of 4th International Joint Conference on Artificial Intelligence (IJCAI'75)*, pages 206–214.

SCHIEX T. (1992). Possibilistic constraint satisfaction problems or "how to handle soft constraints ?". Dans *Proceedings of 8th Annual Conference on Uncertainty in Artificial Intelligence (UAI'92)*, pages 268–275.

SCHIEX T. (2000). Arc consistency for soft constraints. Dans *Proceedings of 6th International Conference on Principles and Practice of Constraint Programming (CP'00)*, volume 1894 of *Lecture Notes in Computer Science*, pages 411–424 : Springer.

SCHIEX T., FARGIER H. ET VERFAILLIE G. (1995). Valued constraint satisfaction problems : Hard and easy problems. Dans *Proceedings of 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 631–637.

SCHRIJVER A. (1998). *The Theory of Linear and Integer Programming*. John Wiley & Sons.

SMITH D. E. ET WELD D. S. (1999). Temporal planning with mutual exclusion reasoning. Dans *Proceedings of 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 326–337.

SRIVASTAVA B. (2000). Realplan : Decoupling causal and resource reasoning in planning. Dans *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 812–818 : AAAI Press / The MIT Press.

SRIVASTAVA B. ET KAMBHAMPATI S. (1999). Scaling up planning by teasing out resource scheduling. Dans *Proceedings of 5th European Conference on Planning, Recent Advances in AI Planning (ECP'99)*, volume 1809 of *Lecture Notes in Computer Science*, pages 172–186 : Springer.

STEFIK M. (1981). Planning with constraints (MOLGEN : part 1). *Artificial Intelligence*, **16**(2), 111–140.

TRUSTRUM K. (1971). *Linear Programming*. Routledge & Kegan Paul.

TSUNETO R., HENDLER J. A. ET NAU D. S. (1998). Analyzing external conditions to improve the efficiency of HTN planning. Dans *Proceedings of 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 913–920.

van Beek P. et Chen X. (1999). CPlan : a constraint programming approach to planning. Dans *Proceedings of 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 585–590.

van den Briel M., Benton J., Kambhampati S. et Vossen T. (2007). An LP-based heuristic for optimal planning. Dans *Proceedings of 13th International Conference, Principles and Practice of Constraint Programming (CP'07)*, pages 651–665.

van den Briel M. et Kambhampati S. (2005). Optiplan : Unifying IP-based and graph-based planning. *Journal of Artificial Intelligence Research*, **24**, 919–931.

van den Briel M., Sanchez R., Do M. B. et Kambhampati S. (2004a). Effective approaches for partial satisfaction (over-subscription) planning. Dans *Proceedings of 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 562–569.

van den Briel M., Sanchez R. et Kambhampati S. (2004b). Over-subscription in planning : A partial satisfaction problem. Dans *Proceedings of ICAPS'04 Workshop on Integrating Planning into Scheduling*, pages 91–98.

van den Briel M., Vossen T. et Kambhampati S. (2005). Reviving integer programming approaches for AI planning : A branch-and-cut framework. Dans *Proceedings of 15th International Conference on Automated Planning and Scheduling (ICAPS'05)*, pages 310–319.

van den Briel M., Vossen T. et Kambhampati S. (2008). Loosely coupled formulations for automated planning : An integer programming perspective. *Journal of Artificial Intelligence Research*, **31**, 217–257.

Vidal V. (2001). *Recherche dans les graphes de planification, satisfaisabilité et stratégies de moindre engagement. Les systèmes LCGP et LCDPP.* PhD thesis, IRIT Université Paul Sabatier Toulouse France.

Vidal V. (2002). Le moindre engagement dans une approche de la planification basée sur la procédure de Davis et Putnam. Dans *Actes des 8ièmes Journées Nationales sur la résolution de problèmes NP-Complets (JNPC'02)*, pages 239–253.

Vidal V. (2004). A lookahead strategy for heuristic search planning. Dans *Proceedings of 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 150–160 : AAAI.

