HAL
open science

Planification SAT et Planification Temporellement Expressive. Les Systèmes TSP et TLP-GP.

Frédéric Maris

▶ **To cite this version:**

Frédéric Maris. Planification SAT et Planification Temporellement Expressive. Les Systèmes TSP et TLP-GP.. Informatique [cs]. Université Paul Sabatier - Toulouse III, 2009. Français. NNT : . tel-00442014

HAL Id: tel-00442014

https://theses.hal.science/tel-00442014v1

Submitted on 17 Dec 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université de Toulouse

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 – Intelligence Artificielle*

Présentée et soutenue par ***Frédéric MARIS***
Le *18 septembre 2009*

Titre : *Planification SAT et Planification Temporellement Expressive*

Les Systèmes TSP et TLP-GP

JURY

Martin COOPER	*Professeur d'Université* *IRIT – Université Paul Sabatier*	*(Président)*
Abdel-Illah MOUADDIB	*Professeur d'Université* *GREYC – Université de Caen*	*(Rapporteur)*
Pierre REGNIER	*Maître de conférences, HDR* *IRIT – Université Paul Sabatier*	*(Examinateur)*
Vincent VIDAL	*Maître de conférences* *CRIL – Université d'Artois*	*(Examinateur)*
François CHARPILLET	*Directeur de recherche* *LORIA – INRIA*	*(Rapporteur, absent)*
Thierry VIDAL	*Maître de conférences* *LGP – ENIT*	*(Invité)*

Ecole doctorale : *Mathématiques Informatique Télécommunications de Toulouse*
Unité de recherche : *Institut de Recherche en Informatique de Toulouse*
Directeur de Thèse : *Pierre REGNIER*

SOMMAIRE

I. Introduction ... 3
1. Introduction générale au domaine ... 3
2. Les différents cadres de travail concernés ... 4
2.1. Le cadre classique ... 4
2.2. Le cadre temporel ... 6
2.3. La planification par compilation ... 6
3. Présentation de la thèse ... 7
II. Algorithmes de planification dans le cadre classique ... 9
1. Définitions ... 9
2. Le langage PDDL ... 11
3. Recherche dans les espaces d'états ... 13
3.1. Définitions ... 13
3.2. Algorithmique ... 14
4. Recherche dans les espaces de plans partiels ... 15
4.1. Définitions ... 15
4.2. Algorithmique ... 18
5. Graphplan ... 19
5.1. Définitions ... 19
5.2. Algorithmique ... 21
6. Conclusion ... 23
III. Planification SAT : le système TSP ... 25
1. Introduction ... 25
2. Etat de l'art de la planification SAT ... 27
2.1. Définitions préliminaires ... 27
2.2. Historique ... 27
2.3. Nomenclature des différents codages ... 33
2.4. Codages MK99 et V01 dans les espaces d'états ... 33
2.5. Codages MK99 et V01 dans les espaces de plans ... 35
2.6. Codages KS99 et V01 du graphe de planification ... 41
3. Introduction de la relation d'autorisation dans les codages ... 43
3.1. Définition de la relation d'autorisation ... 43
3.2. Intégration de la relation d'autorisation dans les codages ... 46
4. Automatisation de la traduction : le traducteur TSP ... 49
4.1. Principe de fonctionnement général de TSP ... 49
4.2. Syntaxe et sémantique de TSPL ... 50
4.3. Fonctionnement interne du traducteur TSP ... 53
4.4. Codage des traductions en TSPL ... 55
4.5. Etude de la complexité des codages ... 55
5. Etude comparative ... 56
5.1. Les benchmarks utilisés ... 57
5.2. Etude comparative des traductions MK99 et V01 ... 58
5.3. Etude comparative des traductions V01, LCS et LCP (introduction de l'autorisation) ... 59
5.4. Etude comparative des traductions KS99 et V01 ... 61
6. Etude et amélioration de codages récents ... 63

6.1. Nomenclature des différents codages 63
6.2. Codages avec indépendance et sémantique ∀-Step 64
6.3. Codages avec autorisation et sémantiques ∃-Step / R∃-Step 66
6.4. Un codage compact utilisant la relation d'autorisation faible 68
7. Conclusion 70
IV. Planification temporelle : le système TLP-GP 71
1. Introduction 71
2. Etat de l'art de la planification temporelle 72
2.1. Représentation de problèmes de planification temporels 72
2.2. Les premiers planificateurs temporels 83
2.3. Algorithmique des planificateurs temporels actuels 84
2.4. Taxinomie des principaux planificateurs temporels 96
3. TLP-GP : un planificateur pour résoudre des problèmes temporellement-expressifs 101
3.1. Introduction 101
3.2. Extension de l'expressivité de PDDL2.1 aux intervalles temporels 101
3.3. Algorithmique de TLP-GP 103
3.4. Présentation de l'interface graphique 115
3.5. Extension supplémentaire de l'expressivité 116
3.6. Benchmarks temporellement expressifs 123
3.7. Résultats expérimentaux 127
V. Conclusion et perspectives 135
1. Ce qui a été fait… 135
2. … et ce qui reste à faire. 136

I. Introduction

1. Introduction générale au domaine

Depuis sa création en 1956, les objectifs et techniques employés par l'*Intelligence Artificielle* (IA) n'ont cessé de s'enrichir et de se diversifier. Un de ses buts essentiels demeure cependant la conception d'un *système robotique intelligent*, autonome et agissant. La satisfaction de cet objectif pose de très nombreux problèmes de perception, de fusion de données, de représentation, de résolution de problèmes, de décision, d'exécution d'actions, de réactivité aux événements et de prise en compte du temps, de l'incertitude… L'idée qui sous-tend la conception de tels systèmes robotiques est généralement la réalisation d'une *boucle perception / modélisation / décision / action*. Un très grand nombre d'études ont été faites sur cette base ; elles ont permis des progrès importants mais on est cependant encore très loin de pouvoir réaliser une machine autonome.

Dans ce contexte, la *planification* est un processus cognitif permettant de générer automatiquement, au travers d'une procédure formalisée, un résultat articulé, sous la forme d'un système intégré de décisions appelé *plan*. Ce dernier se présente généralement sous la forme d'une collection organisée de descriptions *d'opérations* ; il est essentiellement destiné à guider l'action d'un ou plusieurs agents exécuteurs qui ont à agir dans un monde particulier pour atteindre un but prédéfini et qui ont donc à prendre des décisions adaptées aux situations successives qu'ils rencontrent. L'*exécution* est la réalisation *d'actions* qui est effectuée en suivant les directives données par le plan. Elle vise à réaliser la prédiction que constitue ce plan ; lorsqu'elle est conforme à ce qu'il indique, elle doit permettre de faire évoluer l'univers de *l'état initial* vers un état satisfaisant le *but*. Un plan est un *plan-solution* si son exécution permet, partant de l'état initial, d'atteindre le but.

Pour concevoir des systèmes robotiques autonomes, il est par conséquent essentiel de simuler le processus de planification de façon à l'intégrer à la fonction décision. L'IA cherche donc à élaborer des méthodes de représentation des connaissances et des algorithmes permettant de les manipuler puis à les implémenter dans des *planificateurs* afin de reproduire et même d'améliorer ce processus. Un des objectifs poursuivis est de pouvoir, à terme, traiter automatiquement des problèmes difficiles à résoudre par l'homme. Selon le processus de planification employé, la structure des plans-solutions produits peut différer : on peut en particulier obtenir des plans dont les opérations seront totalement ou partiellement ordonnées (ce qui permettra de choisir un ordre d'exécution quelconque ou d'exécuter en parallèle celles qui demeurent non ordonnées). La planification automatique commence ainsi à trouver des applications concrètes dans divers domaines comme la robotique autonome, les applications spatiales, la gestion de production, l'aide au calcul scientifique, le commandement militaire…

Depuis 1998, les différents planificateurs sont régulièrement comparés dans des compétitions bi-annuelles IPC (International Planning Competition). Au sein de ces compétitions, ils peuvent concourir dans différentes catégories (planification déterministe, dans l'incertain, avec apprentissage…) et de nombreux jeux de tests, ou *benchmarks*, que l'on essaie de rendre de plus en plus proches de problèmes réels, ont été conçus pour pouvoir tester leurs performances de la manière la plus objective possible.

Malgré les nombreux progrès récents, les applications actuelles de la planification se limitent encore cependant à des domaines restreints et bien circonscrits. En effet, planifier dans le monde réel est très difficile en raison de sa complexité, de l'incomplétude des informations recueillies, de l'inexactitude de ces informations, de l'incertitude qui les entache… La planification ne peut donc être raisonnablement envisagée que sur un monde simulé qui n'est qu'un modèle, une approximation du monde réel. Cette approximation porte sur la connaissance des états du monde et des opérations réalisables (conditions et conséquences de leur réalisation). En fonction des hypothèses restrictives que l'on s'impose pour modéliser le monde réel, plusieurs cadres de travail peuvent alors être envisagés.

2. Les différents cadres de travail concernés

Dans le cadre classique, le plus restrictif, on considère les actions comme des transitions instantanées sans prendre en compte le temps. Des extensions au cadre classique ont aussi été proposées qui permettent de modéliser certains aspects temporels. Dans le travail que nous présenterons ici, nous nous intéresserons d'abord au cadre classique (cf. section 2.1), puis au cadre temporel (cf. section 2.2) pour lequel nous proposerons des extensions. Dans tous ces travaux, nous utiliserons des solveurs dédiés à la résolution de certains types de problèmes en nous plaçant ainsi dans le cadre de la planification "par compilation" (cf. section 2.3).

2.1. Le cadre classique

La large majorité des études réalisées sur la planification considèrent un cadre de travail simplificateur qui permet d'obtenir de bons résultats. Dans ce dernier, les planificateurs actuels permettent de résoudre rapidement (moins de 30 mn.) des problèmes de taille importante. Lors de la compétition IPC'2006, plusieurs planificateurs ont ainsi réussi à résoudre le plus gros problème proposé dans le domaine "Travelling and Purchase" qui modélise des déplacements de marchandises et leurs achats dans différents marchés. Ce problème comporte 8 marchés et 3 dépots reliés de manière aléatoire. Huit camions peuvent s'y 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, Hsu, Wah, 2005], vainqueur de cette compétition, trouve ainsi en moins de 17 minutes un plan-solution contenant près de 400 actions.

Le cadre classique que nous présentons ici est basé sur les hypothèses restrictives suivantes :

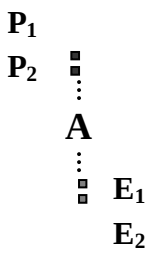
- L'univers est statique : aucun événement imprévu ne peut survenir. Les seuls changements du monde sont ceux effectués par l'agent, qui évolue seul.
- L'agent est omniscient : il possède une connaissance complète de l'univers dans lequel il évolue, et de la nature de ses propres actions.
- Les actions sont atomiques : elles ont un effet immédiat et leur exécution n'est pas interruptible. Elles sont modélisées comme des transitions atomiques d'un état du monde vers un autre.
- Les actions ont des effets déterministes : l'effet de l'exécution d'une action de l'agent est seulement fonction de cette action et de l'état du monde.

Dans ce cadre, le processus de planification nécessite :

- La représentation du domaine et du problème à résoudre (entrée de l'algorithme). Cette représentation est elle même composée par la description :
  - de *l'état initial* de l'univers ;
  - du *but* à atteindre ;
  - des différentes opérations qu'il est possible de réaliser dans le monde : les *opérateurs*.
- Une procédure formalisée (algorithme) qui manipule ces descriptions pour produire un *plan-solution* (ensemble d'actions complètement ou partiellement ordonnées et instanciées, et contraintes temporellement).
- Si possible, des heuristiques, connaissances indépendantes du domaine ou du problème posé [Pearl, 1985] qui permettent de guider la recherche d'un plan-solution de façon à être plus efficace qu'en procédant de manière aléatoire.

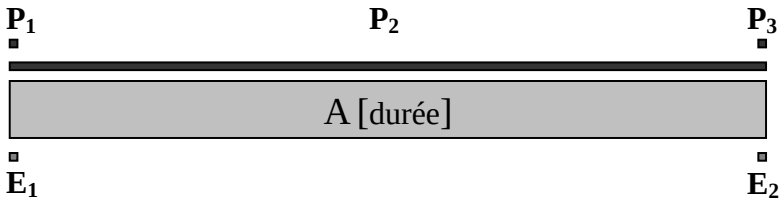
Pour pouvoir représenter les problèmes et les domaines de planification d'une manière indépendante du domaine, plusieurs langages ont été successivement développés. Le planificateur STRIPS (STanford Research Institute Problem Solver) [Fikes, Nilsson, 1971] a été le premier à proposer un tel langage ;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

- Un état du monde est défini comme une conjonction de littéraux propositionnels, encore appelés *fluents*. Les seuls fluents représentés dans un état sont ceux qui sont vrais et on pose également *l'hypothèse du monde clos* : tous les fluents qui ne sont pas vrais dans un état sont considérés comme faux.
- Un état initial est un état, généralement complet, qui décrit l'état initial du monde.
- Un but est un état, généralement partiel qui décrit l'objectif à atteindre. Un état *satisfait* un but si et seulement si 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 ou frame problem), Fikes et Nilsson imposent que chaque fluent non mentionné dans les effets de l'action demeure inchangé par son exécution. Une action A avec deux préconditions P_1 et P_2, et deux effets E_1 et E_2, peut donc être représentée comme une transition instantanée :



2.2. Le cadre temporel

Le cadre classique que nous venons d'évoquer ne permet cependant qu'une représentation limitée des aspects du monde réel et on cherche actuellement à l'étendre progressivement avec des aspects temporels, de gestion de ressources, de l'incertain… tout en conservant de bonnes performances. L'objectif étant de pouvoir, à terme, traiter des problèmes réels. Certains aspects temporels incontournables des actions commencent ainsi à être intégrés au langage PDDL (Planning Domain Description Language), langage qui succède au langage STRIPS. Le cadre temporel que définit actuellement le langage PDDL (version 2.1 et ultérieures) relâche progressivement certaines contraintes du cadre classique. On y suppose qu'à chaque action peut être associée une durée d'exécution fixée. Les préconditions pourront avoir à être satisfaites au début de l'action (at start), tout au long de la durée de l'action (over all), ou à la fin de l'action (at end). Les effets pourront seulement apparaître au début ou à la fin de l'action. Cette prise en compte du temps demeure encore cependant fortement restrictive et limite, comme nous le montrerons par la suite, la recherche de plans-solutions à certains types de problèmes temporels bien précis. Dans ce cadre, une action peut être représentée de la manière suivante :



Lors de la compétition IPC'2008, les planificateurs temporels actuels permettaient de résoudre rapidement (moins de 30 minutes) des problèmes de taille importante. Plusieurs planificateurs ont ainsi réussi à résoudre des problèmes temporels nécessitant plus de 100 actions pour leur résolution. Le planificateur SGPLAN6 [Chen, Wah, Hsu, 2006], vainqueur de cette compétition, trouve en à peine plus de deux secondes un plan-solution contenant plus de 200 actions pour le plus gros problème du domaine "Elevators" qui modélise des déplacements plus ou moins rapides d'ascenseurs emportant des passagers.

2.3. La planification par compilation

Depuis 1992, une nouvelle approche algorithmique de la planification, dite "par compilation", est apparue avec le planificateur SATPLAN [Kautz, Selman, 1992] qui transforme un problème de planification exprimé en langage STRIPS en le codant en une base de clauses dont les modèles, correspondant aux plans-solutions, peuvent être extraits à l'aide d'un prouveur SAT. De nombreux perfectionnements ont ensuite été apportés à cette approche originale, en particulier par la mise au point de codages plus compacts et efficaces dans les planificateurs BLACKBOX [Kautz, Selman, 1998.a], SATPLAN'04 [Kautz et al., 2004], Plan-A [Lv, Chen, Huang, 2008]… A la suite des travaux de [Kautz, Selman, 1992], de nombreuses autres techniques de codage des problèmes de planification ont été développées : problèmes de satisfaction de contraintes (CSP), programmation linéaire (LP)…

Les approches par compilation possèdent l'avantage de permettre de concevoir des planificateurs qui bénéficient directement des améliorations apportées régulièrement aux solveurs pour la résolution des problèmes SAT, CSP… L'exemple le plus évident est celui du

planificateur BLACKBOX (puis de ses clônes, SATPLAN'04 et SATPLAN'06) qui avait déjà participé aux premières compétitions IPC'1998 et IPC'2000 sans pour autant être classé en tête. Grâce aux progrès des solveurs SAT, et sans avoir donné lieu à aucune amélioration notable, il a ensuite remporté les compétitions IPC'2004 et IPC'2006 dans la catégorie "planificateurs optimaux".

La quasi-totalité des approches par compilation envisage la recherche d'un plan-solution de longueur k fixée, longueur qui est augmentée en cas d'échec avant de relancer la recherche d'une solution. Dans le cadre classique, si la complexité de la recherche d'une solution à un problème quelconque est PSPACE-complète, la recherche d'une solution de taille fixée devient NP-complète. Dans le cadre temporel, la complexité est EXPSPACE-complète dans le cas général, mais elle est PSPACE-complète dans le cas d'une recherche de solution de taille fixée [Rintanen, 2007.a], [Bylander, 1994].

3. Présentation de la thèse

Dans cette thèse, nous nous sommes plus particulièrement intéressés à la planification par compilation, d'abord dans le cadre classique avec la planification SAT, puis dans le cadre temporel et nous proposons plusieurs améliorations aux techniques existantes.

Dans la partie II, nous commençons par une présentation rapide des algorithmes essentiels qui sont utilisé dans le cadre classique et qui vont nous servir dans la suite du manuscrit. Nous présentons également les éléments essentiels du langage PDDL.

Dans la partie suivante (cf. partie III), nous nous sommes d'abord placés dans le cadre classique pour étudier les codages SAT existants (cf. chapitre 2) et montrer comment utiliser la relation d'autorisation [Dimopoulos, Nebel, Koehler, 1997], [Vidal, 2001], [Cayrol, Régnier, Vidal, 2001] pour améliorer leurs performances (cf. chapitre 3). Pour cela, nous avons été amené à définir un langage de description des codages (langage TSPL) et à implémenter le planificateur TSP. Celui-ci prend en entrée différents codages et des problèmes de planification pour les transformer en une base de clauses et lui chercher ensuite un modèle (cf. chapitre 4). Le système TSP nous a ainsi permis de comparer équitablement les performances de différents codages avec les notres. Nous avons clairement mis en évidence les gains apportés par l'introduction de la relation d'autorisation (cf. chapitre 5).

Dans la partie IV, nous nous sommes ensuite intéressés à la planification dans le cadre temporel. Nous commençons par un état de l'art des différentes techniques utilisées dans ce cadre et nous étudions l'expressivité des planificateurs temporels les plus importants (cf. chapitre 2). Dans le chapitre 3, nous nous sommes ensuite focalisés sur les algorithmes susceptibles de permettre la résolution de problèmes pour lesquels toutes les solutions nécessitent des actions concurrentes (problèmes temporellement expressifs). Pour cela, nous avons dans un premier temps, proposé plusieurs extensions à l'expressivité temporelle du langage PDDL 2.1 (cf. section 3.2). Nous pouvons ainsi prendre en compte des préconditions qui peuvent être requises et des effets qui peuvent apparaître sur n'importe quel intervalle temporel relatif à l'instant de début ou de fin d'une action. Il nous est aussi possible de traiter des événements exogènes, des buts temporellement étendus, des fenêtres d'activation… Nous avons ensuite proposé deux algorithmes basés sur un graphe de planification simplifié et sur l'utilisation de solveurs DTP (Disjunctive Temporal Problem) ou SMT (Sat Modulo Theory) (cf. section 3.3). Ces algorithmes, implémentés dans le planificateur TLP-GP, permettent de

résoudre des problèmes temporellement expressifs alors que les planificateurs primés aux compétitions IPC n'en sont pas capables. Nous avons ensuite proposé une extension supplémentaire de l'expressivité du langage de représentation de TLP-GP en introduisant, sur les intervalles temporels, des modalités qui permettent une certaine prise en compte de l'incertitude, du choix, ou des transitions continues (cf. section 3.5). Pour évaluer nos algorithmes, nous avons enfin mis au point un ensemble de nouveaux benchmarks temporellement expressifs (cf. section 3.6). Notre étude expérimentale montre que nos algorithmes permettent, en pratique, de résoudre ce type de problèmes et qu'ils montrent de bonnes performances comparés aux planificateurs de l'état de l'art (cf. section 3.7).

II. Algorithmes de planification dans le cadre classique

Depuis une trentaine d'années, la majorité des études du domaine de la planification se situe dans un cadre classique restrictif. L'objectif essentiel consiste à chercher à développer des algorithmes indépendants du domaine qui soient performants pour des représentations simples de l'action. Ce travail doit ensuite pouvoir être étendu en enrichissant cette représentation avec des aspects temporels, de gestion de ressources, de l'incertain… tout en conservant une efficacité suffisante, l'objectif étant, à terme, de pouvoir traiter des problèmes réels.

Dans cette partie, nous présentons rapidement les algorithmes essentiels qui sont utilisés dans le cadre classique ainsi que les principaux éléments du langage PDDL qui permet de formaliser les problèmes de planification. Seuls sont exposés les éléments indispensables à la compréhension de la suite du manuscrit.

1. Définitions

Nous utilisons une logique du premier ordre L, construite à partir des vocabulaires Vx, Vc, Vp qui dénotent respectivement des ensembles finis disjoints deux à deux de symboles de variables, de constantes et de prédicats. Nous n'utilisons pas de fonctions.

Définition 1 (opérateur)

Un *opérateur*, dénoté par o, est un triplet $<pr, ad, de>$ où pr, ad et de dénotent des ensembles finis de formules atomiques du langage L. Prec(o), Add(o), Del(o) dénotent respectivement les ensembles pr, ad et de de l'opérateur o.

Définition 2 (état, fluent)

Un *état* est un ensemble fini de formules atomiques de base (sans symbole de variable) du langage L. Une formule atomique de base est aussi appelée un *fluent*.

Définition 3 (action)

Une *action* dénotée par a est une instance de base $o\theta = <pr\theta, ad\theta, de\theta>$ d'un opérateur o qui est obtenue par l'application d'une substitution θ définie dans le langage L telle que $ad\theta$ et $de\theta$ sont des ensembles disjoints. Prec(a), Add(a) et Del(a) dénotent respectivement les ensembles $pr\theta$, $ad\theta$, $de\theta$ et représentent les préconditions, ajouts et retraits de a.

Pour un ensemble d'actions $A = \{a_1, \ldots, a_n\}$, nous utiliserons les notations :

- $\text{Prec}(A) = \text{Prec}(a_1) \cup \ldots \cup \text{Prec}(a_n)$
- $\text{Add}(A) = \text{Add}(a_1) \cup \ldots \cup \text{Add}(a_n)$
- $\text{Del}(A) = \text{Del}(a_1) \cup \ldots \cup \text{Del}(a_n)$

Définition 4 (problème de planification)

Un *problème de planification* Π est un triplet $<O, I, G>$ où :

- O dénote un ensemble fini d'opérateurs construits à partir du langage L,
- I dénote un ensemble fini de fluents construits à partir du langage L qui représentent l'état initial du problème,
- G dénote un ensemble fini de fluents construits à partir du langage L qui représentent les buts du problème.

En partant de cette représentation des états et des actions, le processus de planification va essayer de synthétiser une collection organisée d'actions appelée plan. Lorsque l'exécution des actions de ce plan permettra de faire passer la représentation du monde de l'état initial à un état satisfaisant le but à atteindre, le plan obtenu sera appelé plan-solution. Suivant les algorithmes utilisés, la forme prise par ces plans-solutions pourra différer et nous les définirons précisément par la suite pour chacun des algorithmes que nous considèrerons.

Les propriétés essentielles que l'on peut attendre des algorithmes de planification sont les suivantes :

Définition 5 (planificateur sain)

Un planificateur est *sain pour un langage L* si, étant donné un problème de planification exprimable dans L, tous les plans qu'il est susceptible de produire pour résoudre ce problème sont des plans-solutions.

Définition 6 (planificateur complet)

Un planificateur est *complet pour un langage L*, si pour tout problème soluble exprimable dans L, le planificateur produit un plan-solution lorsqu'il existe.

Définition 7 (planificateur systématique)

Un planificateur est systématique si et seulement si deux feuilles quelconques de l'arbre de recherche correspondent toujours à des plans-solutions différents.

Définition 8 (planificateur optimal)

Un planificateur sera dit *optimal* suivant un critère particulier qui mesure la qualité des plans si et seulement si les plans-solutions qu'il produit sont optimaux selon ce critère. Ce dernier pourra être, par exemple : le nombre d'actions, le nombre de niveaux, le degré de parallélisme, le temps d'exécution (makespan), la quantité de ressources consommées…

Nous allons maintenant présenter brièvement l'évolution du langage commun de représentation de problèmes PDDL au cours des dix dernières années.

2. Le langage PDDL

Pour comparer les performances des différentes techniques et algorithmes de planification, des compétitions de planificateurs ont été associées aux conférences AIPS puis ICAPS. La nécessité de définir un langage de description commun et plus élaboré que le langage STRIPS a progressivement mené au langage PDDL et à ses différentes versions. Nous présentons dans la Figure 1 l'évolution des différentes catégories de planificateurs et celle de l'expressivité du langage de représentation (les nouveautés essentielles sont en gras).