Vidal V. et Geffner H. (2004). Branching and pruning : An optimal temporal pocl planner based on constraint programming. Dans *Proceedings of 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 570–577 : AAAI Press / The MIT Press.

Vidal V. et Geffner H. (2006). Branching and pruning : an optimal temporal POCL planner based on constraint programming. *Artificial Intelligence*, **170**(3), 298–335.

Vossen T., Ball M., Lotem A. et Nau D. (1999). On the use of integer programming models in ai planning. Dans *Proceedings of 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 304–309.

Vossen T., Ball M., Lotem A. et Nau D. (2001). Applying integer programming to AI planning. *Knowledge Engineering Review*, **16**(1), 85–100.

Wehrle M., Kupferschmid S. et Podelski A. (2008). Useless actions are useful. Dans *Proceedings of 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*.

Wehrle M. et Rintanen J. (2007). Planning as satisfiability with relaxed ∃-step plans. Dans *Proceedings of 20th Australian Joint Conference on Artificial Intelligence (AUSAI'07)*, pages 244–253.

Weld D. S. (1994). An introduction to least commitment planning. *AI Magazine*, **15**(4), 27–61.

Weld D. S. (1999). Recent advances in AI planning. *AI Magazine*, **20**, 93–123.

Weld D. S., Anderson C. R. et Smith D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. Dans *Proceedings of 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 897–904.

Wolfman S. A. et Weld D. S. (1999). The LPSAT engine and its application to resource planning. Dans *Proceedings of 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 310–316.

Wolfman S. A. et Weld D. S. (2001). Combining linear programming and satisfiability solving for resource planning. *Knowledge Engineering Review*, **16**(1), 585–99.

Xing Z., Chen Y. et Zhang W. (2006). Maxplan : Optimal planning by decomposed satisfiability and backward reduction. Dans *Proceedings of 5th International Planning Competition (IPC'2006)*, pages 53–56.

Yang Q. (1992). A theory of conflict resolution in planning. *Artificial Intelligence*, **58**, 361–392.

Yang Q., Tenenberg J. D. et Woods S. (1996). On the implementation and evaluation of ABTWEAK. *Computational Intelligence*, **12**(2), 295–318.

Zhao X., Chen Y. et Zhang W. (2006). Optimal STRIPS planning by maximum satisfiability and accumulative learning. Dans *Proceedings of 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, pages 442–446.

Zhou R. et Hansen E. A. (2004). Breadth-first heuristic search. Dans *Proceedings of 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 92–100.

Zhou R. et Hansen E. A. (2006). Breadth-first heuristic search. *Artificial Intelligence*, **170**(4-5), 385–408.

Index

Liste des définitions

Liste des symboles

$a\#b$: a et b sont des actions indépendantes, 10
$\mathcal{A}[W]$: projection d'une affectation $\mathcal{A}$ sur W, 36
$c[W]$: projection d'une contrainte c sur W, 36

a^+ : action a relaxée, 76
A^+ : ensemble des actions du graphe relaxé G^+, 76
$A^+_{\geq}$: ensemble des actions du graphe relaxé, stabilisé et réduit G^+_r ordonnées par coûts décroissants, 83
$A^+[i]$: ensemble des actions (excepté nooop) appartenant au niveau i du graphe G^+, 76
A^+_r : ensemble des actions du graphe G^+_r, 76
$A^+_{r,-(X,U)}$: ensembles des actions du graphe $G^+_{r,-(X,U)}$, 79
$A_{-(X,U)}$: ensemble des actions de A dans lequel on ignore les actions de X puis les fluents de U dans les effets des actions, 78
$A^+_{-(X,U)}$: ensemble des actions de A^+ dans lequel on ignore les actions de X et l'établissement des fluents de U, 78
AC : cohérence d'arc, 42
add_f : ensemble des actions ajoutant le fluent f, 78
$add(a)$: ensemble des ajouts de l'action a, 9

$B - C$: différence entre deux ensembles, 97
BDD : diagramme de décision binaires, 29
b_f : nombre de fois que le fluent f est vrai dans le but, 99