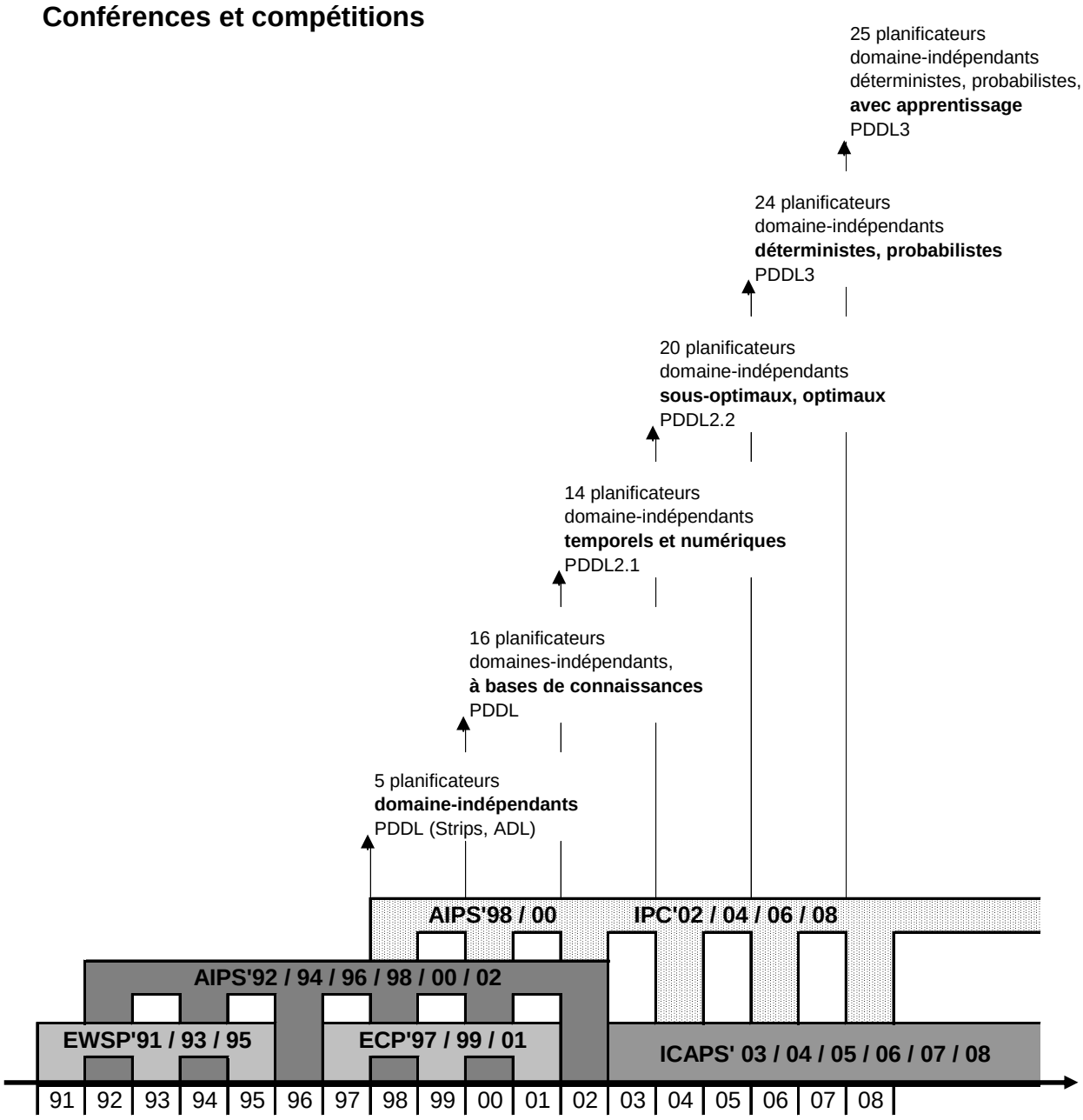


Figure 1 : conférences et compétitions

La version initiale du langage STRIPS permet de représenter un certain nombre de problèmes, mais sa puissance d'expression ne permet pas de modéliser des problèmes réels. Pour cette raison, l'expressivité du langage été progressivement étendue dans le langage ADL [Pednault, 1989] et ensuite dans le langage PDDL [McDermott, 1998] puis PDDL 2.1 [Fox, Long, 2003], en lui ajoutant :

- le typage qui permet de restreindre l'instanciation des variables des opérateurs aux seuls types associés,
- les contraintes d'égalité qui forcent certaines variables des opérateurs à prendre la même valeur ou des valeurs différentes,
- des négations dans les préconditions : un fluent nié en précondition doit être absent de l'état du monde pour que l'action puisse lui être appliquée,
- des préconditions disjonctives : l'action peut être appliquée à un état si au moins un des fluents de la disjonction est présent dans cet état,
- des effets conditionnels : lors de l'application de l'action, des effets supplémentaires peuvent apparaitre si une condition donnée est remplie par l'état avant l'application de l'action,
- des quantificateurs : ils permettent d'agir sur un nombre indéterminé de fluents possédant une propriété commune.

La plupart des ajouts que nous venons d'énumérer ne font qu'améliorer l'expressivité du langage mais pas sa puissance de représentation. Les mêmes domaines peuvent seulement être représentés de manière plus concise et avec une plus grande clarté dans leur expression. Pour être vraiment utiles, et pouvoir traiter des problèmes réels, les planificateurs doivent pouvoir prendre en compte le temps, les ressources, l'incertitude… Les dernières versions du langage, PDDL 2.1, PDDL+ et PDDL 3, créées à l'occasion des compétitions IPC'2002 et IPC'2004 ont pour objectif d'encourager les chercheurs à avancer dans cette voie. Ces versions constituent une réelle évolution avec la possibilité d'exprimer des contraintes temporelles et numériques dans la définition des domaines et des problèmes.

Nous donnons maintenant, comme exemple d'utilisation du langage PDDL, l'un des domaines de la compétition IPC'2002. Le domaine "Depots" a été conçu en combinant les anciens domaines "Logistics" et "Blocks World". Dans ce domaine, des camions transportent des caisses qui doivent, une fois arrivées à destination, être empilées sur des palettes. Les empilements sont réalisés avec des palans, ce qui équivaut aux problèmes de "Blocks World" avec pince.

Exemple d'action codée en PDDL (*Drive*) :

```
(:action Drive
 :parameters (?x - truck ?y - place ?z - place)
 :precondition (and (at ?x ?y))
 :effect (and (not (at ?x ?y)) (at ?x ?z)))
```

Les paramètres de l'action *Drive* sont ?x du type *truck* (?x - truck), ?y du type *place* (?y - place) et ?z du type *place* (?z - place). Pour pouvoir démarrer l'action il est nécessaire (précondition) que le camion ?x se trouve en ?y (at ?x ?y). Après l'application de cette action (effets), le camion ?x ne se trouve plus en ?y (not (at ?x ?y)), mais en ?z (at ?x ?z).

De nombreuses extensions sont possibles (numérique, temporelle). Par exemple, la même action du domaine "Depots" sera codée en PDDL2.1 dans le cadre temporel par :

```
(:durative-action Drive
 :parameters (?x - truck ?y - place ?z - place)
 :duration (= ?duration (/ (distance ?y ?z) (speed ?x)))
 :condition (and (at start (at ?x ?y)))
 :effect (and (at start (not (at ?x ?y))) (at end (at ?x ?z))))
```

Après avoir donné un rapide aperçu du langage commun de représentation des problèmes utilisé par la majorité des planificateurs actuels, nous allons maintenant présenter les algorithmes essentiels qui sont utilisés dans le cadre classique.

3. Recherche dans les espaces d'états

Une des premières approches utilisée aux débuts des recherches en planification avec le planificateur STRIPS [Fikes, Nilsson, 1971] fut la recherche dans les espaces d'états. Partant d'un état initial, d'un ensemble d'actions qui représentent les transitions possibles entre les états, et d'un but qui caractérise l'ensemble des états que l'on désire atteindre, on recherche une séquence d'actions dont l'application à partir de l'état initial permettra d'atteindre un de ces états-buts.

3.1. Définitions

Voici tout d'abord quelques définitions préliminaires indispensables à la compréhension de la suite.

Définition 9 (application d'une action à un état)

L'application d'une action a à un état E (notée $E\uparrow a$) est possible si et seulement si (ssi) $Prec(a) \subseteq E$. L'état résultant E' se calcule en retirant Del(a) et rajoutant Add(a) à l'état initial : $E' = (E - Del(a)) \cup Add(a)$.

Définition 10 (régression d'un état par une action)

La *régression de l'état* B (état partiel en général) par l'action a (notée $B' = B\downarrow a$) est possible si et seulement si (ssi) :

- a est relevante (i.e. utile pour la production d'un fluent du but) : $Add(a) \cap B \neq \emptyset$, et
- a est consistante avec B (i.e. ne détruit pas de fluent du but) : $Del(a) \cap B = \emptyset$,

L'état B' (état partiel en général) qui résulte de cette régression est l'ensemble de fluents calculé en retirant Add(a) à B et en lui ajoutant Prec(a) : $B\downarrow a = (B - Add(a)) \cup Prec(a)$.

Définition 11 (plan séquentiel, espaces d'états)

Un *plan séquentiel*, dénoté par P, est une séquence finie (éventuellement vide) d'actions notée $<a_1, a_2,\ldots, a_n>$ (notée <> si P est la séquence vide).

Définition 12 (application d'une séquence d'actions à un état)

L'application d'une séquence d'actions S = {a_1, …, a_n} à un état E (notée E↑S) est possible si et seulement si (ssi) l'état résultant E' = ((…((E↑a_1)↑a_2)…) ↑a_n) est défini.

Définition 13 (plan-solution séquentiel, correct, espaces d'états)

Un plan séquentiel P est un plan-solution au problème de planification Π = <O, I, B> si et seulement si B ⊆ E' = ((…((E↑a1)↑a2)…) ↑an). Un plan-solution P est encore appelé plan correct.

Définition 14 (plan-solution séquentiel minimal en nombre d'actions)

Un plan séquentiel P = <a_1, a_2,…, a_n> est un *plan-solution séquentiel minimal en nombre d'actions* pour le problème de planification Π = <O, I, B> si et seulement si ∀ P' = <a'$_1$, a'$_2$,…, a'$_m$>, plan-solution au problème Π, alors n≤m.

3.2. Algorithmique

Tous les algorithmes de recherche (heuristique ou brute) dans les espaces d'états peuvent être utilisés : profondeur d'abord, largeur d'abord, A*, WA*… Les deux principales techniques de recherche sont les suivantes :

- La recherche par chaînage avant consiste, partant de l'état initial et d'une séquence d'actions vide, à l'augmenter progressivement par insertion de nouvelles actions en queue. Une action peut être utilisée lorsque ses préconditions sont incluses dans l'état courant ; elle est alors ajoutée à la séquence et le nouvel état courant est calculé en appliquant cette action à l'état précédent. Le choix de l'action est un point de retour arrière de l'algorithme. On a un plan solution lorsque l'état courant contient le but. Cette technique permet naturellement d'utiliser des heuristiques (dépendantes ou non du domaine) car elles s'expriment plus facilement en chaînage avant.
- La recherche par chaînage arrière consiste, partant du but (état partiellement défini) et d'une séquence d'actions vide, à l'augmenter progressivement par insertion de nouvelles actions en tête. Une action peut être employée lorsque l'on peut faire régresser l'état courant par son utilisation : l'un de ses ajouts doit être présent dans l'état courant, et aucun de ses retraits n'est inconsistant avec lui. Partant d'un état courant partiellement défini on obtient le nouvel état courant (lui aussi partiellement défini) en lui appliquant les effets inverses de l'action. Une difficulté pour la mise en œuvre de cette technique est que la régression peut mener à des états qui ne sont pas atteignables (à partir de l'état initial). Le choix de l'action est un point de retour arrière de l'algorithme. On a un plan solution lorsque l'état courant est inclus dans l'état initial. L'avantage de cette méthode est qu'elle est guidée par le but, ce qui réduit généralement le facteur de branchement. Par contre, l'utilisation d'heuristiques est plus difficile à mettre en œuvre.

La recherche dans les espaces d'états a été longtemps délaissée car elle était considérée comme peu efficace. A partir de 1997, plusieurs travaux, initiés par [Bonnet, Loerincs, Geffner, 1997], [Bonnet, Geffner, 1998], montrèrent cependant que cette méthode pouvait s'avérer plus performante que la recherche dans les espaces de plans partiels qui était jusque là privilégiée et que nous présentons dans la section suivante. L'utilisation d'heuristiques très informatives et des progrès sur l'algorithmique de recherche ont maintenant hissé la technique

de recherche dans les espaces d'états au premier rang en terme de taille de problèmes traités et de temps de calcul.

4. Recherche dans les espaces de plans partiels

[Chapman, 1987] a été le premier à proposer une formalisation de l'algorithmique de la planification comme une recherche dans un espace de plans partiels. Dans son planificateur TWEAK, chaque nœud de l'arbre de recherche représente un plan en voie d'élaboration. Le plan initial contient uniquement deux actions factices : une action sans préconditions produisant les fluents de l'état initial du problème et une action sans effets dont les préconditions sont les fluents de l'état but du problème. Dans l'espace de recherche, on passe d'un nœud (représentant un plan partiel) à un autre (représentant un plan partiel plus élaboré) en utilisant des opérateurs de modification de plans. Lorsqu'un nœud contient un plan dont toutes les actions ont leurs préconditions produites par d'autres actions du plan, on a un plan solution. La formalisation de l'algorithmique de recherche de TWEAK est basée sur un *critère de vérité* qui énumère les différentes opérations de modification de plan qui peuvent affecter la valeur de vérité des fluents du plan courant. Les techniques utilisées pour cela sont l'insertion d'actions dans le plan, de contraintes d'ordre entre ces actions, de contraintes entre les variables d'actions partiellement instanciées… L'interprétation du *critère de vérité* comme une procédure non déterministe donne un algorithme qui, en essayant toutes ces opérations dans un ordre défini par une technique de recherche (par exemple la recherche en profondeur d'abord), permet de construire progressivement un plan solution.

4.1. Définitions

Voici quelques définitions préliminaires indispensables à la compréhension de la suite.

Définition 15 (plan partiel)

Un plan partiel est composé de trois ensembles :

- L'ensemble des *étapes* : ce sont des références aux actions qui constituent le plan partiel. Chaque étape est identifiée de façon unique ; un plan partiel peut donc contenir plusieurs étapes qui font référence à une même action.
- L'ensemble des *contraintes d'ordre* : ce sont des contraintes précisant l'ordonnancement de deux étapes distinctes. Elles peuvent être de deux types :
  - *contraintes de précédence* : le fait que l'étape e1 précède l'étape e2 signifie que e1 doit être exécutée avant e2, mais toute étape e3 peut être insérée entre e1 et e2.
  - *contraintes de contiguïté* : le fait que l'étape e1 soit contiguë à l'étape e2 signifie que e1 doit être exécutée immédiatement avant e2, interdisant l'insertion de toute étape e3 entre e1 et e2.
- *L'ensemble des contraintes auxiliaires* : ce sont des conditions qui doivent être vérifiées par un plan partiel. Elles peuvent être de deux types :
  - contraintes de préservation d'intervalles : elles préservent la valeur de vérité d'un fluent sur un intervalle de précédence, c'est-à-dire que les étapes qui vont s'insérer entre les deux étapes de l'intervalle de précédence ne doivent pas avoir pour effet la négation de ce fluent. Ainsi, si une étape e_1 est insérée dans le plan pour établir une

précondition *p* d'une étape e_2, alors on pourra définir une contrainte de précédence entre e_1 et e_2 et protéger la valeur du fluent *p* par une contrainte de préservation de cet intervalle.

- *contraintes ponctuelles de valeur de vérité* : elles protègent la valeur de vérité d'un fluent à un endroit précis du plan partiel.

Définition 16 (parties d'un plan partiel)

Les différentes parties d'un plan partiel sont :

- La *tête du plan* : il s'agit de l'ensemble maximal d'étapes du début du plan liées deux à deux par des contraintes de contiguïté. La dernière étape de la tête du plan est l'*étape de tête*. Comme ces étapes forment une séquence d'actions applicables dans l'état initial du problème, on peut calculer l'état résultant de son application que l'on nommera l'*état de tête*.
- La *queue du plan* : c'est la partie symétrique de la tête du plan pour la fin du plan partiel. La première étape de la queue du plan est l'*étape de queue*, et l'*état de queue* est l'état partiel défini par la régression des étapes de la queue du plan sur les buts.
- Les *étapes centrales* : ce sont toutes les étapes n'appartenant ni à la tête ni à la queue du plan. La *frange de tête* est l'ensemble des étapes pouvant venir directement après l'étape de tête dans une linéarisation, et la *frange de queue* est l'ensemble des étapes pouvant précéder directement l'étape de queue dans une linéarisation.

Définition 17 (demandeur)

Un *demandeur (needer)* du fluent p est une action d du plan pour laquelle il faut établir (asserter) une précondition p.

Définition 18 (établisseur)

Un *établisseur (establisher)* de p est une action e permettant d'établir (d'asserter) le fluent p.

Définition 19 (lien causal)

Un *lien causal (causal link)* est un lien de causalité entre un établisseur e de p et un demandeur d de p. On notera ce lien : $e \rightarrow^{p} d$. Un lien causal entre e et d signifie en même temps que l'établisseur e produit une précondition pour le demandeur d et que e précède d.

Définition 20 (casseur, menace)

Un *casseur* c *(clobberer)* est une action qui *menace* le lien causal $e \rightarrow^{p} d$ c'est-à-dire une action qui est susceptible d'ajouter ou de détruire un fluent q qui pourrait être associé à p. On notera cette menace par le couple : $(c, e \rightarrow^{p} d)$.

Définition 21 (menace positive, menace négative)

Une action qui menace le lien e $\rightarrow^p$d en étant susceptible d'ajouter un fluent q qui pourrait être associé à p est appelé une *menace positive* pour d. Une action qui menace le lien e $\rightarrow^p$d en étant susceptible de détruire un fluent q qui pourrait être associé à p est appelé une *menace négative* pour d.

Définition 22 (chevalier blanc)

Un *chevalier blanc (white knight)* de p est une action w qui permet de rétablir le fluent p détruit par un casseur. C'est un nouvel établisseur.

Les différentes opérations qui peuvent être employées pour modifier un plan partiel pendant la recherche d'un plan solution sont les suivantes :

Définition 23 (choix d'un établisseur)

Cette opération consiste à placer avant le demandeur d une action établisseur e pour établir une précondition p de d en créant un lien e$\rightarrow^p$d. Cet établisseur peut être une action déjà présente dans le plan (ancien établisseur) ou, si une telle action n'existe pas, une nouvelle action que l'on insère (nouvel établisseur).

Définition 24 (promotion d'un casseur)

Cette opération consiste à contraindre le casseur c de p à s'exécuter avant l'établisseur e : c < e (l'établissement de la précondition p par e pour d n'est pas remis en cause par c).

Définition 25 (rétrogradation, démotion d'un casseur)

Cette opération consiste à contraindre le casseur c à s'exécuter après le demandeur d : d < c (ainsi c ne détruira pas la précondition p créée par e pour d). Cette opération est encore couramment appelée *démotion*.

Définition 26 (choix d'un white knight)

Cette opération consiste à placer entre le casseur c et le demandeur d une action w pour rétablir la précondition p de d détruite par c : e <c < w < d (ceci revient à choisir w comme nouvel établisseur de p).

Définition 27 (séparation casseur / demandeur)

Cette opération consiste à contraindre l'instanciation de variables du casseur c pour que ce dernier ne détruise pas la précondition p de d par la suite (c ne peut empêcher d'établir la précondition p de d).

Dans ce formalisme [Kambhampati, 1997], on considère qu'un planificateur effectue une recherche dans l'espace des *séquences d'actions* que l'on peut définir sur un domaine, l'objectif étant de trouver une *séquence solution*. Ces séquences d'actions sont représentées de manière compacte par des plans partiels.

4.2. Algorithmique

La recherche est initialisée par l'ajout à un plan partiel vide d'une étape initiale associée à une action sans préconditions produisant les fluents de l'état initial du problème, et d'une étape but associée à une action sans effets dont les préconditions sont les fluents du but du problème. La recherche *dans les espaces de plans* est guidées par les buts et ne contraint les plans partiels que lorsque cela est indispensable. Ces algorithmes de planification implémentent le cycle suivant :

- choisir un nœud (plan partiel) dans l'arbre de recherche courant,
- choisir dans ce nœud un défaut : précondition à asserter ou menace à traiter non encore résolue,
- choisir un opérateur de modification de plan permettant d'asserter cette précondition ou de résoudre cette menace.
- construire alors le plan partiel (nœud) suivant, l'ajouter à l'arbre de recherche et reprendre le cycle…

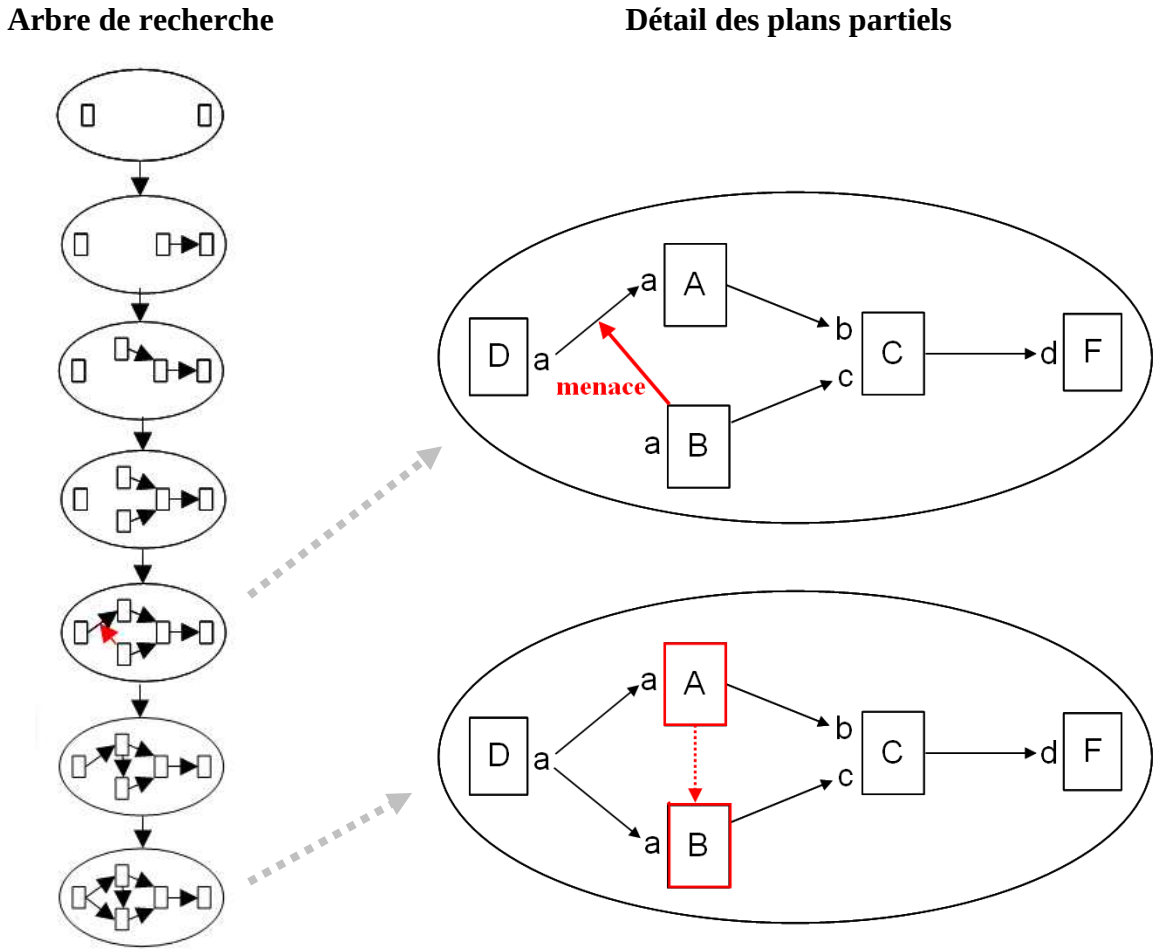


Figure 2 : recherche dans les espaces de plans partiels

Voici un exemple simple qui illustre les aspects essentiels du fonctionnement de la recherche dans les espaces de plans (cf. Figure 2). L'ensemble des propositions est P = {a, b, c, d} et l'ensemble des actions est A = {A, B, C} avec :

Prec(A) = {a}	Prec(B) = {a}	Prec(C) = {b, c}
Add(A) = {b}	Add(B) = {c}	Add(C) = {d}
Del(A) = {}	Del(B) = {a}	Del(C) = {}

L'état initial du problème est I = {a} et le but est B = {d}.

Nous allons maintenant présenter le fonctionnement du planificateur GRAPHPLAN [Blum, Furst, 1995, 1997]. Son apparition, en 1995, fut à l'origine d'un bouleversement des techniques de planification STRIPS.

5. Graphplan

Le planificateur GRAPHPLAN travaille en imbriquant deux processus :

- La construction d'un espace de recherche appelé graphe de planification jusqu'à l'obtention d'un niveau qui contienne le but recherché. A l'aide de la description des opérateurs et des fluents de l'état initial, chaque opérateur est instancié de toutes les façons possibles ; les actions applicables ainsi obtenues sont utilisées pour produire un nouvel ensemble de fluents ainsi que des contraintes sur les fluents et les actions. Cette opération est ensuite répétée à partir de ce nouvel ensemble de fluents, jusqu'à la satisfaction d'une condition nécessaire (mais non suffisante) de validité des buts. Cet espace de recherche est plus compact qu'un espace d'états ou de plans et sa construction est polynomiale en temps et en espace en fonction des données. Une fois le graphe de planification construit, la deuxième phase va chercher à en extraire un plan-solution.
- Partant ensuite du graphe de planification, GRAPHPLAN cherche à en extraire un plan-solution. Cette recherche s'effectue en tenant compte des actions, des fluents et surtout des contraintes entre les actions qui ont été détectées à l'étape d'expansion du graphe. Si aucun plan-solution n'est trouvé et que la condition d'arrêt de l'algorithme n'est pas satisfaite, on continue à développer le graphe de planification ; sinon il n'y a pas de solution au problème posé.

Lors de son apparition, les performances de GRAPHPLAN étaient largement supérieures à celles des autres planificateurs. Les idées qui guidaient sa conception inspirèrent et continuent d'inspirer un grand nombre de travaux. Ceux-ci ont permis aux algorithmes classiques d'augmenter très largement leurs performances et ont donné naissance à de nouvelles méthodes. L'efficacité des méthodes basées sur GRAPHPLAN dépend fortement de la phase d'extraction de la solution.

5.1. Définitions

Dans GRAPHPLAN, les interactions entre actions sont considérées au travers de la notion d'indépendance entre actions. Nous définissons donc maintenant cette notion d'indépendance entre deux actions, puis pour un ensemble d'actions : lorsque cette relation est vérifiée, elle

permet, quel que soit l'ordre d'exécution d'un ensemble d'actions indépendant Q, d'obtenir un état résultant identique.

Définition 28 (indépendance entre deux actions)

Deux actions a et b telles que $a \neq b$ sont *indépendantes* (noté a # b) ssi :

$$(Add(a) \cup Prec(a)) \cap Del(b) = \emptyset \text{ et } (Add(b) \cup Prec(b)) \cap Del(a) = \emptyset$$

Définition 29 (ensemble d'actions indépendant)

Un ensemble d'actions $Q = \{a_1, \ldots, a_n\}$, est indépendant ssi les actions qui le composent sont indépendantes deux à deux, c'est à dire :

$$(Q \text{ indépendant}) \Leftrightarrow (\forall \{a, b\} \subseteq Q, a \neq b, (Prec(a) \cup Add(a)) \cap Del(b) = \emptyset)$$

L'état résultant de l'application de Q à un état E (notée $E \uparrow Q$) se calcule de manière simple en rajoutant Add(Q) à l'état E et en lui retirant Del(Q).

Exemple :

Application d'un plan séquentiel en trois étapes, chacune correspondant à une seule action :

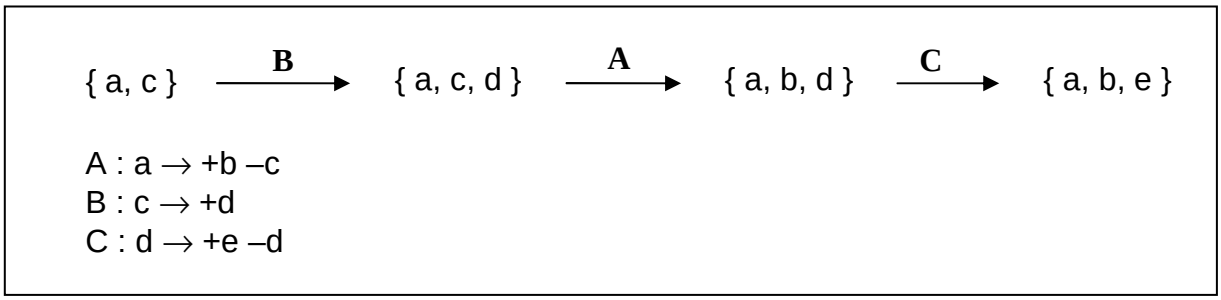


Figure 3 : plan séquentiel

Application d'un plan parallèle en deux étapes en utilisant la relation d'indépendance :

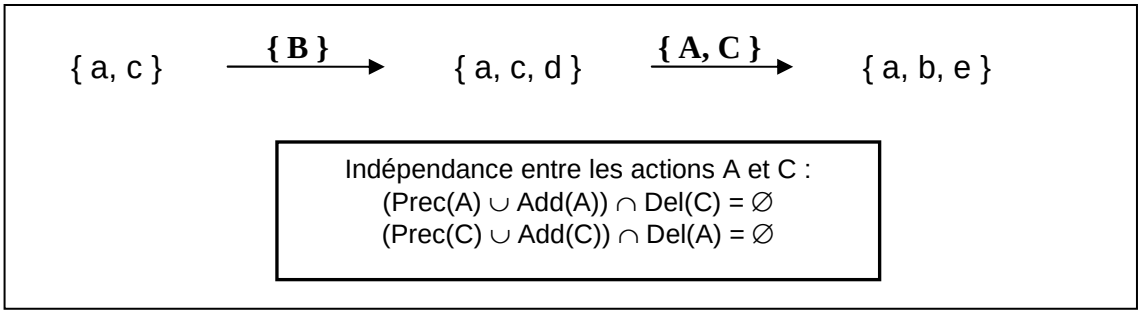


Figure 4 : plan parallèle (indépendance entre deux actions)

5.2. Algorithmique

Dans les espaces d'états, les noeuds du graphe de recherche représentent les états successifs de l'univers du problème et les arcs, les actions qui permettent de passer d'un état à un autre. L'algorithme tente de construire un chemin (plan solution) qui permette de passer de l'état initial du problème à un état-but. Il se termine quand l'état courant contient les buts à atteindre. Dans les espaces de plans partiels, les noeuds du graphe de recherche représentent des plans partiels et les arcs, les opérations de raffinement de ces plans partiels. L'algorithme cherche à raffiner un plan initial qui représente le problème posé, pour obtenir un plan solution. Il se termine donc lorsque toutes les préconditions de toutes les actions présentes dans le plan courant sont assertées par d'autres actions du plan.

GRAPHPLAN travaille différemment en développant dans un premier temps, niveau par niveau, un espace de recherche plus compact appelé graphe de planification. De manière informelle, un graphe de planification est un graphe constitué de niveaux successifs repérés par des entiers naturels, chaque niveau étant composé d'un ensemble d'actions et d'un ensemble de propositions. Le niveau 0 fait exception et ne contient que des propositions qui représentent les fluents de l'état initial.

Pour la phase de construction, GRAPHPLAN n'utilise pas toutes les informations indispensables à l'obtention d'un plan solution qui sont prises en compte au fur et à mesure par les autres approches (exclusions mutuelles entre variables d'état ou entre actions). Ces contraintes sont simplement calculées et enregistrées à chaque niveau du graphe sous forme d'exclusions mutuelles à la manière des CSP [Kambhampati, 2000]. Deux actions sont *indépendantes* (cf. Définition 28, page 20) lorsque leurs effets ne se contredisent pas (une action ne doit pas retirer une proposition ajoutée par l'autre action) et qu'elles n'ont pas d'interactions croisées (une action ne doit pas retirer une proposition qui est une précondition de l'autre). Un ensemble d'actions *indépendant* (cf. Définition 29, page 20) représente un ensemble d'actions indépendantes deux à deux. Pour que ces actions soient exécutables en parallèle, l'algorithme de GRAPHPLAN détecte et exploite aussi le fait que deux actions d'un même ensemble ne doivent pas avoir de préconditions incompatibles. L'espace de recherche se développe donc plus facilement, mais, en contrepartie, son achèvement ne coïncide plus avec l'obtention d'un plan-solution qu'une deuxième phase (dite phase d'extraction) cherche alors à extraire à partir du graphe de planification et des contraintes enregistrées.

Nous reprenons maintenant l'exemple simple que nous avons introduit au chapitre précédent pour illustrer les aspects essentiels du fonctionnement de ce planificateur. Le graphe construit par GRAPHPLAN est donné Figure 5. Les actions A et B sont mutuellement exclusives car B retire a qui est une précondition de A. Au niveau 1, les paires de préconditions qui sont mutuellement exclusives sont donc {a, c} et {b, c}. On ne peut donc pas utiliser l'action C (qui donne le but d) au niveau 2. GRAPHPLAN poursuit donc la construction du graphe : au niveau 2, b et c ne sont plus mutuellement exclusives. L'action C peut donc maintenant être appliquée au niveau 3 où elle produit le but d.

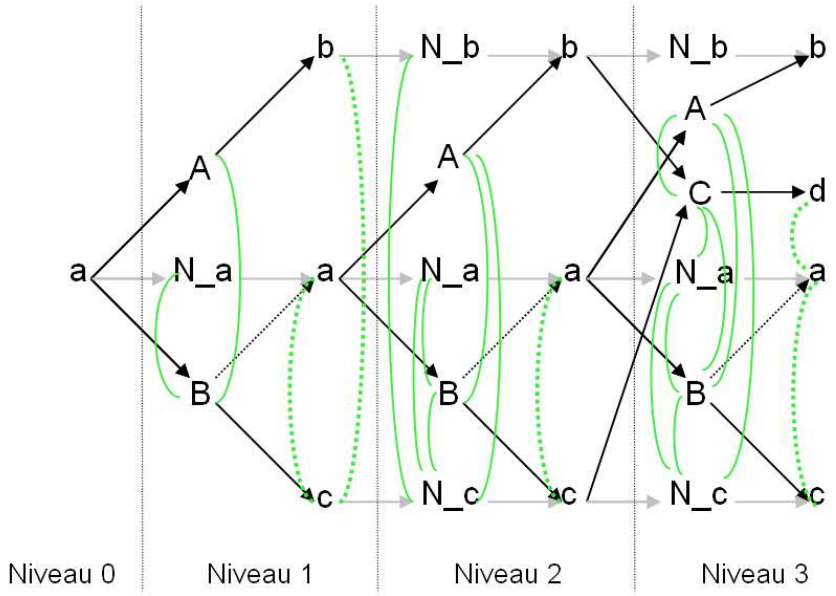


Figure 5 : Graphe de planification de l'exemple (expansion)

Le but étant maintenant potentiellement atteignable, GRAPHPLAN va chercher, dans une deuxième phase, à extraire un plan-solution du graphe de planification (cf. Figure 6). Pour cela, il part de l'ensemble des propositions construit au dernier niveau (et qui contient donc les buts) et considère les différents ensembles d'actions qui permettent d'établir ces buts. Il choisit l'un de ces ensembles (point de backtrack de l'algorithme) et, en passant au niveau précédent, cherche à nouveau les différents ensembles d'actions qui permettent d'établir l'ensemble des préconditions des actions précédemment choisies... A chaque niveau, les actions d'un ensemble doivent être indépendantes deux à deux et leurs préconditions ne doivent pas être contradictoires. Pour cela, GRAPHPLAN vérifie simplement, en utilisant les exclusions mutuelles enregistrées pendant la contruction, qu'il n'existe aucune paire d'actions mutuellement exclusives. Dans notre exemple, le plan-solution obtenu sera <A, B, C>. L'exemple étant simple, il est obtenu sans retour arrière.

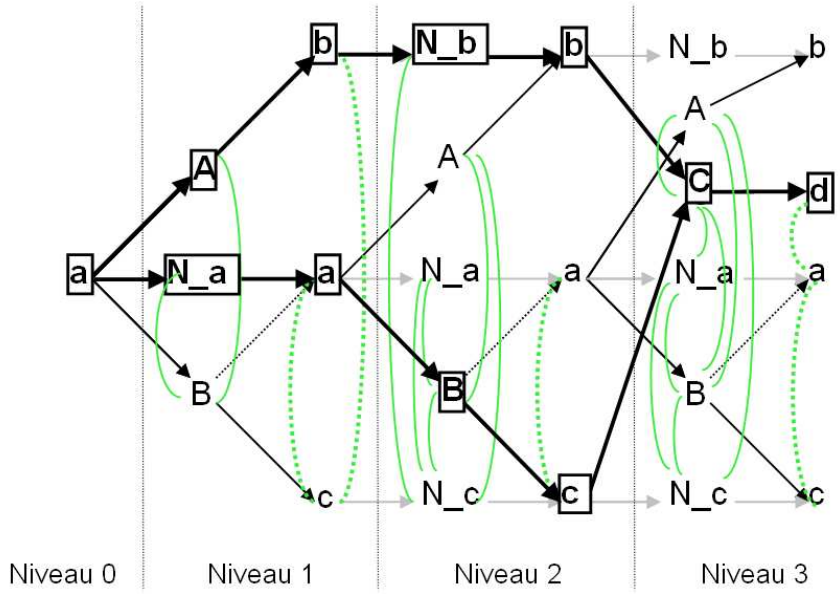


Figure 6 : Graphe de planification de l'exemple (extraction de solution)

6. Conclusion

Dans ce chapitre, nous avons rapidement passé en revue les principaux algorithmes utilisés dans le cadre classique qui sont nécessaires pour la compréhension de la suite du manuscrit. Depuis les débuts de la planification, les progrès ont été considérables, que ce soit dans les espaces d'états, dans les espaces de plans partiels, pour GRAPHPLAN ou pour les approches par compilation. Grace aux compétitions IPC, à la généralisation du langage PDDL, ces progrès considérables dans le cadre classique ouvrent maintenant la voie à la prise en compte de nouveaux aspects (temporels, ressources, incertain…) pour la représentation de problèmes plus proches de problèmes réels.

Nous allons maintenant exposer le travail que nous avons réalisé sur la planification SAT dans le cadre classique, puis sur la planification dans un cadre temporel.

III. Planification SAT : le système TSP

1. Introduction

Comme nous l'avons déjà signalé, l'approche SAT de la planification (encore appellée planification par compilation) est apparue en 1992 avec le planificateur SATPLAN [Kautz, Selman, 1992]. Dans cette approche, on travaille directement sur un ensemble fini de variables propositionnelles. Deux actions identiques pouvant apparaître à des endroits différents d'un même plan doivent pouvoir être différenciées et on leur associe donc des propositions différentes. Comme on ne connaît pas à l'avance la longueur d'un plan solution d'un problème, on ne peut pas créer un codage unique permettant de le résoudre puisqu'il faudrait créer une infinité de variables propositionnelles pour représenter toutes les actions de tous les plans possibles. La solution la plus commune (cf. Figure 7, ci-dessous) consiste alors à créer un codage représentant tous les plans d'une longueur k fixée. La base de clause ainsi obtenue est donnée en entrée à un solveur SAT qui retourne, lorsqu'il existe, un modèle de cette base. Le décodage de ce modèle permet alors d'obtenir un plan-solution. Si la résolution du codage ne donne pas de modèle, la valeur de k est augmentée et le processus réitéré. Pour la complétude du procédé, tous ces modèles doivent correspondre exactement à tous les plans solutions d'une longueur fixée du problème.

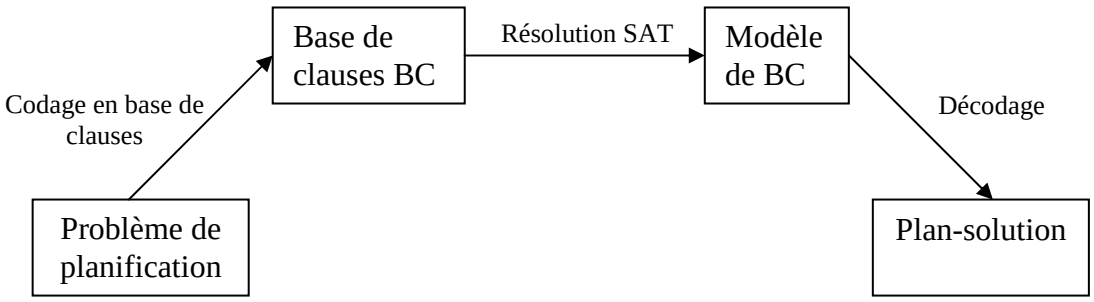


Figure 7 : schéma de principe de l'approche SATPLAN

De nombreux perfectionnements ont été apportés à cette approche, en particulier par BLACKBOX [Kautz, Selman, 1998.a], [Kautz, Selman, 1999] (cf. Figure 8, ci-après) qui encode les actions, fluents, et exclusions mutuelles du graphe de planification de GRAPHPLAN comme un problème SAT pour en extraire une solution. Il permet l'utilisation de la procédure d'extraction de GRAPHPLAN ainsi que celle de plusieurs prouveurs SAT et se montre plus performant que SATPLAN et GRAPHPLAN.

L'avantage de ces techniques de planification par compilation est qu'elles permettent de bénéficier directement des résultats des recherches effectuées dans le domaine de la résolution du problème SAT et des améliorations qui sont régulièrement apportées aux solveurs. Le

planificateur BLACKBOX (renommé SATPLAN’04 puis SATPLAN’06), qui avait déjà participé sans succès aux premières compétitions IPC'1998 et IPC'2000, est ainsi demeuré en tête de la catégorie "planificateurs optimaux" lors des compétitions IPC'2004 et IPC'2006 grâce aux progrès réalisés par les solveurs SAT.

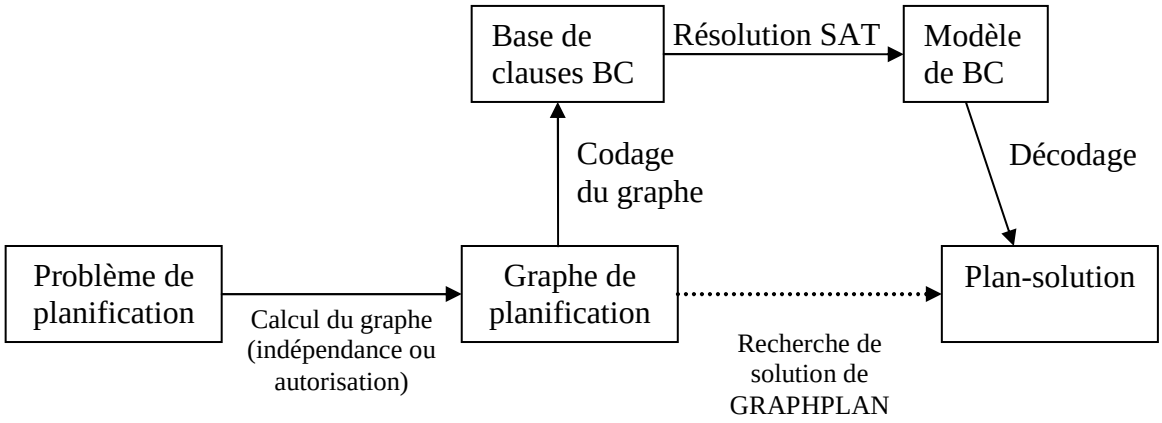


Figure 8 : schéma de principe de l'approche BLACKBOX

Dans cette partie, et sauf mention contraire, nous nous plaçons dans le cadre classique de la planification. Nous commençons, dans le chapitre 2, par dresser un état de l'art des techniques et des codages utilisés par les planificateurs SAT pour encoder les problèmes de planification. Nous présentons ensuite les codages essentiels pour la planification par satisfiabilité proposés dans [Mali, Kambhampati, 1999], [Kautz, Selman, 1999] et les améliorations apportées par [Vidal, 2001] qui produisent des codages plus compacts et permettent le parallélisme dans les codages basés sur les espaces de plans.

Après cet état de l'art, nous détaillons, dans les chapitres suivants, le travail que nous avons effectué pour améliorer ces techniques de planification :

- Dans un premier temps, nous montrons (chapitre 3) comment intégrer aux différents types de codages SAT présentés dans le chapitre 2, la relation d'autorisation [Dimopoulos, Nebel, Koehler, 1997], relation qui avait déjà été utilisée dans [Vidal, 2001], [Cayrol, Régnier, Vidal, 2001] pour améliorer les performances de GRAPHPLAN [Blum, Furst, 1997] et DPPLAN [Baioletti, Marcugini, Milani, 2000].
- Nous présentons ensuite le système TSP (Tunable SatPlan), un planificateur que nous avons développé pour comparer équitablement les performances des différents codages. TSP intègre en effet un traducteur qui utilise un langage que nous avons spécifiquement défini pour représenter les codages (chapitre 4).
- Dans le chapitre 5, nous détaillons les résultats expérimentaux que nous avons obtenus et qui montrent la supériorité des codages intégrant la relation d'autorisation.

2. Etat de l'art de la planification SAT

2.1. Définitions préliminaires

Nous utilisons une logique du premier ordre *L*, construite à partir des vocabulaires *Vx*, *Vc*, *Vp* qui dénotent respectivement des ensembles finis disjoints deux à deux de symboles de variables, de constantes et de prédicats. Nous n'utilisons pas de fonctions. Pour les notations utilisées dans les codages, nous nous basons sur les définitions classiques de logique propositionnelle. Pour deux formules F_1 et F_2 et une propriété G, "si G est satisfaite, alors F_1 sinon F_2" s'écrira en utilisant la règle de réécriture suivante :

$$[G \mapsto F_1 \vdots F_2]$$

Dans la suite nous utiliserons, pour un problème de planification $\Pi = <O, I, G>$ dans lequel :

- *O* dénote un ensemble fini d'opérateurs construits à partir du langage *L*,
- *I* dénote un ensemble fini de fluents construits à partir du langage *L* qui représentent l'état initial du problème,
- *G* dénote un ensemble fini de fluents construits à partir du langage *L* qui représentent les buts du problème.

On définit également les ensembles suivants :

- *Fp* : l'ensemble des fluents apparaissant dans les préconditions des actions (préconditions).
- *Fa* : l'ensemble des fluents qui apparaissent dans les ajouts des actions (ajouts).
- *Fd* : l'ensemble des fluents qui apparaissent dans les retraits des actions (retraits).

Avant de présenter en détail les différents codages sur lesquels nous avons travaillé (sections 2.3, 2.4, 2.5, 2.6), nous allons, dans la section suivante, faire un tour d'horizon rapide des travaux les plus importants qui ont été réalisés dans le domaine de la planification SAT.

2.2. Historique

Les plans-solutions potentiels au problème de planification posé peuvent être représentés par différents codages dont chacun est une manière d'appréhender la structure de ces plans. Il existe ainsi des codages issus du calcul des situations, des codages dans les espaces d'états, dans les espaces de plans, des codages basés sur le graphe de planification de GRAPHPLAN… Les bases de clauses obtenues par le biais de ces codages sont plus ou moins compactes et de nombreux travaux montrent qu'il existe une forte corrélation entre la difficulté du solveur à satisfaire une formule et les nombres de variables et de clauses de cette formule. La phase de codage a donc une incidence considérable sur les temps de réponse du solveur SAT et un grand nombre d'études ont été réalisées pour essayer de rendre ces codages les plus efficaces possibles.

L'article originel de [Kautz, Selman, 1992] montre comment encoder manuellement des problèmes de planification du domaine des cubes sous forme de problèmes SAT. Les codages utilisés sont issus du calcul des situations de [McCarthy, Hayes, 1969] et nécessitent

l'utilisation de « frame-axiomes » qui indiquent, pour chaque action, que les fluents qu'elle n'affecte pas conservent leur valeur de vérité après son application. Le codage « linéaire » 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'opérateurs des plans.

[Kautz, Selman, 1996] montrent ensuite comment réduire la taille des codages en encodant le graphe de planification de GRAPHPLAN [Blum, Furst, 1997] comme un problème SAT pour en extraire un plan-solution. La traduction en base de clauses commence au niveau-but du graphe de planification et se poursuit en arrière. Les règles de codage y sont illustrées avec le problème « rocket » de [Blum, Furst, 1995]. Dans ce problème, un chargement peut être embarqué dans une fusée si le chargement et la fusée sont au même endroit. Une fusée peut se déplacer si elle a fait le plein de combustible et le déplacement consomme la totalité du combustible. LOAD(A,R,L,i) signifie "charger A dans R au lieu L à l'instant i", et MOVE(R,L,P,i) signifie "déplacer R de L à P à l'instant i" et UNLOAD(A,R,L,i) signifie "décharger A de R au lieu L à l'instant i". Les règles de codages utilisées sont les suivantes :

- L'état initial est vérifié au niveau 1 et le but est vérifié au niveau le plus élevé k.

$$1.\quad \left(\bigwedge_{f\in I} f(1)\right)\wedge\left(\bigwedge_{f\in G} f(k)\right)$$

- Les opérateurs impliquent leurs préconditions, par exemple LOAD(A,R,L,2) $\Rightarrow$ (AT(A,L,1) $\wedge$ AT(R,L,1)).

$$2.\quad \bigwedge_{i\in[1,k]}\bigwedge_{a\in O}\left(a(i)\Rightarrow\left(\bigwedge_{f\in \mathrm{Pr}ec(a)} f(i-1)\right)\right)$$

- Chaque fait au niveau i implique la disjonction de tous les opérateurs du niveau i – 1 qui ont ce fait comme effet d'ajout, par exemple IN(A,R,3) $\Rightarrow$ (LOAD(A,R,L,2) $\vee$ LOAD(A,R,P,2) $\vee$ MAINTAIN (IN (A,R),2)).

$$3.\quad \bigwedge_{i\in[1,k]}\bigwedge_{f\in F}\left(f(i)\Rightarrow\bigvee_{a\in O\,/\,f\in Add(a)} a(i-1)\right)$$

- Les action conflictuelles sont mutuellement exclusives, par exemple ¬LOAD(A,R,L,2) $\vee$ ¬MOVE(R,L,P,2).

$$4.\quad \bigwedge_{i\in[1,k]}\ \bigwedge_{\substack{(a_m,a_n)\in O^2\,/\,m<n\\ \left(\left(\mathrm{Pr}ec(a_m)\cup Add(a_m)\cap Del(a_n)\neq\varnothing\right)\\ \vee\left(Del(a_n)\cup Add(a_n)\cap \mathrm{Pr}ec(a_m)\neq\varnothing\right)\right)}} \left(\neg a_m(i)\vee\neg a_n(i)\right)$$

Kautz et Selman développent les codages dans les espaces d'états avec des no-ops et des frame-axiomes explicatifs [Kautz, Selman, 1996]. Ils remarquent que l'utilisation de ces derniers permet le parallélisme, contrairement aux frame-axiomes classiques qui imposent l'utilisation d'une unique action à chaque niveau. Ils comparent leur système SATPLAN aux planificateurs de référence UCPOP [Penberthy, Weld, 1992] et GRAPHPLAN et montrent que l'approche SAT peut être compétitive. Les codages de [Kautz, Selman, 1996] seront

largement repris, étudiés et améliorés, en particulier dans [Ernst, Millstein, Weld, 1997], [Mali, Kambhampati, 1999].