C^*_k : coût d'un plan-solution P^*_k, 23
C^* : coût optimal, 25
C_{min} : coût minimum des actions, 122
$cout(a)$: coût de l'action a, 22
CP : programmation par contrainte, 20
CPT : transformation préservant le coût, 111
CSP : problème de satisfaction de contraintes, 20, 36

DAC : cohérence d'arc directionnelle, 43
DCSP : CSP dynamique, 21
del_f : ensemble des actions retirant le fluent f, 78
$del(a)$: ensemble des retraits de l'action a, 9

E^+ : ensemble des fluents positifs associés à E, 97
E^- : ensemble des fluents négatifs associés à E, 97
EAC : cohérence d'arc existentielle, 46
EDAC : cohérence d'arc existentielle et directionnelle, 46
e_f : nombre de fois que le fluent f est établi, 99
$e_{\neg f}$: nombre de fois que le fluent f passe de la valeur vrai ou indéfini à la valeur faux, 99
eff_f : ensemble des actions ayant le fluent f dans ses effets, 78
$eff(a)$: ensemble des effets de l'action a, 9
e^{utile}_f : nombre de fois que le fluent f est utilement établi, 102

$F^+[i]$: ensemble des fluents appartenant au niveau i du graphe G^+, 76
F_A : version positives et négatives des fluents associés à l'ensemble des actions A, 97
F^+_A : ensemble des fluents positifs associés à l'ensemble des actions A, 97
FAC : cohérence d'arc totale, 43
FDAC : cohérence d'arc directionnelle totale, 44

G^+ : graphe stabilisé du problème Π^+, 76
G^+_r : graphe stabilisé et réduit du problème Π^+, 76

$G^+_{r,-(X,U)}$: graphe stabilisé et réduit du problème $\Pi^+_{-(X,U)}$, 79

i_f : nombre de fois que le fluent f est vrai dans l'état initial, 99
i_f^{utile} : nombre de fois que le fluent f est utilement présent dans l'état initial I, 102
IP : programme linéaire en nombres entiers, 19
0-1-IP : programme binaire en nombres entiers, 19
IPC : International Planning Competition, 4

$X \leftarrow v$: Affectation de la valeur v à la variable X, 36
LP : programmation linéaire, 19

$mincout(X)$: minimum des coûts des actions de X, 78

NC : cohérence de nœud, 41

o_a : nombre de fois que l'action a apparaît, 101
O_k : ensemble des actions trop onéreuses à partir du niveau k, 87
OSAC : cohérence d'arc optimale, 47
$\overline{T}$: ensemble des négations des fluents de T, 97

$\mathcal{P}^+$: classe des problèmes de planification optimaux sans retrait, 77
PDB : bases de données de motifs, 28
Π^+ : problème Π relaxé, 76
$\Pi^+_{-(X,U)}$: problème Π^+ dans lequel on ignore les actions de X puis les fluents de U dans I et dans les ajouts des actions, 78
$\Pi_{-(X,U)}$: problème Π dans lequel on ignore les actions de X puis les fluents de U dans I et dans les effets des actions, 78
$\mathcal{P}_k$: ensemble des plans-solutions de longueur inférieure ou égale à k étapes, 23
$\mathcal{P}^*_k$: ensemble des plans-solutions de $\mathcal{P}_k$ qui minimisent la métrique, 23

P^* : plan de coût optimal, 25
P^*_k : plan appartenant à $\mathcal{P}^*_k$, 23
$X_i \prec X_j$: instanciation de X_i avant X_j, 43
$prec_f$: ensemble des actions ayant le fluent f en précondition, 78
$prec(a)$: ensemble des préconditions de l'action a, 9

SAC : cohérence d'arc souple, 42
SAT : problème de satisfiabilité, 16

$E \uparrow a$: application d'une action a à un état E, 9
$E \uparrow Q$: application d'un ensemble d'actions Q sur un état E, 10
$E \uparrow P$: application d'un plan P sur un état E, 10