[Kautz, McAllester, Selman, 1996] montrent comment encoder automatiquement des problèmes de planification en problèmes SAT. Ils introduisent l'utilisation du parallélisme, proposent la première version des codages dans les espaces de plans partiels, et comparent la taille de différents codages. Comme les temps de réponse des algorithmes SAT sont exponentiels en fonction de la taille de la formule et que cette dernière dépend linéairement de la longueur du plan, l'efficacité de la planification SAT dépend fortement de la taille des plans. L'introduction du parallélisme permet de réduire la taille des formules, donc d'augmenter les performances.

[Ernst, Millstein, Weld, 1997] étudient l'impact de différentes représentations des actions dans les codages des problèmes de planification. Leur système 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. [Baioletti, Marcugini, Milani, 1998], dans leur système CSATPLAN, encodent le graphe de planification de GRAPHPLAN pour produire une base de clauses pour laquelle des solveurs SAT cherchent ensuite un modèle. [Mali, Kambhampati, 1998.a] améliorent les performances des codages dans les espaces de plans en réduisant le nombre de variables et de clauses nécessaires. [Mali, Kambhampati, 1998.b] 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. [Giunchiglia, Massarotto, Sebastiani, 1998] remarquent que les valeurs de vérités des fluents sont induites par celles des actions. Ils modifient donc le prouveur TABLEAU [Crawford, Auton, 1996] pour que les choix effectués dans la procédure de Davis et Putnam ne portent que sur les actions. Les valeurs des fluents sont alors déduites par une simple propagation unitaire. Les performances du prouveur SAT ainsi obtenu sont comparées avec sa version originale sur plusieurs des codages de [Ernst, Millstein, Weld, 1997]. [Mali, 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, Selman, 1996] et [Ernst, Millstein, Weld, 1997], et démontrent également que les codages les plus compacts sont ceux des espaces d'états.

[Kautz, Selman, 1998.a], [Kautz, Selman, 1999] développent le planificateur BLACKBOX qui encode les actions, fluents, et exclusions mutuelles du graphe de planification de GRAPHPLAN comme un problème SAT pour en extraire une solution. Ce planificateur se montre plus performant que SATPLAN et GRAPHPLAN. BLACKBOX permet l'utilisation de la procédure d'extraction de GRAPHPLAN ainsi que celle de plusieurs prouveurs SAT. Il possède de nombreuses options et permet l'exécution séquentielle de ces prouveurs sur une certaine durée, avec leurs différentes options de fonctionnement.

[Mali, Kambhampati, 1998.c], [Mali, 1999.b] montrent comment encoder la planification de type HTN (Hierarchical Task Network), et donnent la taille des codages ainsi obtenus. [Mali, 2000.a] développe plusieurs heuristiques qui, grâce aux HTN, utilisent des connaissances sur le domaine pour générer des codages souvent plus compacts et plus faciles à résoudre. [Kautz, Selman, 1998.b] montrent comment améliorer les performances de SATPLAN en codant des connaissances spécifiques au domaine de manière déclarative et indépendante du solveur utilisé. Le planificateur TLPLAN [Bacchus, Kabanza, 1995, 2000], efficace sur de nombreux domaines, utilise un langage de représentation des connaissances sous forme déclarative, expressif et basé sur une logique temporelle du premier ordre. Il effectue une recherche en

chaînage avant dans les espaces d'états, en vérifiant, à chaque état, la validité de formules de ce langage qui représentent les connaissances sur le domaine. [Kautz, Selman, 1998.a] montrent que ce type de contrôle d'information est possible dans les approches SAT et améliore beaucoup les performances. [Huang, Selman, Kautz, 1999] montrent que les connaissances que l'on peut représenter dans le langage de TLPLAN peuvent être classées en trois catégories : (1) des connaissances statiques basées sur l'état initial et le but qui servent à éliminer directement des actions avant même de les utiliser dans la base de clauses, (2) des connaissances qui dépendent de l'état courant et peuvent être exprimées par ajout de clauses dans la base sans créer de nouvelles variables, (3) des connaissances qui dépendent de l'état courant et qui nécessitent la création de nouvelles variables. Ils encodent les deux premières catégories dans BLACKBOX pour plusieurs domaines et apportent ainsi une réduction significative du temps de recherche. La troisième catégorie semble être inutilisable, car elle augmente la taille de la base de clause de manière trop importante.

Pour traiter plus spécifiquement des problèmes de planification, [vanGelder, Okushi, 1999] implémentent le prouveur MODOC. Il est basé sur des techniques de résolution en chaînage arrière qui lui permettent de se focaliser sur les clauses qui correspondent aux buts du problème. Il est également capable de déterminer les clauses qui ne sont pas pertinentes pour la résolution du problème par rapport à une interprétation partielle, ce qui lui permet de compléter plus efficacement cette interprétation partielle pour en faire un modèle. Les performances de MODOC sont compétitives avec celles du prouveur incomplet WALKSAT [Selman, Kautz, Cohen, 1994], mais sont beaucoup moins bonnes que celles du prouveur complet SATO [Zhang, 1997] qui possède une implémentation extrêmement efficace de la propagation unitaire.

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 un algorithme qui évite cette transformation, en appliquant une méthode inspirée de la procédure de Davis et Putnam [Davis, Putnam, 1960], [Davis, Logemann, Loveland, 1962] directement sur les actions et les fluents que l'on peut obtenir à partir de la description STRIPS du problème (pour un nombre de niveaux donné). Cet algorithme utilise des règles de propagation sur des clauses issues d'un codage parallèle dans les espaces d'états du problème avec frame-axiomes explicatifs. D'autres règles de propagation sont appliquées, correspondant aux effets de la propagation unitaire dans un prouveur SAT. Les performances observées sont largement supérieures à celles de GRAPHPLAN, mais inférieures à celles de SATZ et WALKSAT. Cette idée est reprise dans le planificateur DPPLAN [Baioletti, Marcugini, Milani, 2000], qui utilise une procédure de Davis et Putnam pour extraire un plan-solution directement du graphe de planification. Plusieurs fonctions propagent des valeurs attachées aux nœuds d'actions et de fluents, ce qui donne une procédure de recherche bidirectionnelle. Les performances de DPPLAN sont comparables à celles obtenues avec SATZ. Cette approche est perfectionnée dans le planificateur LCDPPLAN [Vidal, 2001, 2002] qui utilise la relation d'autorisation [Cayrol, Régnier, Vidal, 2000, 2001] et améliore ainsi les performances de DPPLAN.

[Do, Srivastava, Kambhampati, 2000.a/b] 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.

[Vidal, 2001] simplifie les codages originaux (clauses redondantes), introduit le parallélisme dans les codages d'espaces de plans, code le graphe de planification par ses seules actions et exclusions mutuelles. Le travail présenté dans le chapitre 3 montre comment améliorer ces codages en utilisant la relation d'autorisation afin de produire des bases de clauses plus compactes en nombre de clauses et de variables. L'étude expérimentale de ces codages montre une amélioration significative des performances (cf. chapitres 4 et 5). [Rintanen, Heljanko, Niemelä, 2004] développent une approche similaire à la notre pour montrer comment les performances de la résolution peuvent être améliorées en employant la relation d'autorisation à la place de celle d'indépendance. Nous comparons, dans le chapitre 3, leur approche avec celle que nous avons employée.

[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.

Les premières approches de la planification SAT [Kautz, Selman, 1992, 1996], [Kautz, McAllester, Selman, 1996] s'intéressent à des codages pour lequels on connait à l'avance la longueur minimale d'un plan-solution. Dans les approches suivantes [Ernst, Millstein, Weld, 1997], [Kautz, Selman, 1998, 1999], [Baioletti, Marcugini, Milani, 1998], la stratégie de résolution standard cherche à satisfaire des codages représentant tous les plans d'une longueur fixée en commençant à une longueur minimale (soit 1, soit une longueur obtenue grâce à la construction d'un graphe de planification permettant d'obtenir les fluents du but). Tant qu'un plan-solution n'est pas trouvé par la résolution de ce codage, cette longueur est augmentée pour produire le codage suivant, ce qui garantit la production d'un plan-solution de nombre d'étapes minimal. [Rintanen, 2004], [Rintanen, Heljanko, Niemelä, 2006] étudient des stratégies basées sur l'évaluation parallèle ou intercalée de plusieurs formules. Ces stratégies peuvent éviter l'évaluation de certaines formules insatisfiables très difficiles et permettent des améliorations importantes des temps de résolution au prix d'une perte de la garantie d'optimalité du plan solution en nombre de niveaux.

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 de niveaux du plan). Ce résultat est inattendu puisque, par rapport à la version précédente de BLACKBOX, SATPLAN'04 n'intègre pas de nouveau codage ou prétraitement, et pas non plus de procédure de propagation des mutex. [Kautz, 2006] analyse les raisons de ce succès au premier plan desquelles il place les gains en performance des solveurs SAT, la difficulté des problèmes posés et leur forte structuration.

Le système MAXPLAN [Chen, Zhao, Zhang, 2007], [Xing, Chen, Zhang, 2006], [Zhao, Chen, Zhang, 2006] remporte, à égalité avec SATPLAN'06 [Kautz, Selman, Hoffman, 2006], la compétition IPC 2006 dans la catégorie des planificateurs optimaux en intègrant à SATPLAN'04 des mutex de longues distances (londex). Ceux-ci permettent une réduction de l'espace de recherche : ils généralisent les mutex classiques et peuvent capturer des contraintes entre actions situées à différents niveaux. Ils 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. SATPLAN'06 intègre une procédure de

propagation des mutex mais seulement pour les fluents car la prise en compte des mutex d'actions entraîne une saturation rapide de la mémoire.

[Robinson, Gretton, Pham, Sattar, 2008] utilisent dans leur système PARA-L un découpage des opérateurs, originellement proposé par [Kautz, Selman, 1992] : cette technique, qui avait déjà été introduite dans MEDIC uniquement dans le cas d'une recherche de plans séquentiels, est ici perfectionnée avec une prise en compte du parallélisme. Nous proposons, dans le chapitre 3, un codage compact qui utilise un découpage similaire et qui relaxe un peu plus le parallélisme en intégrant la relation d'autorisation.

Plusieurs études cherchent maintenant à étendre l'utilisation des techniques SAT pour sortir du cadre « classique » de la planification (environnement statique, observabilité totale, agent omniscient, actions atomiques et déterministes). [Wolfman, Weld, 1999, 2000] proposent ainsi, dans leur planificateur LPSAT, une extension de la planification SAT pour la résolution de problèmes de planification avec prise en compte de ressources. Le principe consiste à résoudre un problème constitué par une base de clauses et un ensemble de contraintes sous forme d'équations linéaires et d'inégalités, ces contraintes étant déclenchées par l'assignation de la valeur de vérité vrai à certaines variables de la base. On peut par exemple associer à une variable propositionnelle une contrainte précisant que la quantité de carburant disponible pour un véhicule doit être supérieure à une certaine valeur. Lorsque la valeur de cette variable passe à vrai, il faut s'assurer que cette contrainte est vérifiée, ainsi que les autres contraintes qui dépendent de cette variable. Dans le cas contraire, un retour arrière dans la procédure de Davis et Putnam est effectué, même si la base de clauses reste satisfiable. Le planificateur LPSAT utilise à cet effet une version modifiée de RELSAT [Bayardo, Schrag, 1997] et le solveur de contraintes arithmétiques linéaires CASSOWARY [Borning et al., 1997]. [Hoffmann, Kautz, Gomes, Selamn, 2007] développent une méthode permettant d'encoder des problèmes de planification numérique sous forme de problèmes SAT. Cette approche utilise une approximation de l'atteignabilité des variables du domaine. Ils la comparent à une autre traduction plus directe et qui utilise la « SAT Modulo Theory », extension de SAT avec des variables numériques et des contraintes arithmétiques. Dans les exemples qu'ils ont pu pratiquement tester, leur approche, optimale en nombre de niveaux, est plus efficace que la traduction avec SMT et rivalise même avec des planificateurs non-optimaux. [Mattmüller, Rintanen, 2007] étendent la planification SAT aux buts temporellement étendus (TEG : Temporally Extended Goals) qui permettent d'exprimer des contraintes qui imposent le passage par certains états pendant l'exécution du plan solution.

Plusieurs travaux [Castellini, Giunchiglia, Tacchclla, 2001, 2003], [Ferraris, Giunchiglia, 2000], [Giunchiglia, 2000] s'intéressent également à l'utilisation des techniques SAT pour la planification dans l'incertain dans le cas bien particulier de l'observabilité nulle (dans ce cas, on parle de « conformant planning »). [Rintanen, 2007.b] propose aussi plusieurs codages QBF (Quantified Boolean Formulae : extension de SAT avec des quantificateurs) pour ce type de problèmes de planification.

Après ce rapide tour d'horizon des travaux essentiels réalisés dans le domaine de la planification SAT, nous allons maintenant présenter en détail les codages qui ont servi de base à notre travail.

2.3. Nomenclature des différents codages

Nous emploierons les notations suivantes : MK99 pour dénoter les codages de [Mali, Kambhampati, 1999], KS99 pour les codages de [Kautz, Selman, 1999], V01 pour les codages de [Vidal, 2001]. Les différents codages que nous utiliserons seront donc désignés par les abréviations suivantes :

1. Codages dans les espaces d'états :
   - MK99 (1) et V01 (1) : espaces d'états avec *frame-axiomes explicatifs*.
   - MK99 (2) et V01 (2) : espaces d'états avec *no-ops*.
2. Codages dans les espaces de plans :
   - MK99 (3) et V01 (3) : espaces de plans par *liens causaux*, *protection d'intervalles* et *ordre partiel*.
   - MK99 (4) et V01 (4) : espaces de plans par *liens causaux*, *protection d'intervalles* et *étapes contiguës*.
   - MK99 (5) et V01 (5) : espaces de plans avec *white knight*.
3. Codages des graphes de planification :
   - KS99 (6) et V01 (6)

2.4. Codages MK99 et V01 dans les espaces d'états

Ces codages sont basés sur les transitions entre les niveaux successifs du plan, en partant de l'état initial et en se dirigeant vers le but. Le parallélisme peut être codé grâce à la notion d'indépendance entre actions simultanées. Afin de conserver les fluents qui ne sont pas affectés par les actions du niveau suivant, on introduit dans le codage la notion de *frame-axiome*. Nous décrivons une première technique, exposée par [Mali, Kambhampati, 1999], qui utilise des *frame-axiomes* explicatifs et une deuxième technique mise en oeuvre dans BLACKBOX [Kautz, Selman, 1999] utilisant des no-ops. Nous indiquons aussi les modifications apportées par [Vidal, 2001].

2.4.1. Codage dans les espaces d’états avec frame-axiomes explicatifs

Les règles d'un codage dans les espaces d'états produisent deux formes de propositions. Les propositions de la forme $A(i)$, où A est une action et i un entier naturel, représentent le fait que l'action A est appliquée à un niveau i du plan si et seulement si $A(i)$ a la valeur de vérité *vrai*. Les propositions de la forme $f(i)$, où f est un fluent et i un entier naturel, représentent le fait que le fluent f est présent au niveau i si et seulement si $f(i)$ a la valeur de vérité *vrai*. Le fait que le fluent f soit présent au niveau i signifie qu'il est présent après l'application successive de toutes les actions du niveau 1 jusqu'au niveau i qui sont associées à des propositions ayant la valeur de vérité *vrai*. Le nombre d'actions et le nombre de niveaux étant finis, nous nous plaçons bien dans le cadre de la logique des propositions. L'utilisation de parenthèses autorise une notation prédicative d'une logique d'ordre supérieur mais la base de clauses obtenue reste propositionnelle.

Le codage V01 comporte quatre règles :

- **La première règle** décrit l'état initial et l'état but : les fluents de l'état initial sont vrais au niveau 0, tous ceux qui n'en font pas partie sont faux au niveau 0, et les fluents du but sont vrais au niveau *k*.

$$1.\quad \left(\bigwedge_{f\in I} f(0)\right)\wedge\left(\bigwedge_{f\in(F-I)} \neg f(0)\right)\wedge\left(\bigwedge_{f\in G} f(k)\right)$$

- **La deuxième règle** traduit les préconditions et les effets des actions : une action du niveau *i* implique la conjonction de ses préconditions au niveau *i-1*, de ses ajouts et des négations de ses retraits au niveau *i*.

$$2\quad \bigwedge_{i\in[1,k]}\bigwedge_{a\in O}\left(a(i)\Rightarrow\left(\bigwedge_{f\in \mathrm{Pr}ec(a)} f(i-1)\right)\wedge\left(\bigwedge_{f\in Add(a)} f(i)\right)\wedge\left(\bigwedge_{f\in Del(a)} \neg f(i)\right)\right)$$

- **La troisième règle** définit les *frame-axiomes* explicatifs. Elle a été améliorée par [Vidal, 2001] pour supprimer des clauses inutiles lorsqu'un fluent n'est ajouté ni par l'état initial ni par une action, ou lorsqu'il n'est retiré par aucune action :
  - *frame-axiomes* de retrait : dans MK99 si un fluent vrai au niveau *i-1* devient faux au niveau *i*, alors la disjonction des actions du niveau *i* qui peuvent le rendre faux doit être vraie, c'est à dire qu'une action au moins qui le retire doit avoir été appliquée. Pour V01, il faut de plus que le fluent puisse exister à un moment donné, donc qu'il soit un élément de *I* ou de *Fa*. Il faut aussi qu'il puisse être retiré par une action, donc qu'il soit un élément de *Fd*. Il doit donc être un élément de $(I \cup Fa) \cap Fd$.

$$3.1\quad \bigwedge_{i\in[1,k]}\ \bigwedge_{f\in((I\cup F_a)\cap F_d)}\left(\left(f(i-1)\wedge\neg f(i)\right)\Rightarrow \bigvee_{a\in O\,/\,f\in Del(a)} a(i)\right)$$

  - *frame-axiomes* d'ajout : dans MK99, si un fluent faux au niveau *i-1* devient vrai au niveau *i*, alors la disjonction des actions du niveau *i* qui peuvent l'établir est vraie, c'est à dire qu'une action au moins qui l'établit doit avoir été appliquée. Pour V01, il faut de plus que le fluent puisse ne pas exister à un moment donné, donc il ne doit pas être élément de *I* ou il doit être élément de *Fd*. Il faut aussi qu'il puisse être ajouté par une action et donc être un élément de *Fa*. Il doit donc être un élément de $((F - I) \cup Fd) \cap Fa$.

$$3.2\quad \bigwedge_{i\in[1,k]}\ \bigwedge_{f\in(((F-I)\cup F_d)\cap F_a)}\left(\left(\neg f(i-1)\wedge f(i)\right)\Rightarrow \bigvee_{a\in O\,/\,f\in Add(a)} a(i)\right)$$

- Comme l'explique [Vidal, 2001], le codage MK99 ne prend pas en compte les interactions croisées. Nous devons donc interdire à une action de retirer une précondition d'une autre action à un même niveau. L'ajout de cette quatrième règle, décrite dans [Mali, Kambhampati, 1998], nous fournit un codage complet. Cependant, ce codage produit des propositions et clauses inutiles éliminées par la version améliorée du codage V01. En effet, la quatrième règle modifiée du codage V01 n'ajoute une clause d'exclusion mutuelle entre actions que si cette dernière n'a pas déjà été ajoutée par la règle 2 qui prend en

compte les effets contradictoires : l'exclusion est ajoutée si les deux actions ont une interaction croisée et pas d'effets contradictoires :

$$4.\quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2 \,/\, m<n \\ \wedge((Pre c(a_m)\cap Del(a_n)\neq\emptyset)\vee(Del(a_m)\cap Pre c(a_n)\neq\emptyset)) \\ \wedge(Add(a_m)\cap Del(a_n)=\emptyset)\wedge(Del(a_m)\cap Add(a_n)=\emptyset)}} \left(\neg a_m(i)\vee\neg a_n(i)\right)$$

2.4.2. Codage dans les espaces d'états avec no-ops

Ce codage, qui reste très proche du précédent (les règles 1, 4 et la moitié de la règle 2 sont conservées), utilise des no-ops à la place des *frame-axiomes* explicatifs. L'idée provient du graphe de planification du planificateur GRAPHPLAN [Blum, Furst, 1995], [Blum, Furst, 1997] et a été proposée pour la première fois dans [Kautz, McAllester, Selman, 1996]. Nous ajoutons de nouveaux symboles qui correspondent aux propositions associées aux no-ops. Ces propositions sont de la forme $N(f, i)$, où f est un fluent et i est un entier naturel, et représentent le fait que le fluent f présent au niveau $i-1$ est présent au niveau i si $N(f, i)$ a la valeur de vérité *vrai*. Il faut ajouter à la deuxième règle du codage précédent le fait que le no-op d'un fluent f appliqué au niveau i implique sa précondition au niveau $i-1$ et son effet au niveau i, selon le même principe que les autres actions. La troisième règle qui concernait les *frame-axiomes* explicatifs est entièrement modifiée pour prendre en compte les no-ops. La nouvelle règle est unique et elle exprime le fait que si un fluent f est présent au niveau i, c'est soit parce que l'on a appliqué son no-op ($N(f, i)$ a la valeur de vérité *vrai*), soit parce que l'on a appliqué une action qui le produit.

Ce codage produit plus de propositions et de clauses que le codage dans les espaces d'états avec *frame-axiomes* explicatifs mais l'introduction des no-ops produit des clauses de taille 2 par la deuxième règle et élimine des clauses de taille supérieure ou égale à 3 produites par la troisième règle du codage avec *frame-axiomes* explicatifs. Pour ne pas alourdir inutilement la lecture, nous le donnons en Annexe 1 (Codage 2).

2.5. Codages MK99 et V01 dans les espaces de plans

Ces codages ne traduisent plus seulement les transitions entre niveaux successifs du plan. Ils expriment maintenant les relations de causalité entre les actions du plan dans son ensemble. Dans les espaces d'états, on considérait l'application des actions de manière séquentielle : une action pouvait apparaître à un niveau du plan parce que ses préconditions étaient satisfaites au niveau précédent. Ici, une action apparaît à un niveau du plan parce qu'une action qui la précède à un niveau antérieur (pas forcément au niveau précédent) établit ses préconditions et qu'une action qui la suit réclame ses effets. Le plan est considéré comme un ensemble inter-relié d'actions.

L'existence des différentes variantes de codage dans les espaces de plans découle de la traduction des différentes stratégies de recherche dans les espaces de plans. Les ensembles d'actions indépendantes deux à deux qui peuvent faire partie d'un plan sont associés à des symboles qui représentent des étapes. Un ordre sur ces étapes détermine un ordre total pour l'exécution des ensembles d'actions indépendantes.

Les règles des différentes formes de codage dans les espaces de plans produisent plusieurs formes de propositions :

- Les propositions de la forme ($a = p$), où p est une étape du plan et a une action, représentent le fait que l'étape p est l'action a si et seulement si ($a = p$) a la valeur de vérité *vrai*. Dans ce cas, une étape représente une seule action et le parallélisme est interdit. Ces propositions sont utilisées dans les codages MK99.
- Les propositions de la forme ($a \in p$), où p est une étape du plan et a une action, représentent le fait que l'action a appartient à l'étape p si et seulement si ($a \in p$) a la valeur de vérité *vrai*. Ces propositions remplacent les précédentes dans les codages V01 et permettent ainsi de prendre en compte le parallélisme.
- Les propositions de la forme Adds(p, f), où p est une étape du plan et f est un fluent, représentent le fait que l'étape p contient une action qui ajoute le fluent f si et seulement si Adds(p, f) a la valeur de vérité *vrai*.
- Les propositions de la forme Needs(p, f), où p est une étape du plan et f est un fluent, représentent le fait que l'étape p contient une action qui nécessite f comme précondition si et seulement si Needs(p, f) a la valeur de vérité *vrai*.
- Les propositions de la forme Dels(p, f), où p est une étape du plan et f est un fluent, représentent le fait que l'étape p contient une action qui détruit le fluent f si et seulement si Dels(p, f) a la valeur de vérité *vrai*.
- Les propositions Link(p_i, f, p_j), où p_i et p_j sont deux étapes du plan et f un fluent, représentent un lien causal : l'étape p_i produit le fluent f qui est une précondition de l'étape p_j si et seulement si Link(p_i, f, p_j) a la valeur de vérité *vrai*.
- Les propositions de la forme Pred(p_i, p_j), où p_i et p_j sont deux étapes du plan, représentent le fait que l'étape p_i précède l'étape p_j, c'est-à-dire que les actions de l'ensemble d'actions indépendantes associées à p_i doivent être exécutées avant les actions associées à p_j si et seulement si Pred(p_i, p_j) a la valeur de vérité *vrai*. Ces propositions sont utilisées lorsque l'on veut traduire un ordre partiel sur les étapes.

2.5.1. Partie commune des codages dans les espaces de plans

Cette partie commune permet d'établir une correspondance entre toutes les actions disponibles et celles qui font partie des plans solutions. Le codage MK99 de cette partie utilise cinq règles (Annexe 1, Codage 3, p. 154) :

- La première règle établit l'équivalence entre étape et action : une étape peut être soit une action, soit Φ (l'action nulle, ou no-op).

$$1. \quad \bigwedge_{i \in [1,k]} \left(\bigvee_{a \in O} (p_i = a) \vee (p_i = \varphi) \right)$$

- **La deuxième règle** impose l'unicité des étapes : une étape correspond à une et une seule action (ou à Φ) à un niveau donné du plan. Cette règle interdit le parallélisme entre les actions et elle est supprimée dans le codage V01.

2. $$\bigwedge_{i\in[1,k]} \bigwedge_{a_1\in O} \bigwedge_{a_2\in O / a_1\neq a_2} \left(\neg(p_i = a_1) \vee \neg(p_i = a_2)\right)$$
$$\bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \left(\neg(p_i = a) \vee \neg(p_i = \varphi)\right)$$

- **La troisième règle** définit l'étape *Init* de niveau 0 qui produit l'état initial : Adds(*Init*, *f*) est vraie pour tous les fluents *f* de *I* et fausse pour les autres. Elle définit aussi l'étape *Goal* de niveau *k*+1, où *k* est le niveau le plus haut considéré pour une autre étape, dont les préconditions sont les fluents de l'état but : Needs(*Goal*, *f*) est vraie pour tous les fluents *f* de l'état but.

3. $$\left(\bigwedge_{f\in I} Adds(Init, f)\right) \wedge \left(\bigwedge_{f\in(F-I)} \neg Adds(Init, f)\right) \wedge \left(\bigwedge_{f\in G} Needs(Goal, f)\right)$$

- **La quatrième règle** établit que si l'étape *p* est associée à une action *a* (différente de Φ), alors la conjonction des ajouts, des préconditions et des retraits de *a* est vraie : Adds(*p*, *f*) est vraie pour les fluents *f* de Add(*a*), Needs(*p*, *f*) est vraie pour les fluents *f* de Prec(*a*), Dels(*p*, *f*) est vraie pour les fluents *f* de Del(*a*).

4. $$\bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \left((p_i = a) \Rightarrow \left(\bigwedge_{f\in Add(a)} Adds(p_i, f)\right) \wedge \left(\bigwedge_{f\in Prec(a)} Needs(p_i, f)\right) \wedge \left(\bigwedge_{f\in Del(a)} Dels(p_i, f)\right)\right)$$

- **La cinquième règle** établit que si une étape ajoute, retire ou a pour précondition un fluent, alors cette étape peut correspondre à n'importe quelle action qui a le même comportement vis-à-vis de ce fluent.

5. $$\bigwedge_{i\in[1,k]} \bigwedge_{f\in F} \left(Adds(p_i, f) \Rightarrow \bigvee_{a\in O / f\in Add(a)} (p_i = a)\right)$$
$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in F} \left(Dels(p_i, f) \Rightarrow \bigvee_{a\in O / f\in Del(a)} (p_i = a)\right)$$
$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in F} \left(Needs(p_i, f) \Rightarrow \bigvee_{a\in O / f\in Prec(a)} (p_i = a)\right)$$

Dans le codage V01 (Annexe 1, Codage 4, p. 154), l'action Φ n'est plus considérée comme une action particulière et elle peut ou non faire partie de l'ensemble *O* de toutes les actions. Dans la suite, lorsque nous comparerons les différents codages, nous ne la considèrerons pas car son introduction n'apporte rien et dégrade généralement les performances.

- **La première règle** qui est supprimée dans le codage V01 original sera conservée sous une forme modifiée pour obtenir un nombre d'étapes égal au nombre de niveaux du plan solution.

$$1.\quad \bigwedge_{i\in[1,k]}\left(\bigvee_{a\in O}(a\in p_i)\right)$$

- **La deuxième règle** du codage MK99 est simplifiée et modifiée pour autoriser le parallélisme lorsque les actions sont indépendantes.

$$2.\quad \bigwedge_{i\in[1,k]}\ \bigwedge_{\substack{(a_m,a_n)\in O^2\,/\,m<n\wedge\\ ((Add(a_m)\cup Prec(a_m))\cap Del(a_n)\neq\varnothing\\ \vee(Add(a_n)\cup Prec(a_n))\cap Del(a_m)\neq\varnothing)}}\left(\neg(a_m\in p_i)\vee\neg(a_n\in p_i)\right)$$

- **La troisième règle** est conservée et la quatrième adaptée de la manière suivante :

$$4.\quad \bigwedge_{i\in[1,k]}\bigwedge_{f\in F_a}\left(Adds(p_i,f)\Leftrightarrow\bigvee_{a\in O\,/\,f\in Add(a)}(a\in p_i)\right)$$

$$\bigwedge_{i\in[1,k]}\bigwedge_{f\in F_d}\left(Dels(p_i,f)\Leftrightarrow\bigvee_{a\in O\,/\,f\in Del(a)}(a\in p_i)\right)$$

$$\bigwedge_{i\in[1,k]}\bigwedge_{f\in F_p}\left(Needs(p_i,f)\Leftrightarrow\bigvee_{a\in O\,/\,f\in Prec(a)}(a\in p_i)\right)$$

Pour trouver un plan minimal en nombre d'étapes, nous emploierons une approche incrémentale similaire à celle utilisée dans BLACKBOX [Kautz, Selman, 1999] : la traduction sera itérée en augmentant la longueur du plan solution recherché jusqu'à ce qu'une solution soit trouvée.

Pour la suite du codage, nous utiliserons soit la partie commune des codages MK99 (Annexe 1, Codage 3, p. 154) soit celle de V01 (Annexe 1, Codage 4, p. 154).

2.5.2. Codage par liens causaux, protection d'intervalles et ordre partiel

Dans ce codage, l'établissement des préconditions se fait par un codage direct des liens causaux de la forme Link(p_i, f, p_j), qui permettent de spécifier que l'étape p_i produit le fluent f qui est une précondition de l'étape p_j. La protection de f dans l'intervalle compris entre les deux étapes se fait soit par promotion, c'est-à-dire que l'étape qui menace f est placée avant p_i, soit par démotion, ce qui veut dire que l'étape qui menace f est placée après p_j. Les numéros d'indice des étapes ne correspondent pas à l'ordre d'exécution des actions. Ce dernier est donné par une relation de précédence de la forme Pred(p_i, p_j).

Le codage V01 apporte des corrections au codage MK99 pour supprimer des clauses inutiles et des modèles qui ne correspondent pas à des plans solutions. En ce qui concerne cette partie, nous n'utiliserons dans la suite que le codage corrigé V01.

- **La première règle** à ajouter à la partie commune est une règle d'établissement des préconditions : chaque fluent est supporté par un lien causal entre l'étape qui le produit (qui peut être l'état initial) et l'étape qui l'utilise en tant que précondition.

$$1.\quad \bigwedge_{j\in[1,k]} \bigwedge_{f\in F_p} \left(Needs(p_j, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{i\in[1,k]/\, i\neq j} Link(p_i, f, p_j) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Link(Init, f, p_j) \vdots \bot \right] \end{array} \right) \right)$$

$$\bigwedge_{f\in G} \left(Needs(Goal, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{i\in[1,k]} Link(p_i, f, Goal) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Link(Init, f, Goal) \vdots \bot \right] \end{array} \right) \right)$$

- **La deuxième règle** concerne les liens causaux et indique que la présence d'un lien causal supportant un fluent *f* entre une étape p_i et une étape p_j implique que l'étape p_i ajoute *f*, que l'étape p_j a pour précondition *f*, et que l'étape p_i précède l'étape p_j.

$$2.\quad \bigwedge_{i\in[1,k]} \bigwedge_{j\in[1,k]/\, j\neq i} \bigwedge_{f\in(F_p\cap F_a)} \left(Link(p_i, f, p_j) \Rightarrow Adds(p_i, f) \wedge Needs(p_j, f) \wedge \Pr ed(p_i, p_j) \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(G\cap F_a)} \left(Link(p_i, f, Goal) \Rightarrow Adds(p_i, f) \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(F_p\cap I)} \left(Link(Init, f, p_i) \Rightarrow Needs(p_i, f) \right)$$

- **La troisième règle** de protection d'intervalles exige que s'il y a un lien causal qui supporte le fluent *f* entre une étape p_i et une étape p_j, et si une étape p_q retire *f*, alors p_q doit précéder p_i (promotion) ou p_j doit précéder p_q (démotion).

$$3.\quad \bigwedge_{i\in[1,k]} \bigwedge_{j\in[1,k]/\, j\neq i} \bigwedge_{q\in[1,k]/\, q\neq i\wedge q\neq j} \bigwedge_{f\in(F_p\cap F_a\cap F_d)} \left(\begin{array}{l} Link(p_i, f, p_j) \wedge Dels(p_q, f) \\ \Rightarrow \Pr ed(p_q, p_i) \vee \Pr ed(p_j, p_q) \end{array} \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{j\in[1,k]/\, j\neq i} \bigwedge_{f\in(G\cap F_a\cap F_d)} \left(Link(p_i, f, Goal) \wedge Dels(p_j, f) \Rightarrow \Pr ed(p_j, p_i) \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{j\in[1,k]/\, j\neq i} \bigwedge_{f\in(F_p\cap I\cap F_d)} \left(Link(Init, f, p_i) \wedge Dels(p_j, f) \Rightarrow \Pr ed(p_i, p_j) \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(G\cap I\cap F_d)} \left(Link(Init, f, Goal) \wedge Dels(p_i, f) \Rightarrow \bot \right)$$

- **La quatrième règle** donne les propriétés de la relation de précédence qui doit être irréflexive, antisymétrique et transitive.

$$4.\quad \bigwedge_{i\in[1,k]} \bigwedge_{j\in[1,k]/\, j\neq i} \bigwedge_{s\in[1,k]/\, s\neq i\wedge s\neq j} \left(\Pr ed(p_i, p_j) \wedge \Pr ed(p_j, p_s) \Rightarrow \Pr ed(p_i, p_s) \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{j\in[i+1,k]} \left(\neg \Pr ed(p_i, p_j) \vee \neg \Pr ed(p_j, p_i) \right)$$

2.5.3. Codage par liens causaux, protection d'intervalles et étapes contiguës

Le but de ce codage est de simplifier le précédent dans lequel plusieurs modèles correspondent au même plan solution, à la numérotation des étapes près. Pour cela, on contraint les étapes à suivre un ordre prédéfini et on considère qu'elles sont ordonnées suivant

l'ordre croissant de leur indice, ce qui rend inutile la relation de précédence. La protection d'intervalles ne se fait plus par promotion ou démotion en choisissant un ordre partiel sur les étapes, mais en interdisant qu'une étape comprise dans un intervalle défini par un lien causal ne menace ce dernier. Pour obtenir ce résultat, on ajoute deux règles à la partie commune du codage :

- La règle d'établissement des préconditions indique que chaque fluent est supporté par un lien causal entre l'étape qui le produit (qui peut être l'état initial) et l'étape qui l'utilise en tant que précondition.

$$1.\quad \bigwedge_{i\in[1,k]} \bigwedge_{f\in F_p} \left(Needs(p_i, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{j\in[1,i-1]} Link(p_j, f, p_i) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Link(Init, f, p_i) \vdots \bot \right] \end{array} \right) \right)$$

$$\bigwedge_{f\in G} \left(Needs(Goal, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{i\in[1,k]} Link(p_i, f, Goal) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Link(Init, f, Goal) \vdots \bot \right] \end{array} \right) \right)$$

- L'autre règle concerne les liens causaux et la protection d'intervalles et impose que la présence d'un lien causal supportant un fluent *f* entre une étape p_i et une étape p_j implique que l'étape p_i ajoute *f* et que l'étape p_j a pour précondition *f*. De plus, une étape qui retire *f* ne peut pas s'insérer entre p_i et p_j.

$$2.\quad \bigwedge_{i\in[1,k-1]} \bigwedge_{j\in[i+1,k]/\, j\neq i} \bigwedge_{f\in(F_p\cap F_a)} \left(Link(p_i, f, p_j) \Rightarrow \left(\begin{array}{l} Adds(p_i, f) \wedge Needs(p_j, f) \\ \wedge \left[f \in F_d \mapsto \bigwedge_{q\in[i+1, j-1]} \neg Dels(p_q, f) \vdots \mathrm{T} \right] \end{array} \right) \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(G\cap F_a)} \left(Link(p_i, f, Goal) \Rightarrow Adds(p_i, f) \wedge \left[f \in F_d \mapsto \bigwedge_{j\in[i+1,k]} \neg Dels(p_j, f) \vdots \mathrm{T} \right] \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(F_p\cap I)} \left(Link(Init, f, p_i) \Rightarrow Needs(p_i, f) \wedge \left[f \in F_d \mapsto \bigwedge_{j\in[1,i-1]} \neg Dels(p_j, f) \vdots \mathrm{T} \right] \right)$$

$$\bigwedge_{f\in(I\cap G)} \left(Link(Init, f, Goal) \Rightarrow \left[f \in F_d \mapsto \bigwedge_{i\in[1,k]} \neg Dels(p_i, f) \vdots \mathrm{T} \right] \right)$$

Des corrections similaires à celles-ci sont apportées par [Vidal, 2001] avec le codage V01. Pour la même raison nous n'utiliserons que cette version des codages dans la suite.

2.5.4. Codage du "white knight"

Si l'on se base sur le nombre de variables et sur la taille de la base de clauses produite, ce codage est actuellement le plus compact parmi les codages dans les espaces de plans. Il n'utilise plus de liens causaux comme précédemment mais les exprime directement : si une étape a besoin d'un fluent, c'est qu'une étape précédente doit l'avoir créé. Ceci permet de n'utiliser que les variables déjà présentes dans la partie commune du codage. La protection d'intervalles se réalise en codant la technique du "white knight" introduite dans le planificateur TWEAK [Chapman, 1987]. Pour cela, on ajoute deux règles à celles de la partie

commune en y intègrant les corrections apportées par [Vidal, 2001] pour supprimer des clauses inutiles et des modèles qui correspondent à des plans ne résolvant pas le problème.
Pour notre étude comparative des codages nous n'utiliserons donc que la version V01 de ces règles.

- **La règle d'établissement des préconditions** impose que la précondition d'une étape d'un niveau *i* doit être ajoutée par une étape qui la précède.

$$1.\quad \bigwedge_{i\in[1,k]} \bigwedge_{f\in F_p} \left(Needs(p_i, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{j\in[1,i-1]} Adds(p_j, f) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Adds(Init, f) \vdots \bot \right] \end{array} \right) \right)$$

$$\bigwedge_{f\in G} \left(Needs(Goal, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{i\in[1,k]} Adds(p_i, f) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Adds(Init, f) \vdots \bot \right] \end{array} \right) \right)$$

- **La règle du "white knight"** impose que, si une première étape (ou l'étape but Goal) a pour précondition le fluent *f* au niveau *i*, et qu'une deuxième étape retire ce fluent à un niveau *j* antérieur au niveau *i*, avant que la première étape puisse l'utiliser, alors il doit y avoir une troisième étape qui rétablit *f* à un niveau *q* tel que $j < q < i$.

$$2\quad \bigwedge_{i\in[2,k]} \bigwedge_{j\in[1,i-1]} \bigwedge_{f\in(F_p\cap F_d)} \left(Needs(p_i, f) \wedge Dels(p_j, f) \Rightarrow \left[f \in F_a \mapsto \bigvee_{q\in[j+1,i-1]} Adds(p_q, f) \vdots \bot \right] \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(G\cap F_d)} \left(Needs(Goal, f) \wedge Dels(p_i, f) \Rightarrow \left[f \in F_a \mapsto \bigvee_{j\in[i+1,k]} Adds(p_j, f) \vdots \bot \right] \right)$$

2.6. Codages KS99 et V01 du graphe de planification

L'approche utilisée dans BLACKBOX [Kautz, Selman, 1999] est celle qui a donné, pour la planification de type SAT, les résultats les plus convainquants. BLACKBOX code le graphe de planification construit en un temps polynomial par GRAPHPLAN en une base de clauses pour en extraire un plan solution. Ce graphe peut être considéré comme une étape de filtrage qui permet de diminuer la taille de la base de clauses en ne sélectionnant, parmi toutes les actions et fluents du problème, que ceux qui sont susceptibles de contribuer à l'établissement des buts. La phase d'extraction qui est originellement réalisée par une procédure spécifique à GRAPHPLAN est dévolue, dans BLACKBOX, à un prouveur SAT. Le graphe peut être représenté :

- soit directement par le codage, aux différents niveaux, des actions, des fluents et de leurs exclusions mutuelles (codage KS99),
- soit sous une forme condensée (codage V01) qui prends en compte le fait que la représentation des seules actions du plan permet l'extraction d'une solution : en effet, la représentation des fluents n'est pas indispensable car ils peuvent facilement être déduits des actions utilisées.

2.6.1. Codage KS99 du graphe de planification

Le codage KS99 contient quatre règles :

- **Règle 1 (Etat initial et but) :** les fluents de l'état initial et du but sont vrais.

$$1. \quad \left(\bigwedge_{n_f \in NoeudsInit(GP)} n_f \right) \wedge \left(\bigwedge_{n_f \in NoeudsBut(GP)} n_f \right)$$

- **Règle 2 (Préconditions des actions) :** si une action est utilisée, alors ses préconditions sont vraies au niveau précédent.

$$2. \quad \bigwedge_{(n_f, n_a) \in Arcs\,Pr\,ec(GP)} (n_a \Rightarrow n_f)$$

- **Règle 3 (Etablissement des fluents) :** si un fluent est vrai à un niveau supérieur à l'état initial, alors la disjonction des actions qui peuvent le produire au niveau précédent est vraie (no-op compris s'il est présent à ce niveau). Ces clauses sont déterminées par les arcs d'ajouts calculés pendant la construction du graphe de planification.

$$3. \quad \bigwedge_{n_f \in (NoeudsF(GP) - NoeudsInit(GP))} \left(n_f \Rightarrow \bigvee_{(n_a, n_f) \in ArcsAdd(GP)} n_a \right)$$

- **Règle 4 (Exclusions mutuelles) :** les actions qui correspondent à des nœuds mutuellement exclusifs ne peuvent pas être vraies en même temps. Ces exclusions mutuelles sont les mutex du graphe et les mutex supplémentaires d'atteignabilité trouvés lors de sa construction.

$$4. \quad \bigwedge_{\{n_a, n_b\} \in MutexA(GP)} (\neg n_a \vee \neg n_b)$$

2.6.2. Codage V01 du graphe de planification

La simplification du codage KS99 précédent donne le codage V01 qui ne comporte que des actions et nécessite les trois règles suivantes :

- **Règle 1 (But) :** pour chacun des fluents du but, il y a au moins une action qui le produit qui doit avoir été appliquée.

$$1. \quad \bigwedge_{n_f \in NoeudsBut(GP)} \left(\bigvee_{(n_a, n_f) \in ArcsAdd(GP)} n_a \right)$$

- **Règle 2 (Etablissement des préconditions) :** les préconditions de chaque action du plan, si elles ne se trouvent pas dans l'état initial, doivent être établies par au moins une action.

2. $$\bigwedge_{(n_f,n_a)\in Arcs\,Prec(GP)\,/\,n_f\notin NoeudsInit(GP)} \left(n_a \Rightarrow \bigvee_{(n_b,n_f)\in ArcsAdd(GP)} n_b \right)$$

- **Règle 3 (Exclusions mutuelles) :** règle 4 du codage KS99.

3. $$\bigwedge_{\{n_a,n_b\}\in MutexA(GP)} \left(\neg n_a \vee \neg n_b\right)$$

Ces deux formes de codage du graphe de planification diffèrent par le nombre de variables et de clauses qu'elles vont produire pour un même problème (nombre de variables moindre et nombre de clauses plus important pour V01). Leurs performances respectives devront néanmoins être soigneusement étudiées.

Après avoir, dans ce chapitre 2, dressé un état de l'art des techniques et des codages utilisés par les planificateurs SAT et présenté en détail les codages essentiels de [Mali, Kambhampati, 1999], [Kautz, Selman, 1999] et les améliorations apportées par [Vidal, 2001], nous allons maintenant détailler, dans le chapitre 3 ci-après, le travail que nous avons effectué pour améliorer ces codages. Nous présenterons ensuite, dans les chapitres suivants, le système TSP (Tunable SatPlan, cf. chapitre 4) que nous avons développé pour comparer équitablement les performances de ces codages et les résultats expérimentaux qui montrent la supériorité des codages intégrant la relation d'autorisation (cf. chapitre 5).

3. Introduction de la relation d'autorisation dans les codages

3.1. Définition de la relation d’autorisation

3.1.1. Introduction

La relation d'autorisation a été originellement proposée dans une version que nous appelerons « autorisation forte » par [Dimopoulos, Nebel, Koehler, 1997] pour le codage d’un problème de planification en logique non monotone et la recherche de modèles stables. Lorsque cette relation (moins contraignante que la relation d'indépendance) est vérifiée, elle permet, pour un ordre d'exécution déterminé d'un ensemble d'actions fortement autorisé, d'obtenir un état résultant par un calcul identique à celui effectué pour l'indépendance (cf. section II.5.1, Définition 28 et Définition 29). Par la suite, [Vidal, 2001] a introduit, de manière indépendante de [Dimopoulos, Nebel, Koehler, 1997], une relation, dite d’ « autorisation faible », encore moins contraignante que la relation d’autorisation forte. Vidal utilise cette relation pour améliorer de manière très importante les performances des planificateurs GRAPHPLAN [Blum, Furst, 1997] et DPPLAN [Baioletti, Marcugini, Milani, 2000] en leur permettant de diminuer le nombre de niveaux nécessaires pour obtenir des plans-solutions [Vidal, 2001], [Cayrol, Régnier, Vidal, 2001]. La relation d’autorisation forte de [Dimopoulos, Nebel, Koehler, 1997] est donc plus contraignante que la relation d'autorisation faible de [Vidal, 2001] et ne présente pas d'avantage particulier par rapport à cette dernière. C’est donc la relation d'autorisation faible que nous utilisons ici. Dans la suite de notre exposé, et sauf mention expresse, le terme « autorisation » signifiera « autorisation faible ».

3.1.2. Définitions préliminaires

Nous donnons ci-après, et avant de les utiliser, les définitions respectives de ces différentes relations d'autorisation ainsi que quelques exemples de leur utilisation.

Définition 30 (autorisation forte entre deux actions)

Une action a autorise fortement une action $b \neq a$ (noté $a \angle b$) ssi :

$$Add(a) \cap Del(b) = \emptyset \text{ et } (Prec(b) \cup Add(b)) \cap Del(a) = \emptyset$$

Définition 31 (séquence fortement autorisée)

Une séquence d'actions $S = \langle a_i \rangle_n$ est fortement autorisée ssi :

$$\forall\, i, j \in [1, n], i<j \Rightarrow a_i \angle a_j, \text{ ce qui équivaut à :}$$
$$Del(S) \cap Add(S) = \emptyset \text{ et } \forall\, i, j \in [1, n-1], Prec(a_{i+1}) \cap (Del(a_1) \cup \ldots \cup Del(a_i))$$

Exemple : Ici, les actions A et B ne sont pas indépendantes car l'action B retire le fluent a qui est une précondition de l'action A : $(Prec(A) \cup Add(A)) \cap Del(B) = \{a\} \neq \emptyset$. Si l'on se base sur cette relation (qui est trop contrainte), on en déduit que ces actions ne peuvent pas être appliquées simultanément et que deux étapes sont donc nécessaires.