$\mathcal{V}(X_i)$: voisinage de la variable X_i, 42
$\mathcal{V}^+(X_i)$: voisinage supérieur de la variable X_i, 43
$\mathcal{V}^-(X_i)$: voisinage inférieur de la variable X_i, 43
VAC : cohérence d'arc virtuelle, 48

WCSP : problème de satisfaction de contraintes pondérées, 37

$\mathcal{X}$: ensemble d'ensembles indispensables d'actions, 124

$\mathcal{Z}$: ensemble des actions indispensables, 124

Liste des algorithmes

Liste des figures

Liste des tableaux

Table des matières

Marie de ROQUEMAUREL

A weighted CSP approach to cost-optimal planning

Abstract

For planning to come of age, plans must be judged by a measure of quality, such as the total cost of actions. This thesis describes an optimal-cost planner in the classical planning framework except that each action has a cost.

We code the extraction of an optimal plan, from a planning graph with a fixed number k of levels, as a weighted constraint satisfaction problem (WCSP). The specific structure of the resulting WCSP means that a state-of-the-art exhaustive solver was able to find an optimal plan in planning graphs containing several thousand nodes.

We present several methods for determining a tight bound on the number of planning-graph levels required to ensure finding a globally optimal plan. These include universal notions such as indispensable sets S of actions : every valid plan contains at least one action in S. Different types of indispensable sets can be rapidly detected by solving relaxed planning problems related to the original problem. On extensive trials on benchmark problems, the bound on the number of planning-graph levels was reduced by an average of 60% allowing us to solve many instances to optimality.

Thorough experimental investigations demonstrated that using the planning graph in optimal planning is a practical possibility, although not competitive, in terms of computation time, with a recent state-of-the-art optimal planner.

KEYWORDS

Cost-optimal planning, planning graph, weighted CSP

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Toulouse III - Paul Sabatier, 118 route de Narbonne
31062 TOULOUSE cedex 9

Marie de ROQUEMAUREL

Planification de coût optimal basée sur les CSP pondérés

Directeur de thèse : Pierre REGNIER, Maître de conférences, HdR
Lieu et date de soutenance : Université Toulouse III - 12 Mars 2009
Discipline Administrative : Informatique

Résumé

Un des challenges actuels de la planification est la résolution de problèmes pour lesquels on cherche à optimiser la qualité d'une solution telle que le coût d'un plan-solution. Dans cette thèse, nous développons une méthode originale pour la planification de coût optimal dans un cadre classique non temporel et avec des actions valuées.

Pour cela, nous utilisons une structure contenant tous les plans d'une longueur donnée, appelé graphe de planification. L'extraction d'une solution optimale, à partir d'un graphe de planification de longueur fixée, est codée comme un problème de satisfaction de contraintes pondérées (WCSP). La structure spécifique des WCSP obtenus permet aux solveurs actuels de trouver, pour une longueur donnée, une solution optimale dans un graphe de planification contenant plusieurs centaines de nœuds.

Nous présentons ensuite plusieurs méthodes pour déterminer la longueur maximale des graphes de planification nécessaire pour garantir l'obtention d'une solution de coût optimal. Ces méthodes incluent plusieurs notions universelles comme par exemple la notion d'ensembles d'actions indispensables pour lesquels toutes les solutions contiennent au moins une action de l'ensemble. Ces ensembles peuvent être rapidement détectés en résolvant une version relaxée du problème original. Les résultats expérimentaux effectués montrent que l'utilisation de ces méthodes permet une diminution de 60% en moyenne de la longueur requise pour garantir l'obtention d'une solution de coût optimal.

La comparaison expérimentale avec d'autres planificateurs montre que l'utilisation du graphe de planification et des CSP pondérés pour la planification optimale est possible en pratique même si elle n'est pas compétitive, en terme de temps de calcul, avec les planificateurs optimaux récents.

Mots-clés

Planification avec actions valuées, Optimalité en coût, Graphe de planification, CSP pondérés

Institut de Recherche en Informatique de Toulouse - UMR 5505
Université Toulouse III - Paul Sabatier, 118 route de Narbonne
31062 TOULOUSE cedex 9