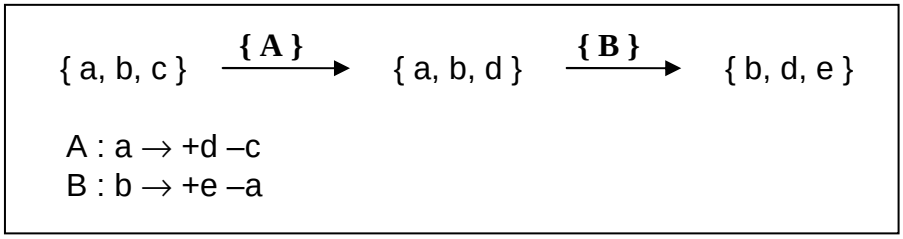


Figure 9 : plan séquentiel (conflit entre deux actions non indépendantes)

Si l'on utilise maintenant la relation d'autorisation forte, on voit que A autorise fortement B. On en déduit que les deux actions peuvent être appliquées en une seule étape en donnant le même état résultant {b, d, e} :

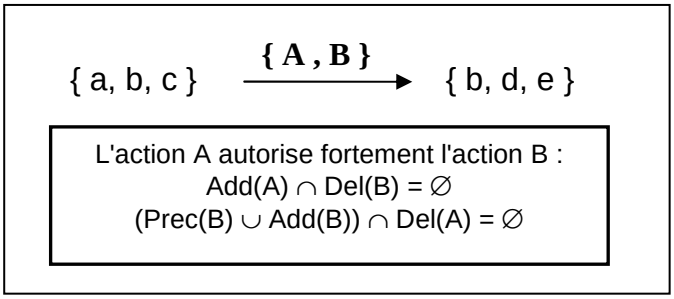


Figure 10 : plan fortement autorisé (autorisation forte entre deux actions)

Dans cet exemple, l'utilisation de la relation d'autorisation forte (cf. Figure 10) permet de ne pas tenir compte du conflit qui existait entre A et B et que capturait la relation d'indépendance puisque $(Prec(A) \cup Add(A)) \cap Del(B) = \{a\} \neq \emptyset$. Avec la relation d'autorisation forte, $Add(A)) \cap Del(B) = \emptyset$ et l'on en déduit que les actions A et B peuvent être appliquées au même niveau.

Voici maintenant la définition de la relation d'autorisation faible qui permet de relaxer encore d'avantage les contraintes permettant l'application d'actions au même niveau.

Définition 32 (autorisation faible entre deux actions)

Une action a autorise faiblement une action b ≠ a (noté a $\angle$ b) ssi :

$$Add(a) \cap Del(b) = \emptyset \text{ et } Prec(b) \cap Del(a) = \emptyset$$

Définition 33 (séquence faiblement autorisée)

Une séquence d'actions $<a_i>_n$ est faiblement autorisée ssi :

$$\forall\, i, j \in [1, n],\ i<j \Rightarrow a_i \angle a_j, \text{ ce qui équivaut à :}$$
$$\forall\, i \in [1, n-1],\ Del(a_{i+1}) \cap (Add(a_1) \cup \ldots \cup Add(a_i)) = \emptyset \text{ et}$$
$$Prec(a_{i+1}) \cap (Del(a_1) \cup \ldots \cup Del(a_i))$$

Les ajouts des actions $a_1 \ldots a_i$ sont conservés par a_{i+1} et l'exécution de a_{i+1} reste possible après celle des actions qui la précèdent. Cette relation d'autorisation faible permet une relaxation des contraintes encore plus importante que l'autorisation forte. En contrepartie, le calcul de l'état résultant doit prendre en compte l'ordre dans lequel les actions sont autorisées. Pour l'autorisation forte, le calcul de l'état résultant se faisait d'une manière identique à celle employée dans le cas de l'indépendance. Ainsi, pour une séquence fortement autorisée $S=<a_i>_n$ et un état E :

$$E \uparrow S = \left(s - \bigcup_{i \in \{1,n\}} Del(a_i) \right) \cup \bigcup_{i \in \{1,n\}} Add(a_i)$$

Pour l'autorisation faible, le calcul de l'état résultant doit maintenant se faire en prenant en compte l'ordre d'autorisation des actions (comme si on appliquait un plan séquentiel à l'état initial). Ainsi, pour une séquence faiblement autorisée $S=<a_i>_n$ et un état E :

$$E \uparrow S = \left(\cdots\left(\left(s - Del(a_1)\right) \cup Add(a_1)\right)\cdots - Del(a_n)\right) \cup Add(a_n)$$

Dans l'exemple de la Figure 11, l'action A et l'action C ne sont pas indépendantes et A n'autorise pas non plus fortement l'action C car (Prec(C) ∪ Add(C)) ∩ Del(A) = {c} ≠ ∅. Par contre, A autorise faiblement C car Add(A) ∩ Del(C) = ∅ et Prec(C) ∩ Del(A) = ∅. L'inverse n'est pas vrai : C n'autorise pas faiblement A puisque a, précondition de A, est détruite par C. Les actions A et C peuvent donc s'appliquer au même niveau et donner le même état résultant {b, c, d, e} :

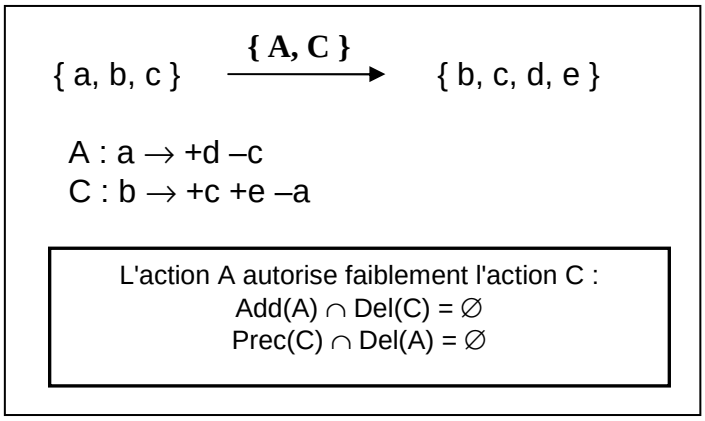


Figure 11 : plan faiblement autorisé (autorisation faible entre deux actions)

Comme le montre [Vidal, 2001], [Cayrol, Régnier, Vidal, 2001], la relation d'autorisation permet d'améliorer de manière très importante les performances des planificateurs GRAPHPLAN [Blum, Furst, 1997] et DPPLAN [Baioletti, Marcugini, Milani, 2000]. Cette amélioration est due à la diminution significative du nombre de niveaux du graphe de planification qu'il est nécessaire de construire pour obtenir une solution. Les graphes de planification produits sont en moyenne deux fois plus courts que ceux de GRAPHPLAN, en particulier sur des domaines pouvant combiner des manipulations d'objets et des déplacements. Cette baisse du nombre de niveaux entraîne une amélioration des temps de calculs. Sur les benchmarks testés, LCGP fonctionne environ 1800 fois plus vite que GRAPHPLAN et LCDPP environ 118 fois plus vite que DPPLAN. Cette diminution du temps de calcul n'a aucune influence sur la qualité des plans-solutions en nombre d'actions ou d'étapes indépendantes, puisque ces valeurs demeurent identiques dans la majorité des cas.

Dans le chapitre suivant nous allons maintenant montrer commment la relation d'autorisation peut être utilisée dans de nouveaux codages. La seule différence entre autorisation et indépendance concerne l'interaction entre actions et nous pouvons donc naturellement penser que la modification des seules règles concernant les interactions, sans modifier les autres règles, peut nous permettre d'obtenir des codages complets utilisant l'autorisation. Nous détaillons dans les sections suivantes les modifications à prendre en compte pour obtenir ces nouveaux codages et les nouvelles contraintes imposées par l'utilisation de cette relation, en particulier lors du décodage du plan-solution.

3.2. Intégration de la relation d'autorisation dans les codages

3.2.1. Nomenclature des différents codages

Dans la suite, les principaux codages intégrant la relation d'autorisation que nous allons définir seront désignés par les abréviations suivantes :

1. Codages dans les espaces d'états :
    - LCS (*Least Committed State-space encoding*) : espaces d'états avec *frame-axiomes explicatifs* et *relation d'autorisation*.
2. Codages dans les espaces de plans :
    - LCP (*Least Committed Plan-space encoding*) : espaces de plans avec *relation d'autorisation*.
3. Codages des graphes de planification :
    - LCG-KS99 et LCG-V01 (Least Committed Graphplan encoding) : graphes de planification avec relation d'autorisation.

3.2.2. Codages dans les espaces d'états et relation d'autorisation

Dans les deux codages dans les espaces d'états que nous avons étudiés dans l'état de l'art (section 2.4), la règle 4 permet d'interdire l'apparition simultanée de deux actions à un même niveau lorsqu'elles ne sont pas indépendantes. Cette contrainte est forte car, dans certains cas, la présence à un même niveau de deux actions non indépendantes mais autorisées pourrait permettre d'obtenir un état résultant identique à celui qui est obtenu en répartissant ces deux actions sur deux niveaux successifs. La modification de cette règle permet, en remplaçant la

relation d'indépendance par celle d'autorisation, de réduire le nombre de niveaux du plan solution, la taille de la base de clauses qui lui correspond, et on peut donc le penser, d'améliorer les performances (cf. chapitre 5). Le seul traitement à rajouter est la vérification, pour le plan correspondant au modèle renvoyé par le solveur, de l'existence d'une séquence d'actions autorisée pour chacun des niveaux successifs. Sous cette condition, le plan retourné est un plan solution au problème posé. Cette vérification se fait en un temps polynomial, et indépendamment du solveur. En cas de non satisfaction de cette condition, on rajoute une clause qui interdit le modèle correspondant et on relance le solveur au même niveau. Notons que sur les benchmarks classiques, ce cas particulier ne s'est jamais présenté. La nouvelle règle 4 (qui code le fait qu'aucune action n'autorise l'autre) est la suivante :

$$4. \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2 / m<n \\ \wedge((Add(a_m)\cap Del(a_n)\neq\emptyset)\vee(Del(a_m)\cap(Pre c(a_n)\cup Add(a_n))\neq\emptyset)) \\ \wedge((Del(a_m)\cap Add(a_n)\neq\emptyset)\vee((Pre c(a_m)\cup Add(a_m))\cap Del(a_n)\neq\emptyset))}} \left(\neg a_m(i)\vee\neg a_n(i)\right)$$

Cette règle peut être simplifiée en considérant que la deuxième règle interdit déjà les actions ayant des effets contradictoires.

$$4. \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2 / m<n \\ \wedge(Pre c(a_m)\cap Del(a_n)\neq\emptyset)\wedge(Del(a_m)\cap Pre c(a_n)\neq\emptyset) \\ \wedge(Del(a_m)\cap Add(a_n)=\emptyset)\wedge(Add(a_m)\cap Del(a_n)=\emptyset)}} \left(\neg a_m(i)\vee\neg a_n(i)\right)$$

Pour utiliser la relation d'autorisation faible, qui permet une relaxation supplémentaire du parallélisme, nous donnons un nouveau codage détaillé en Annexe 2. Nous présentons ici sa forme simplifiée :

- La quatrième règle doit encore être modifiée pour que deux actions qui ne s'autorisent pas faiblement l'une ou l'autre ne puissent être actives au même niveau. Ceci garantit l'existence d'une séquence autorisée lorsqu'il n'existe pas de cycle.

$$4. \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2 / m<n \\ \wedge((Add(a_m)\cap Del(a_n)\neq\emptyset)\vee(Del(a_m)\cap Pre c(a_n)\neq\emptyset)) \\ \wedge((Del(a_m)\cap Add(a_n)\neq\emptyset)\vee(Pre c(a_m)\cap Del(a_n)\neq\emptyset))}} \left(\neg a_m(i)\vee\neg a_n(i)\right)$$

Cette modification n'est pas suffisante et permet d'obtenir à nouveau un codage utilisant la relation d'autorisation forte. En effet, les effets contradictoires, interdits par la deuxième règle, doivent être partiellement autorisés lorsqu'on veut utiliser la relation d'autorisation faible.

- La deuxième règle doit donc être modifiée afin de ne plus rendre systématiquement faux les retraits des actions.

$$2.1 \quad \bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \left(a(i) \Rightarrow \left(\bigwedge_{f\in Pre c(a)} f(i-1) \right) \wedge \left(\bigwedge_{f\in Add(a)} f(i) \right) \right)$$

- Une nouvelle règle permet de rendre faux un fluent retiré par une action faisant partie de la séquence autorisée des actions actives au niveau i uniquement si aucune autre action suivante dans la séquence ne rétablit ce fluent.

$$2.2 \quad \bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \bigwedge_{f\in Del(a)} \left(\left(a(i) \wedge \bigwedge_{\substack{b\in O/(b\neq a)\wedge(f\in Add(b)) \\ \wedge(Add(a)\cap Del(b)=\emptyset)\wedge(Del(a)\cap Pre c(b)=\emptyset)}} \neg b(i) \right) \Rightarrow \neg f(i) \right)$$

Dans cette règle, l'action a autorise l'action b, mais l'action b n'autorise pas l'action a car Del(a) $\cap$ Add(b) $\supseteq$ { f } $\neq \emptyset$. Cette contrainte garantit bien que l'action b qui rétablit éventuellement f soit postérieure à l'action a dans la séquence autorisée des actions actives au niveau i.

Ce dernier codage simplifié permet la relaxation du parallélisme la plus importante et n'augmente ni le nombre de variables, ni le nombre de clauses par rapport au codage avec autorisation forte.

3.2.3. Codages dans les espaces de plans et relation d'autorisation

Dans la partie commune du codage V01 dans les espaces de plans, la règle 1 permet d'interdire l'apparition simultanée de deux actions dans une même étape lorsqu'elles ne sont pas indépendantes. Cette contrainte est forte car, dans certains cas, la présence dans une même étape, de deux actions non indépendantes mais autorisées pourrait permettre d'obtenir un état résultant identique à celui qui est obtenu en répartissant ces deux actions dans deux étapes successives. La modification de cette règle permet, en remplaçant la relation d'indépendance par celle d'autorisation, de réduire le nombre d'étapes du plan solution, la taille de la base de clauses qui lui correspond, et donc d'améliorer les performances (cf. chapitre 5). Le seul traitement à rajouter est la vérification, pour le plan correspondant au modèle renvoyé par le solveur, de l'existence d'une séquence d'actions autorisée au sein de chacune des étapes. Sous cette condition, le plan retourné est un plan solution au problème posé. Cette vérification se fait en un temps polynomial, et indépendamment du solveur. En cas de non satisfaction de cette condition, on rajoute une clause qui interdit le modèle correspondant et on relance le solveur au même niveau. Notons que sur les benchmarks classiques, ce cas particulier ne s'est jamais présenté. La nouvelle règle 1 (qui code l'autorisation) est la suivante :

$$1. \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2/m<n \\ \wedge((Add(a_m)\cap Del(a_n)\neq\emptyset)\vee(Del(a_m)\cap(Pre c(a_n)\cup Add(a_n))\neq\emptyset)) \\ \wedge((Del(a_m)\cap Add(a_n)\neq\emptyset)\vee((Pre c(a_m)\cup Add(a_m))\cap Del(a_n)\neq\emptyset))}} \left(\neg(a_m\in p_i)\vee\neg(a_n\in p_i)\right)$$

Comme dans le cas des codages dans les espaces d'états, un codage utilisant la relation d'autorisation faible peut être également envisagé.

$$1. \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2/m<n \\ \wedge((Add(a_m)\cap Del(a_n)\neq\emptyset)\vee(Del(a_m)\cap Pre c(a_n)\neq\emptyset)) \\ \wedge((Del(a_m)\cap Add(a_n)\neq\emptyset)\vee(Pre c(a_m)\cap Del(a_n)\neq\emptyset))}} \left(\neg(a_m\in p_i)\vee\neg(a_n\in p_i)\right)$$

La deuxième partie de la troisième règle doit être modifiée pour prendre en compte les éventuels rétablissement de fluent dans la séquence autorisée des actions actives dans une étape.

$$3.2 \quad \bigwedge_{i\in[1,k]} \bigwedge_{f\in F_d} \left[Dels(p_i, f) \Leftrightarrow \bigvee_{a\in O\,/\,f\in Del(a)} \left((a\in p_i) \wedge \bigwedge_{\substack{b\in O\,/(b\neq a)\wedge(f\in Add(b)) \\ \wedge(Add(a)\cap Del(b)=\varnothing)\wedge(Del(a)\cap Prec(b)=\varnothing)}} \neg(b\in p_i) \right) \right]$$

Cependant, contrairement au codage dans les espaces d'états, cette règle augmente fortement la complexité du codage en raison de la structure de la formule utilisée (disjonctions de conjonctions qui doivent être développée en clauses). Elle n'est donc pas utilisable en pratique.

3.2.4. Codage du graphe de planification et relation d'autorisation

La seule modification à apporter, pour prendre en compte la relation d'autorisation, consiste à remplacer, lors de la traduction, l'ensemble des actions mutuellement exclusives suivant la relation d'indépendance par celui des actions mutuellement exclusives suivant la relation d'autorisation. Ces ensembles d'actions mutuellement exclusives selon les relations d'autorisation forte ou faible peuvent être fournis par un générateur de graphe de planification autorisé comme celui de LCGP [Vidal, 2001].

Nous allons maintenant, dans le chapitre 4, présenter le système que nous avons développé pour évaluer les performances de nos différents codages.

4. Automatisation de la traduction : le traducteur TSP

Pour comparer les performances des codages sur des bases identiques, nous avons défini un langage basé sur la logique du second ordre avec domaines finis, Tunable SatPlan Language (TSPL). L'implémentation de ce langage dans le système TSP (Tunable SatPlan), permet d'automatiser la traduction de codages pour passer d'un problème de planification STRIPS à une base de clauses. Nous pouvons alors utiliser un solveur SAT pour trouver un modèle de la base de clauses et, par une traduction inverse, fournir un plan solution lorsque la base est satisfiable.

4.1. Principe de fonctionnement général de TSP

Le code source du traducteur TSP est écrit en C et les générateurs d'instances et de graphes de planification utilisés sont ceux de LCGP [Vidal, 2001] qui est écrit en Caml. Ces deux programmes s'exécutent en un temps polynomial qui dépend du codage utilisé (cf. section 4.5). TSP se compile et fonctionne soit sur un ordinateur PC sous linux, soit sur une station Sun sous Solaris. Le principe du fonctionnement de TSP est décrit dans la Figure 12. TSP prend en paramètre deux fichiers au format PDDL (Planning Domain Definition Language) qui décrivent le domaine et le problème de planification, et un fichier au format TSPL contenant les règles de codage. Les deux fichiers PDDL sont envoyés au générateur d'instances qui produit un fichier décrivant le problème instancié au format TSPL. Le traducteur utilise ensuite ce problème instancié et les règles de codage pour produire un arbre de syntaxe abstraite et un environnement permettant d'obtenir, en parcourant l'arbre, une base de clauses correspondant à la recherche d'un plan d'une longueur fixée. TSP génère la base pour une longueur minimale de plan prédéfinie et lance le solveur SAT pour essayer d'en

trouver un modèle. Cette longueur est fixée par défaut à 1 et peut être fournie avec une valeur plus élevée lorsqu'on peut déterminer la longueur minimale d'un plan solution. Si aucun modèle n'est trouvé, le traducteur incrémente la longueur du plan recherché, puis génère une nouvelle base et relance le solveur jusqu'à ce qu'il trouve un modèle ou qu'il arrive à une borne supérieure prédéfinie. Lorsqu'un modèle est trouvé, TSP utilise un algorithme linéaire de traduction inverse pour exhiber le plan solution correspondant. Lorsque les règles de codage utilisent un graphe de planification, TSP est exécuté avec un paramètre spécifique qui permet de remplacer le générateur d'instances par un générateur de graphes de planification (traits pointillés sur la Figure 12).

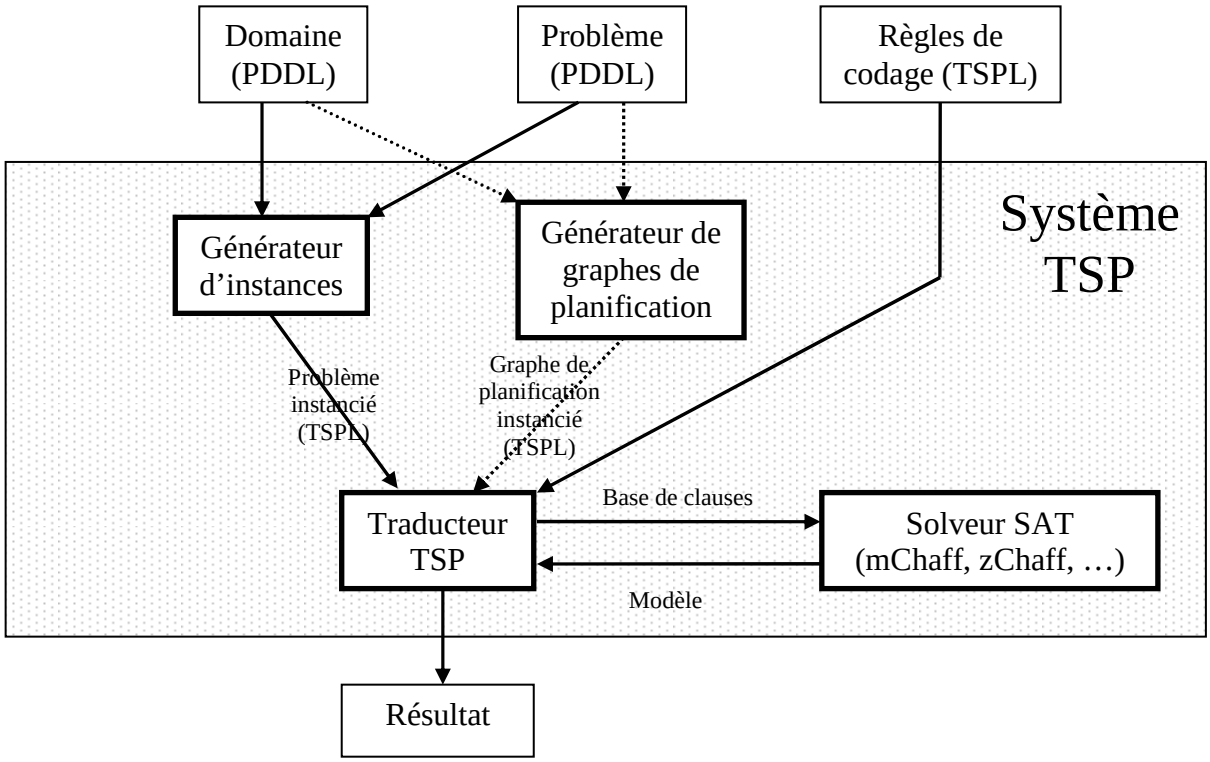


Figure 12 : principe de fonctionnement du système TSP

Comme toutes les approches basées sur SATPLAN, TSP est un planificateur semi-décidable puisque lorsqu'il n'existe aucun plan solution, la recherche ne s'arrête jamais. Cependant, lorsque la recherche est stoppée sans obtenir de solution pour une longueur de plan maximale donnée, cela prouve qu'il n'existe pas de plan solution de longueur inférieure ou égale à cette borne.

4.2. Syntaxe et sémantique de TSPL

4.2.1. Structure d'une spécification TSPL

TSPL est un langage fonctionnel préfixé qui spécifie la construction d'une base de clauses à partir d'un jeu d'ensembles prédéfinis. Une spécification écrite en TSPL est composée de deux blocs principaux. Le premier bloc, délimité par les mots clé **sets** et **endsets**, définit les ensembles utilisés pour la traduction. Dans TSP, ce premier bloc est automatiquement produit par le générateur d'instances. Le deuxième bloc, délimité par les mots clé **rules** et **endrules**, définit les règles de codage.

4.2.2. Syntaxe, sémantique et types de TSPL

La partie "définition des ensembles" consiste uniquement en une énumération alors que la partie "définition des règles" doit être considérée comme une fonction qui prend en paramètres des ensembles, des formules (en forme normale conjonctive) et des booléens et qui renvoie une base de clauses. Dans le tableau suivant, qui donne une liste des principales fonctions et instructions de TSPL, les lettres "F", "S" et "B" représentent respectivement des quantités de type formule (en CNF), ensemble et booléen. La colonne "sortie", donne le type de la valeur renvoyée s'il s'agit d'une fonction.

Syntaxe	Sémantique	Sortie
	Délimiteurs de blocs	
sets	Début du bloc de définition des ensembles	
endsets	Fin du bloc de définition des ensembles	
rules	Début du bloc de définition des règles de codage	
endrules	Fin du bloc de définition des règles de codage	
	Définitions	
s $\$v_1$…$\$v_m$ #e_1#...#e_n#	Définit l'ensemble $v_1(v_2, \ldots, v_m)$ ayant pour éléments $e_1, \ldots, e_n$ (## représente l'ensemble vide)	
f $\$v_1$…$\$v_m$	Définit un filtre sur l'ensemble $v_1(v_2, \ldots, v_m)$	
	Identificateurs	
$nom_ident	Désigne l'identificateur nom_ident : - Un identificateur lié est considéré comme une variable - Un identificateur libre est considéré comme une constante - Les entiers naturels sont codés par les identificateurs $0, $1, $2, … - Des identificateurs spéciaux agissant comme opérateurs sont également prédéfinis	
	Fonctions de construction de bases de clauses	
true	Codage de la base de clauses {}	F
false	Codage de la base de clauses $\{\perp\}$	F
p $\$v_1$…$\$v_m$	Codage du prédicat $v_1(v_2, \ldots, v_m)$	F
~p $\$v_1$…$\$v_m$	Codage de la négation du prédicat $v_1(v_2, \ldots, v_m)$	F
and F1 F2	Codage de F1 $\wedge$ F2	F
or F1 F2	Codage de F1 $\vee$ F2	F
imp F1 F2	Codage de F1 $\rightarrow$ F2	F
andset $v S F($v)	Codage de la conjonction des bases de clauses F($v) pour $v $\in$ S	F
orset $v S F($v)	Codage de la disjonction des bases de clauses F($v) pour $v $\in$ S	F
if B F1 F2	Renvoie F1 si B est vraie, F2 sinon	F
	Fonctions de construction d'ensembles	
s $\$v_1$…$\$v_m$	Renvoie l'ensemble $v_1(v_2, \ldots, v_m)$	S
union S1 S2	S1 $\cup$ S2	S
inter S1 S2	S1 $\cap$ S2	S
minus S1 S2	S1 – S2	S
such $v S B($v)	Ensemble des éléments $v $\in$ S tels que B($v) soit vraie	S
if B S1 S2	Renvoie S1 si B est vraie, S2 sinon	S
upperset $v S	Ensemble des éléments de S strictement supérieurs à $v (l'ordre total utilisé sur S est l'ordre d'énumération des éléments)	S
counter $\$n_1$ to $\$n_2$	Ensemble ordonné des entiers naturels de n_1 à n_2. $\$n_1$ et $\$n_2$ peuvent être remplacés par des suites d'identificateurs spéciaux produisant des entiers naturels.	S

Fonctions de construction de booléens		
true	Valeur de vérité Vrai	B
false	Valeur de vérité Faux	B
member $v S	Vrai ssi $v ∈ S	B
subset S1 S2	Vrai ssi S1 ⊆ S2	B
equal v_1$ v_2$	Vrai ssi v_1$ = v_2$	B
empty S	Vrai ssi l'ensemble S est vide	B
and B1 B2	B1 ∧ B2	B
or B1 B2	B1 ∨ B2	B
not B	¬B	B

Les identificateurs peuvent, comme les fonctions, avoir des types différents. Il existe trois types d'identificateurs : les symboles, les entiers naturels, et les opérateurs spéciaux :

- Les symboles peuvent être composés de lettres non accentuées de l'alphabet latin, de chiffres (non exclusivement), et des caractères "_" et "-". Ce sont des identificateurs substituables qui se comportent comme des variables.
- Les entiers naturels sont composés exclusivement de chiffres. Ils ne sont pas directement substituables mais certains opérateurs spéciaux peuvent agir sur eux.
- Les opérateurs spéciaux peuvent avoir une arité variable et, lorsqu'elle n'est pas nulle, ils agissent directement sur les identificateurs qui les suivent dans une suite d'identificateurs. Les principaux opérateurs spéciaux sont :
    - $nat_succ : fonction successeur (Nat → Nat)
    - $nat_pred : fonction prédécesseur (Nat → Nat)
    - $nat_add : fonction addition (Nat x Nat → Nat)
    - $nat_mult : fonction multiplication (Nat x Nat → Nat)

Exemple : lors d'une traduction, $h$nat_add$2$3 qui représente h(2+3) est remplacé par h5 qui représente h(5).

4.2.3. Données prédéfinies par TSP

Lorsqu'on utilise TSP, le générateur d'instances définit les ensembles qui correspondent au codage sous forme STRIPS du problème. Les ensembles suivants peuvent donc être utilisés dans la partie **rules** … **endrules** de la spécification en TSPL :

- s$F : ensemble des fluents ;
- s$O : ensemble des opérateurs ;
- s$I : ensemble des fluents de l'état initial ;
- s$G : ensemble des fluents de l'état but ;
- s$Prec$nom_op : ensemble des préconditions de l'opérateur nom_op ;
- sAddnom_op : ensemble des ajouts de l'opérateur nom_op ;
- sDelnom_op : ensemble des retraits de l'opérateur nom_op ;
- s$Fp : ensemble des fluents $f ∈ s$F tels que ∃$a ∈ s$O et $f ∈ s$Prec$a ;

- s$Fa : ensemble des fluents $f ∈ s$F tels que ∃$a ∈ s$O et $f ∈ s$Add$a ;
- s$Fd : ensemble des fluents $f ∈ s$F tels que ∃$a ∈ s$O et $f ∈ s$Del$a.

La variable $length est également prédéfinie. Elle représente la longueur du plan recherché et elle est automatiquement incrémentée par TSP.

Lorsque TSP est utilisé avec le générateur de graphe de planification, les ensembles précédents ne sont plus définis et ils sont remplacés par les suivants :

- s$NoeudsInit : ensemble des noeuds fluents de l'état initial ;
- s$NoeudsBut : ensemble des noeuds fluents de l'état but ;
- s$NoeudsF : ensemble des noeuds fluents du graphe de planification ;
- s$NoeudsA : ensemble des noeuds actions du graphe de planification ;
- s$ArcsPrec$noeud_action : ensemble des noeuds fluents préconditions du nœud action $noeud_action ;
- s$ArcsAdd$noeud_action : ensemble des noeuds fluents ajouts du nœud action $noeud_action ;
- s$MutexA$noeud_action : ensemble des nœuds actions mutex avec le nœud action $noeud_action.

Si aucune solution n'est trouvée, le graphe est automatiquement étendu d'un niveau supplémentaire par TSP.

4.3. Fonctionnement interne du traducteur TSP

Le module de traduction de TSP utilise l'analyseur syntaxique LALR(1) YACC pour produire un arbre de syntaxe abstraite qui représente les règles de codage et un environnement contenant les noms d'ensembles et les listes d'éléments correspondantes. La grammaire utilisée pour l'analyse syntaxique est donnée ci-après :

```
<tspl_spec> → <sets> <rules>

<sets> → sets <defsets> endsets

<defsets> → λ
<defsets> → <defsets> s <suitevar> ##
<defsets> → <defsets> s
              <suitevar> #ident# <defoneset>

<defoneset> → λ
<defoneset> → <defoneset> ident#

<rules> → rules <atom> <suiterules> endrules
<rules> → rules <aconj> <suiterules> endrules
<rules> → rules <adisj> <suiterules> endrules
<rules> → rules <conj> <suiterules> endrules

<suiterules> → λ
<suiterules> → <atom> <suiterules>
<suiterules> → <aconj> <suiterules>
<suiterules> → <adisj> <suiterules>
<suiterules> → <conj> <suiterules>

<conj> → and <atom> <conj>
<conj> → and <conj> <atom>
<conj> → and <aconj> <adisj>
<conj> → and <adisj> <aconj>
<conj> → and <conj> <adisj>
<conj> → and <adisj> <conj>
<conj> → and <atom> <adisj>
<conj> → and <adisj> <atom>
<conj> → and <adisj> <adisj>
<conj> → or <atom> <aconj>
<conj> → or <aconj> <atom>
<conj> → or <adisj> <aconj>
<conj> → or <aconj> <adisj>
<conj> → or <atom> <conj>
<conj> → or <conj> <atom>
<conj> → or <adisj> <conj>
<conj> → or <conj> <adisj>
<conj> → imp <aconj> <aconj>
```

```
<conj> → imp <atom> <aconj>
<conj> → imp <aconj> <conj>
<conj> → imp <atom> <conj>
<conj> → imp <adisj> <atom>
<conj> → imp <adisj> <adisj>
<conj> → andset $ident <set> <adisj>
<conj> → andset $ident <set> <conj>
<conj> → if <boolexp> <atom> <conj>
<conj> → if <boolexp> <conj> <atom>
<conj> → if <boolexp> <aconj> <conj>
<conj> → if <boolexp> <conj> <aconj>
<conj> → if <boolexp> <adisj> <conj>
<conj> → if <boolexp> <conj> <adisj>
<conj> → if <boolexp> <conj> <conj>

<adisj> → or <atom> <atom>
<adisj> → or <atom> <adisj>
<adisj> → or <adisj> <atom>
<adisj> → or <adisj> <adisj>
<adisj> → imp <atom> <atom>
<adisj> → imp <atom> <adisj>
<adisj> → imp <aconj> <atom>
<adisj> → imp <aconj> <adisj>
<adisj> → orset $ident <set> <atom>
<adisj> → orset $ident <set> <adisj>
<adisj> → if <boolexp> <atom> <adisj>
<adisj> → if <boolexp> <adisj> <atom>
<adisj> → if <boolexp> <adisj> <adisj>

<aconj> → and <atom> <atom>
<aconj> → and <atom> <aconj>
<aconj> → and <aconj> <atom>
<aconj> → and <aconj> <aconj>
<aconj> → andset $ident <set> <atom>
<aconj> → andset $ident <set> <aconj>
<aconj> → if <boolexp> <atom> <aconj>
<aconj> → if <boolexp> <aconj> <atom>
<aconj> → if <boolexp> <aconj> <aconj>

<atom> → p <suitevar>
<atom> → ~p <suitevar>
<atom> → true
<atom> → false
<atom> → if <boolexp> <atom> <atom>

<suitevar> → $ident
<suitevar> → <suitevar> $ident

<set> → s
<set> → s <suitevar>
<set> → union <set> <set>
<set> → inter <set> <set>
<set> → minus <set> <set>
<set> → such $ident <set> <boolexp>
<set> → counter <begin_counter> to <suitevar>
<set> → upperset $ident <set>
<set> → if <boolexp> <set> <set>

<begin_counter> → <suitevar>

<boolexp> → true
<boolexp> → false
<boolexp> → member $ident <set>
<boolexp> → subset <set> <set>
<boolexp> → equal $ident1 $ident2
<boolexp> → empty <set>
<boolexp> → and <boolexp> <boolexp>
<boolexp> → or <boolexp> <boolexp>
<boolexp> → imp <boolexp> <boolexp>
<boolexp> → not <boolexp>
```

Une fois l'analyse syntaxique terminée, l'arbre de syntaxe abstraite est parcouru en profondeur d'abord en constituant comme attribut hérité une liste de substitutions pour les identificateurs liés et comme attribut synthétisé une base de clauses. La grammaire précédente, augmentée de ces deux attributs, est attribuée à gauche ce qui permet d'obtenir le résultat recherché en un seul parcours de l'arbre.

4.4. Codage des traductions en TSPL

Voici ici un exemple de codage dans les espaces d'états avec frames-axiomes explicatifs MK99 (cf. section 2.4) sous sa forme TSPL. Pour l'ensemble des codages on pourra se reporter à [Maris, Régnier, Rodriguez, Vidal, 2004].

```
(rules

(andset $f s$I p$f$0)
(andset $f minus s$F s$I ~p$f$0)
(andset $f s$G p$f$length)

(andset $i (counter $1 to $length)
        (andset $a s$O
        (or  ~p$a$i
          (and (and
             (andset $f s$Prec$a p$f$nat_pred$i)
             (andset $f s$Add$a p$f$i))
             (andset $f s$Del$a ~p$f$i)))))

(andset $i (counter $1 to $length)
   (andset $f s$F
      (or   (or ~p$f$nat_pred$i p$f$i)
         (orset  $a
                 (such $v s$O (member $f s$Del$v))
                 p$a$i))))
(andset $i (counter $1 to $length)
   (andset $f s$F
      (or   (or p$f$nat_pred$i ~p$f$i)
         (orset  $a
                 (such $v s$O (member $f s$Add$v))
                 p$a$i))))

(andset $i (counter $1 to $length)
   (andset $a1 s$O
      (andset $f s$Prec$a1
         (andset $a2
                 (such $v s$O (and   (not (equal $v $a1))
                 (member $f s$Del$v)))
                 (and ~p$a1$i ~p$a2$i)))))
endrules)
```

Nous allons maintenant donner, pour chacun des codages, une estimation maximale de la complexité en temps de son interprétation par TSP et de la taille de la base de clauses produite.

4.5. Etude de la complexité des codages

La résolution du problème de planification SAT encodé est NP-complète puisqu'elle consiste en une recherche de modèle d'une base de clauses. Par contre, la complexité en temps dans le pire des cas de la traduction du problème de planification original en base de clause est polynomiale et dépend du codage utilisé. Les ensembles de fluents qui interviennent dans les préconditions, les ajouts et les retraits des opérateurs peuvent être bornés et les deux principaux paramètres à prendre en compte dans le calcul des complexités sont le nombre k de niveaux du plan recherché et le nombre n d'opérateurs instanciés. Dans les espaces d'états et dans les espaces de plans, pour un même codage, la taille de la base de clause et la complexité en temps de sa traduction sont du même dégré.

- Les codages dans les espaces d'états produisent des bases de clauses de taille $O(k.n^2)$ à cause de la règle 4 qui produit une clause par paire d'actions et pour chaque niveau. Les autres règles produisent un codage linéaire.

- Le principe d'exclusion mutuelle par paires d'actions étant le même, les parties communes des codages dans les espaces de plans produisent également des bases de clauses de taille $O(k.n^2)$ à cause des règles interdisant plus d'une action par étape pour MK99 ou interdisant les actions qui ne sont pas indépendantes dans une même étape pour V01 et LCP. La taille globale des codages dans les espaces de plan est cubique par rapport à la longueur du plan recherché car, pour chacun de ces codages, les règles spécifiques produisent $O(k^3)$ clauses. Cette taille est également quadratique par rapport au nombre d'opérateurs en raison du codage de la partie commune $O(k.n^2)$.
- Pour les codages du graphe de planification, le nombre de nœuds d'actions est borné par k.n et le nombre d'exclusions mutuelles pour chaque nœud d'action est borné par n. D'après la règle de gestion des exclusions mutuelles (règle 4 pour KS99 et règle 3 pour V01), le temps de traduction du codages est donc $O(k^2.n^3)$ et la taille de la base de clause produite est $O(k.n^2)$. Le temps de traduction de ces codages est donc cubique par rapport au nombre d'opérateurs ce qui ne permet pas de les tester sur des problèmes de taille importante comme nous le verrons dans le chapitre suivant.

Nous allons maintenant, dans le chapitre suivant, détailler les résultats expérimentaux que nous avons obtenus en comparant les performances des différents codages.

5. Etude comparative

Dans cette partie nous étudions les performances des différentes traductions proposées. Nos tests ont été effectués à partir de problèmes classiques de planification, avec un PC intégrant un CPU AMD Athlon 500 et 320 Mo SDRAM pour les codages dans les espaces d'états et de plans (sections 5.2 et 5.3). Le système d'exploitation utilisé est Linux RedHat 6.1. Pour les tests des codages des graphes de planification (section 5.4), le PC utilisé intègre un CPU Intel Celeron 2,60 Ghz et 256 Mo DDR. Le système d'exploitation utilisé est Linux Mandrake 9.2.

Pour ne pas alourdir la lecture, les résultats détaillés sont systématiquement donnés en annexe. Nous ne présentons ici que les graphiques correspondants. Dans les tableaux, le caractère "-" indique que le problème n'a pas été résolu en moins de 3600 secondes et "*" signifie que l'espace mémoire alloué a été insuffisant pour permettre sa résolution. Les différents codages sont désignés par les noms suivants :

1. Codages dans les espaces d'états :
    - MK99 (1) et V01 (1) : espaces d'états avec *frame-axiomes explicatifs*.
    - MK99 (2) et V01 (2) : espaces d'états avec *no-ops*.
    - LCS (*Least Committed State-space encoding*) : espaces d'états avec *frame-axiomes explicatifs* et *relation d'autorisation*.
2. Codages dans les espaces de plans :
    - MK99 (3) et V01 (3) : espaces de plans par *liens causaux*, *protection d'intervalles* et *ordre partiel*.
    - MK99 (4) et V01 (4) : espaces de par *liens causaux*, *protection d'intervalles* et *étapes contiguës*.
    - MK99 (5) et V01 (5) : espaces de plans avec *white knight*.

- LCP (*Least Committed Plan-space encoding*) : espaces de plans avec white knight et *relation d'autorisation*.

3. Codages des graphes de planification :
   - KS99 (6) et V01 (6) : graphes de planification avec *relation d'indépendance*.
   - LCG-KS99 et LCG-V01 (Least Committed Graphplan encoding) : graphes de planification avec relation d'autorisation.

5.1. Les benchmarks utilisés

Ce sont des benchmarks classiques utilisés pour les tests des compétitions AIPS. Ils présentent des caractéristiques différentes qui les rendent complémentaires pour notre étude.

- **Domaine du Gripper :** dans ce domaine, le robot, équipé de p pinces, doit déplacer n balles d'une table vers une autre. Trois opérateurs sont utilisables : Prendre une balle avec une pince, Poser une balle tenue dans une pince et se Déplacer d'une table à une autre. Les problèmes généraux à n balles et p pinces comportent beaucoup de symétries : il y a un grand nombre d'actions applicables à chaque état et le planificateur peut donc essayer de saisir chacune des différentes balles disponibles alors qu'elles sont identiques les unes aux autres. Les plans solutions produits comportent des suites de séquences d'actions <Prendre, Déplacer, Poser>.

- **Domaine du Ferry :** dans ce domaine, qui est équivalent à celui du Gripper à une pince, il s'agit de déplacer n voitures d'une berge à l'autre en utilisant un ferry qui ne peut contenir qu'une voiture. Trois opérateurs sont utilisables : Embarquer une voiture, la Débarquer, Traverser d'une rive à l'autre. Ce domaine est fortement symétrique : il y a un grand nombre d'actions applicables à chaque état, le planificateur pouvant essayer d'embarquer chacune des différentes voitures disponibles alors qu'elles sont identiques les unes aux autres. Les plans solutions produits comportent des suites de séquences d'actions <Embarquer, Traverser, Débarquer>.

- **Blocks-world :** dans ce domaine, plus contraint que les précédents, le système doit construire, à partir d'un état initial préalablement défini, un ensemble de tours en déplaçant des cubes distincts. La taille de l'espace de recherche est importante (pour une pile de n cubes, il existe n! façons de les agencer) et la taille de l'espace d'états est bornée par 2p où p est le nombre de propositions logiques (ordre 0). Même en tenant compte des liens sémantiques qui existent entre les propositions, pour n cubes, le nombre d'états reste supérieur à n!. Ce domaine comporte des symétries en moins grand nombre que les précédents domaines mais il contient beaucoup d'états de même forme (états identiques à une permutation du nom des cubes près). Dans la représentation classique, trois prédicats (Sur, Sur-Table, Libre) sont nécessaires pour sa représentation STRIPS et trois opérateurs sont possibles : DéplacerDeTable(x,y), DéplacerSurTable(x,y), Déplacer(x,y,z). Dans la représentation du planificateur Prodigy, ce sont six opérateurs qui sont utilisés.

- **Domaine Logistics :** ce domaine décrit le transport de paquets (en nombre variable) dans des villes par des avions et des camions qui ont des capacités illimitées. Les plans contiennent beaucoup de parallélisme et sont moins contraints, comparativement au nombre d'actions qu'ils contiennent, que dans d'autres domaines (par exemple blocks-word). Ce domaine, du point de vue des symétries qu'il contient est proche de celui des cubes. En effet, si on considère chaque ville comme étant une pile de paquets, c'est un problème de cubes, à la seule différence près que l'ordre des paquets n'a pas d'importance (comme dans les domaines du Gripper et du Ferry).

- **Domaine des Tours de Hanoï :** dans ce domaine très contraint, un problème est constitué par p piques et n disques de taille décroissante empilés en pyramide sur une des piques. La pyramide doit être déplacée sur une autre pique. Les disques ne peuvent être transférés sur une autre pique qu'en respectant la contrainte qu'un disque doit toujours être empilé sur un disque plus large que lui. Ce domaine peut être modélisé par un seul opérateur Déplacer(x,y,z). Les symétries du domaine sont peu nombreuses, puisque seules les piques peuvent être échangées et non les disques puisqu'ils sont de tailles différentes.

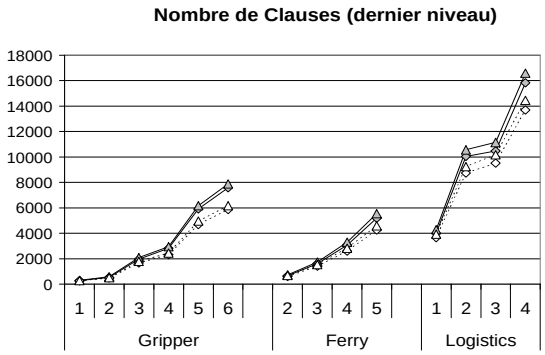
Dans un premier temps (section 5.2) nous allons comparer les codages MK99 et V01 dans les espaces d'états et dans les espaces de plans partiels (décrits dans les sections 2.4 et 2.5). Nous comparerons ensuite, dans la section 5.3, les meilleurs de ces codages avec nos codages LCS et LCP qui introduisent la relation d'autorisation (cf. section 3.2), tout d'abord par recherche incrémentale de plan-solution, puis par l'étude du niveau pour lequel un plan-solution est retourné. Enfin, dans la section 5.4, nous étudierons les différents codages du graphe de planification KS99 et V01 qui utilisent soit la relation d'indépendance, soit la relation d'autorisation.

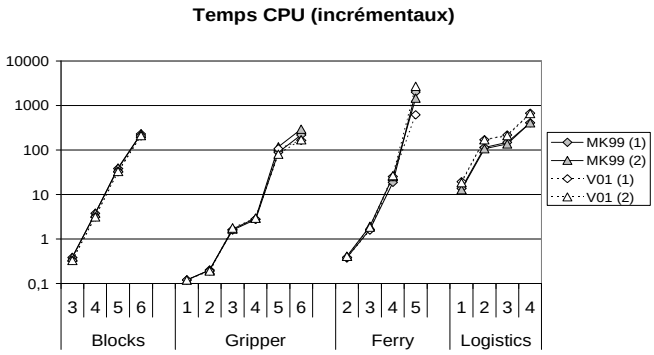
5.2. Etude comparative des traductions MK99 et V01

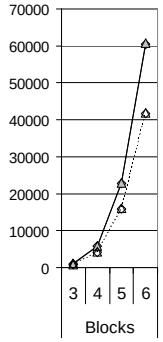
Dans cette section, nous utilisons TSP de manière incrémentale. Nous considérons le cumul des temps de recherche d'un plan solution du niveau 1 jusqu'au niveau pour lequel le plan est effectivement trouvé. Les temps CPU indiqués représentent la somme des temps de traduction et de recherche.

5.2.1. Tests des codages dans les espaces d'états

Les codages que nous utilisons pour cette partie des tests sont les codages dans les espaces d'états avec frame-axiomes explicatifs (Codage 1, page 153) et avec no-ops (Codage 2, page 153) sous forme TSPL.





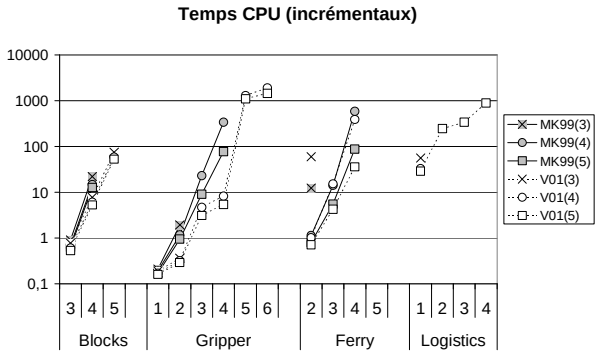


On peut constater que les codages V01 réduisent toujours le nombre de clauses par rapport aux codages MK99, en moyenne de 23%. De plus, le nombre de variables est en général minimisé par le codage V01(1) : **espaces d'états avec *frame-axiomes explicatifs***. Ce codage est le plus performant sur la moitié des problèmes comme par exemple sur le domaine des cubes. Cependant, il existe des domaines, en particulier Logistics, pour lesquels les temps CPU (traduction + recherche) sont meilleurs de 36% pour les codages MK99. Pour d'autres domaines (Gripper, Ferry), les performances dépendent du problème. Les codages MK99 ou V01 ne peuvent donc être départagés sur la simple considération du temps CPU.

Le temps de recherche n'étant pas ici un critère pertinent pour la détermination du meilleur codage, nous retenons pour la suite de notre étude le codage produisant la base de clause la plus compacte V01(1).

5.2.2. Tests des codages dans les espaces de plans

Les codages que nous utilisons pour cette partie des tests sont les codages dans les espaces de plans formés à partir de la partie commune du codage MK99 (Codage 3, page 154) ou V01 (Codage 4, page 154), et de la partie propre à chaque codage (Codage 5, Codage 6 et Codage 7, pages 155-156).



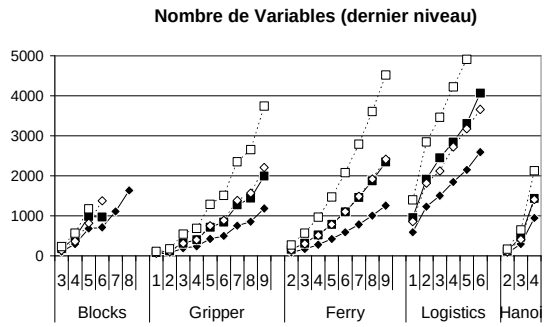
L'ordre de performances entre les trois types de codages est, du plus lent au plus rapide : (3), (4), (5). Contrairement à ce qui se passait avec les codages dans les espaces d'états, l'étude des performances des codages dans les espaces de plans démontre clairement l'avantage des traductions V01 sur les codages MK99 et dans plusieurs cas, le codage V01(3) s'avère même meilleur que MK99(5) (Annexe 3, Tableau 4 et Tableau 6). Quoi qu'il en soit, pour un même type de codage dans les espaces de plans, V01(k) est très souvent bien meilleur que MK99(k). Ce résultat est dû au parallélisme autorisé par les codages V01, puisque le nombre de niveaux peut être considérablement réduit. Même dans le domaine du Ferry, qui n'est pas du tout parallèle, V01 est encore généralement plus efficace.

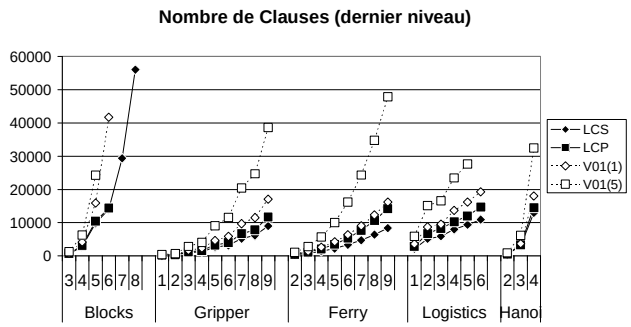
Les codages V01 s'avérant plus compacts et généralement plus performants, ce sont donc eux que nous allons conserver pour la suite de notre étude. Dans un premier temps, nous allons évaluer les gains apportés par l'introduction de la relation d'autorisation.

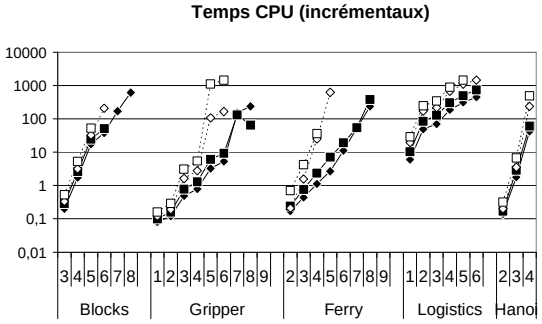
5.3. Etude comparative des traductions V01, LCS et LCP (introduction de l'autorisation)

5.3.1. Etude par recherche incrémentale du plan solution

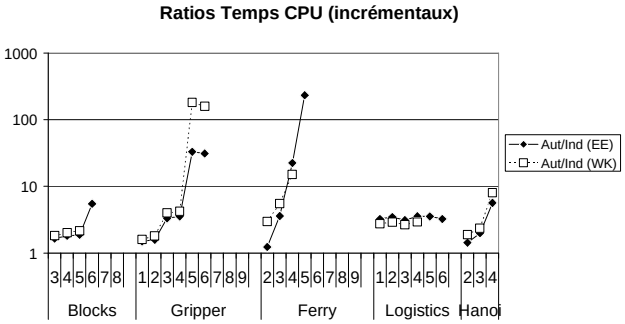
Dans cette partie, nous utilisons toujours TSP de manière incrémentale. Nous considérons donc toujours le cumul des temps de recherche d'un plan solution du niveau 1 jusqu'au niveau pour lequel le plan est effectivement trouvé.







L'introduction de la relation d'autorisation améliore systématiquement la compacité (en nombre de variables et de clauses pour des problèmes identiques) et les performances (en temps CPU) de façon conséquente. Ceci se vérifie pour les deux types de codages, espaces d'états et espaces de plans. En effet, pour tous les problèmes que nous avons testés, l'autorisation réduit fortement le nombre de niveaux pour la recherche d'un plan solution, ce qui permet de traiter des problèmes plus difficiles et en plus grand nombre.



L'analyse des ratios des temps CPU entre les codages V01 et les codages LCS et LCP confirme l'amélioration apportée par l'utilisation de la relation d'autorisation. En outre, elle montre également que cette amélioration croît en général fortement avec la complexité du problème (échelle logarithmique).

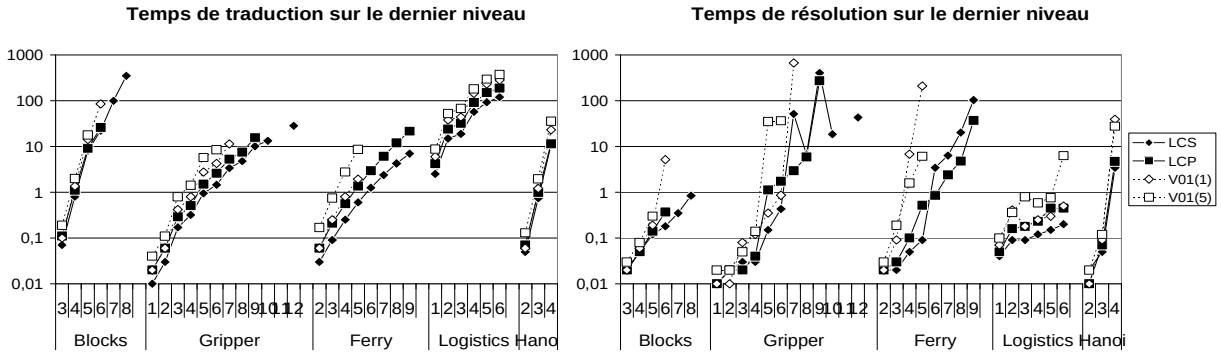
Parmi tous les codages, LCS est donc le plus efficace, en termes de compacité et de performances, suivi par LCP, V01(1) puis V01(5).

Pour poursuivre notre étude, nous allons maintenant prendre en compte uniquement les performances des codages au dernier niveau (celui pour lequel un plan solution est trouvé). En effet, on peut considérer qu'une borne inférieure proche de cette information peut être fournie (par exemple par la construction, en temps polynomial, du graphe de planification associé au

problème posé). Le temps de recherche d'une solution évolue alors d'une manière similaire pour une recherche incrémentale et une recherche au dernier niveau.

5.3.2. Etude des performances sur le dernier niveau

Pour les différents codages, nous considérerons séparément les temps de traduction et de résolution en fournissant directement au traducteur le niveau pour lequel un plan solution est trouvé.

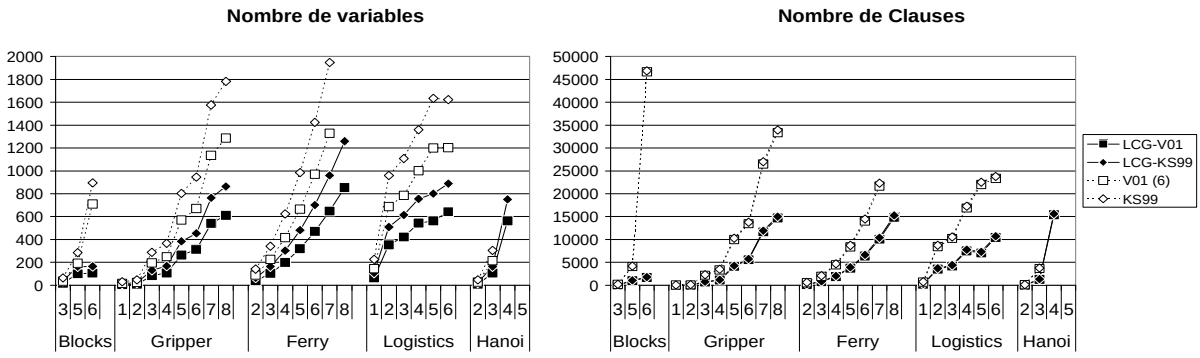


Ces tests, qui prennent uniquement en compte le dernier niveau, permettent de traiter plus de problèmes et confirment les performances du codage LCS (Cf. Annexe 3, Tableau 9). Il est à la fois le plus compact, et le plus performant en temps de traduction, de résolution et temps de recherche. Les codages qui viennent ensuite, sont LCP, V01(1) et enfin V01(5). On peut également constater que tous les codages avec autorisation sont meilleurs que tous ceux avec indépendance. Enfin, pour les codages examinés, le temps global (traduction + résolution) est d'abord lié à la structure des problèmes de planification traités : il diffère en effet de manière importante en fonction des domaines pour des bases de clauses de taille similaire (taille estimée par le produit nombre de variables × nombre de clauses). Pour chacun des domaines, le temps global évolue ensuite d'une manière similaire à la taille de la base de clauses du problème (toujours estimée par le produit nombre de variables × nombre de clauses).

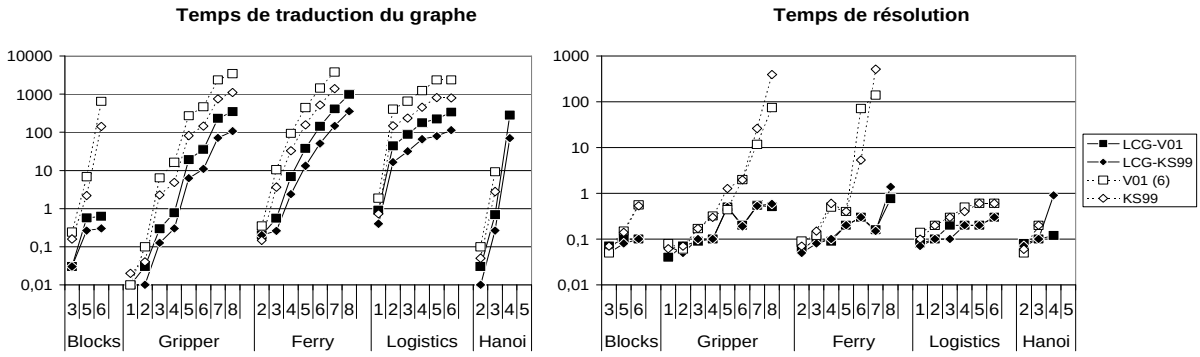
Les codages qui intègrent la relation d'autorisation sont les meilleurs. Parmi eux, le codage LCS est le plus compact et le plus performant en temps de traduction, de résolution, et c'est celui qui permet de traiter le plus grand nombre de problèmes.

5.4. Etude comparative des traductions KS99 et V01

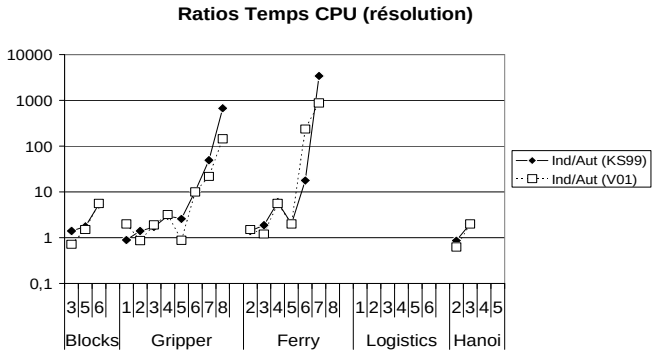
Les codages que nous utilisons pour cette partie des tests sont les codages des graphes de planification avec relation d'indépendance (Codage 8 et Codage 9, page 157) ou avec relation d'autorisation (cf. section 3.2, page 46) sous forme TSPL.



Le codage V01, qui ne prend en compte que les nœuds d'actions du graphe, permet une réduction de 29% en moyenne du nombre de variables par rapport au codage KS99 qui prend également en compte les nœuds de fluents. La réduction du nombre de clauses est par contre très faible car les clauses d'exclusions mutelles provenant de la règle de codage dont le degré de complexité est le plus élevé restent les mêmes dans les deux codages. La relation d'autorisation diminuant en moyenne de 38% le nombre de niveaux nécessaires à l'obtention d'un plan-solution, les codages LCG-KS99 et LCG-V01 diminuent environ de 44% le nombre de variables et de 55% le nombre de clauses par rapport aux précédents.



Le codage V01 augmente systématiquement de façon importante le temps de traduction qui est 3 fois supérieur à celui du codage KS99. Par contre, l'analyse des temps de résolution ne permet pas de déterminer, même si V01 semble globalement plus performant, dans quelle mesure l'un de ces codages est meilleur que l'autre. Comme nous pouvions l'espérer, la relation d'autorisation permet aux codages LCG-KS99 et LCG-V01 de diminuer fortement les temps de calcul. La traduction est 6 fois plus rapide quel que soit le codage et la résolution est 60 fois plus rapide pour LCG-V01 et 150 fois plus rapide pour LCG-KS99 par rapport aux codages V01 et KS99.



Par l'analyse des ratios ci-dessus, on peut constater que le gain de temps apporté par la relation d'autorisation croît de manière exponentielle avec la difficulté des problèmes traités.

Le codage LCG-V01 est plus compact et sensiblement plus performant que LCG-KS99. Le remplacement de la relation d'indépendance par la relation d'autorisation augmente fortement les performances de ces deux codages par rapport à V01 et KS99. L'écart se creuse avec l'augmentation de la difficulté des problèmes traités.

Sur l'ensemble de ces tests, nous pouvons remarquer que les codages les plus performants sont ceux dans les espaces d'états qui utilisent la relation d'autorisation. Les performances des codages du graphe de planification, qui pourraient a priori bénéficier des avantages liés aux informations contenues dans sa structure, sont altérées par les temps de traduction.

Dans le chapitre suivant, nous allons maintenant étudier les codages récents proposés par [Rintanen, Heljanko, Niemelä, 2004, 2006], [Robinson, Gretton, Pham, Sattar, 2008] et nous en proposerons plusieurs améliorations.

6. Etude et amélioration de codages récents

Très récemment plusieurs nouvelles méthodes de codage ont été proposées qui améliorent l'état de l'art. Dans ce chapitre nous proposons d'utiliser les idées précédentes pour améliorer ces méthodes. Nous analysons d'abord les codages basés sur les sémantiques ∀-Step et ∃-Step de [Rintanen, Heljanko, Niemelä, 2004, 2006] et le codage compact basé sur la relation d'indépendance et utilisant un découpage des opérateurs de [Robinson, Gretton, Pham, Sattar, 2008]. Nous proposons ensuite un nouveau codage compact basé sur la relation d'autorisation. Les améliorations que nous proposons n'ont pas été validées avec TSP car leur codage en TSPL nécessite son adaptation pour la prise en compte du découpage d'opérateurs.

6.1. Nomenclature des différents codages

Nous emploierons les notations suivantes : RHN04 pour dénoter les codages de [Rintanen, Heljanko, Niemelä, 2004, 2006] et RGPS08 pour le codage compact de [Robinson, Gretton, Pham, Sattar, 2008]. Nous proposons également un nouveau codage combinant l'utilisation de la relation d'autorisation et les avantages en terme de compacité de RGPS04. Les différents codages que nous utiliserons seront donc désignés par les abréviations suivantes :

1. Codages dans les espaces d'états :
    - RHN04 : espaces d'états avec *frame-axiomes explicatifs*, sémantique ∀-Step (équivalente à la relation d'indépendance), effets conditionnels,
        - RHN04 (1), espace quadratique,
        - RHN04 (2), espace en O(n.log n),
        - RHN04 (3), espace linéaire.
    - LC-RHN04 : espaces d'états avec *frame-axiomes explicatifs*, sémantique ∃-Step (équivalente à la relation d'autorisation forte), effets conditionnels,
        - LC-RHN04 (1), espace quadratique,
        - LC-RHN04 (2), espace en O(n.log n),

- LC-RHN04 (3), espace linéaire.
  - LLCS (*Lifted Least Committed State-space encoding*) : codage compact dans les espaces d'états avec *frame-axiomes explicatifs* et *relation d'autorisation*.

2. Codages des graphes de planification :
   - RGPS08 : codages compacts des graphes de planification.

6.2. Codages avec indépendance et sémantique ∀-Step

Le premier codage proposé par [Rintanen, Heljanko, Niemelä, 2004, 2006] utilise pour l'application parallèle des opérateurs une sémantique appelée ∀-step qui correspond au codage de l'indépendance entre actions.

Les améliorations apportées par rapport aux codages existants sont, d'une part la prise en compte d'opérateurs avec effets conditionnels et, d'autre part, la réduction de l'ordre de grandeur de la taille du codage qui passe de quadratique à linéaire par rapport au nombre d'opérateurs. Dans la suite, contrairement à ce que proposent [Rintanen, Heljanko, Niemelä, 2004, 2006], nous nous placerons sous l'hypothèse du monde clos. Les ensembles définissant les opérateurs et les états sont des ensembles de fluents.

Définition 34 (opérateur avec effets conditionnels)

Un opérateur o avec effets conditionnels est un opérateur pour lequel les ensembles d'ajouts et de retraits (effets inconditionnels) sont complétés par des ensembles d'ajouts conditionnels CondAdd(o) et de retraits conditionnels CondDel(o). Ce sont des ensembles de paires $f \triangleright d$ où f et d sont des fluents.

Les effets actifs (ajouts et retraits) d'un opérateur avec effets conditionnels o sur un état s sont :

$$ActiveAdd(o,E) = Add(o) \cup \bigcup \{d \,/\, f \triangleright d \in CondAdd(o),\, f \in E\}$$

$$ActiveDel(o,E) = Del(o) \cup \bigcup \{d \,/\, f \triangleright d \in CondDel(o),\, f \in E\}$$

L'opérateur o est applicable dans s, si Prec(o) $\subseteq$ E et ActiveAdd(o,E) $\cap$ ActiveDel(o,E) = $\varnothing$. Dans ce cas, on définit : E $\uparrow$ o = E' comme l'état unique obtenu en rendant vrais les effets actifs de o et en conservant à vrai les variables d'état qui l'étaient déjà et qui n'interviennent pas dans les effets actifs de o.

Définition 35 (plans ∀-Step)

Soit un ensemble d'opérateurs O et un état initial I. Un plan ∀-step pour O et I est une séquence S=<Q_0, …, Q_{k-1},> d'ensembles d'opérateurs pour k ≥ 0 telle qu'il existe une séquence d'états E_0, …, E_k (l'exécution de S) telle que :

- E_0 = I
- ∀i ∈ {0, …, k-1} et pour tout ordonnancement total o_1, …, o_n, de Q_i, l'application successive de o_1, …, o_n, sur E_i est possible et que l'état résultant $((\ldots((E_i \uparrow o_1) \uparrow o_2)\ldots) \uparrow o_n)$ est égal à E_{i+1}.

Les codages de [Rintanen, Heljanko, Niemelä, 2004, 2006] avec indépendance et effets conditionnels apportent une amélioration aux codages précédents grace à la prise en compte d'effets conditionnels pour les actions. Nous allons intégrer les améliorations apportées par [Vidal, 2001], en les étendant aux effets conditionnels. Pour ces codages, les ensembles *Fp*, *Fa* et *Fd* seront redéfinis ainsi :

- *Fp* : l'ensemble des fluents apparaissant dans les préconditions des actions (préconditions).
- *Fa* : l'ensemble des fluents qui apparaissent dans les ajouts ou les ajouts conditionnels des actions (ajouts).
- *Fd* : l'ensemble des fluents qui apparaissent dans les retraits ou les retraits conditionnels des actions (retraits).

Les règles du codage V01(1) peuvent être réécrites ainsi :

- Une partie est ajoutée à la règle 2 pour prendre en compte les effets conditionnels :

$$2.2 \quad \bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \bigwedge_{(c\rhd f)\in CondAdd(a)} \big((a(i)\wedge c(i-1))\Rightarrow f(i)\big)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \bigwedge_{(c\rhd f)\in CondDel(a)} \big((a(i)\wedge c(i-1))\Rightarrow \neg f(i)\big)$$

- Les règles des frame-axiomes de retrait et d'ajout sont modifiées pour prendre en compte les effets conditionnels :

$$3.1. \quad \bigwedge_{i\in[1,k]} \bigwedge_{f\in((I\cup F_a)\cap F_d)} \left(\big(f(i-1)\wedge\neg f(i)\big)\Rightarrow \left(\left(\bigvee_{a\in O/\ f\in Del(a)} a(i) \right) \vee \left(\bigvee_{a\in O/(c\rhd f)\in CondDel(a)} \big(a(i)\wedge c(i-1)\big) \right) \right) \right)$$

$$3.2. \quad \bigwedge_{i\in[1,k]} \bigwedge_{f\in(((F-I)\cup F_d)\cap F_a)} \left(\big(\neg f(i-1)\wedge f(i)\big)\Rightarrow \left(\left(\bigvee_{a\in O/\ f\in Add(a)} a(i) \right) \vee \left(\bigvee_{a\in O/(c\rhd f)\in CondAdd(a)} \big(a(i)\wedge c(i-1)\big) \right) \right) \right)$$

- La définition la plus simple pour les exclusions mutuelles est semblable à celle du codage original de [Kautz, Selman, 1992]. Comme pour le codage V01, nous modifions la quatrième règle pour n'ajouter une clause d'exclusion mutuelle entre actions que si cette dernière n'a pas déjà été ajoutée par la règle 2 qui prend en compte les effets contradictoires : l'exclusion est ajoutée si les deux actions ont une interaction croisée *et pas d'effets contradictoires* :

4. $$\bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2 / m<n \\ \wedge\left(\begin{array}{l}(Prec(a_m)\cap(Del(a_n)\cup CondDel(a_n))\neq\emptyset) \\ \vee((Del(a_m)\cup CondDel(a_m))\cap Prec(a_n)\neq\emptyset)\end{array}\right) \\ \wedge\left(\begin{array}{l}((Add(a_m)\cup CondAdd(a_m))\cap(Del(a_n)\cup CondDel(a_n))=\emptyset) \\ \vee((Del(a_m)\cup CondDel(a_m))\cap(Add(a_n)\cup CondAdd(a_n))=\emptyset)\end{array}\right)}} \left(\neg a_m(i) \vee \neg a_n(i)\right)$$

Contrairement à l'ensemble des règles précédentes qui produisent un codage de taille linéaire O(k) par rapport à la longueur du plan recherché k, cette dernière règle produit un codage quadratique O(k.n²) par rapport au nombre d'opérateurs n. [Rintanen, Heljanko, Niemelä, 2004, 2006] proposent un codage linéaire de cette dernière règle. Ce codage produit un nombre de clauses binaires pour la règle 4 égal au maximum à 6 fois le nombre d'opérateurs. Cependant, ce nouveau codage comporte un grand nombre de nouvelles variables propositionnelles, ce qui peut dégrader les temps de résolution. Ils proposent un compromis avec un codage comprenant uniquement un nombre logarithmique de nouvelles variables et un nombre de clauses en O(n.log(n)) qui améliore le codage quadratique pour le nombre de clauses et le codage linéaire en ce qui concerne le nombre de variables propositionnelles.

6.3. Codages avec autorisation et sémantiques ∃-Step / R∃-Step

Pour relaxer le parallélisme dans les codages [Rintanen, Heljanko, Niemelä, 2004, 2006] proposent un nouveau codage intégrant la linéarisation de séquences d'actions non indépendantes mais pouvant être exécutées dans un ordre donné.

Définition 36 (plans ∃-Step)

Soit un ensemble d'opérateurs O et un état initial I. Un plan ∃-step pour O et I est conjointement une séquence $T=<S_0, \ldots, S_{k-1},> \in (2^O)^k$ et une séquence d'états $E_0, \ldots, E_k$ (l'exécution de T) pour $k \geq 0$ telles que :

- $E_0 = I$
- $\forall i \in \{0, \ldots, k-1\}$ il existe un ordonnancement total $o_1 < \ldots < o_n$ de S_i tel que $E_{i+1} = ((\ldots((E_i \uparrow o_1) \uparrow o_2)\ldots) \uparrow o_n)$ soit l'état résultant de l'application successive de $o_1, \ldots, o_n$, sur E_i

Cette approche est similaire à la notre avec l'introduction de la relation d'autorisation, mais elle ne va pas aussi loin dans la relaxation du parallélisme à cause d'une contrainte sur les effets contradictoires des opérateurs. En effet, cette contrainte très forte, imposée dans l'ensemble des sémantiques proposées par [Rintanen, Heljanko, Niemelä, 2004, 2006], consiste à imposer la consistance des effets actifs d'un ensemble d'actions pour son application à un état. Pour deux opérateurs STRIPS classiques o et o' nous avons donc en particulier $Add(o) \cap Del(o') = \emptyset$ et $Add(o') \cap Del(o) = \emptyset$

Une relation binaire d'affectation est définie entre opérateurs. Lorsqu'un premier opérateur en affecte un second, ils ne peuvent faire partie d'une séquence d'un plan ∃-step dans cet ordre. La définition de l'affectation d'un opérateur o' par un opérateur o, donnée dans [Rintanen, Heljanko, Niemelä, 2006], peut être simplifiée dans le cas des opérateurs STRIPS classiques en $Prec(o') \cap Del(o) \neq \emptyset$. Cette définition nous permet de déterminer l'existence d'une relation entre affectation et autorisation forte.

Lemme 1

Soient deux opérateurs o et o'.
Alors o n'affecte pas o' si et seulement si o autorise fortement o'.

Preuve : Par définition de l'affectation, o n'affecte pas o' si et seulement si Prec(o') $\cap$ Del (o) = $\emptyset$. De plus, si nous considérons la contrainte sur les effets contradictoires permettant d'appliquer o et o' dans un même état (Add(o) $\cap$ Del (o') = $\emptyset$ et Add(o') $\cap$ Del (o) = $\emptyset$), dire que o n'affecte pas o' est donc équivalent à Add(o) $\cap$ Del (o') = $\emptyset$ et (Prec(o') $\cup$ Add(o')) $\cap$ Del (o) = $\emptyset$. Par définition de l'autorisation forte, o n'affecte pas o' est donc équivalent à o autorise fortement o'.

Pour gérer la linéarisation dans la recherche de plans $\exists$-step, [Rintanen, Heljanko, Niemelä, 2004, 2006] utilisent un graphe de désactivation défini à partir de la relation d'affectation :

Définition 37 (graphe de désactivation)

Soit π = <O, I, G> une instance de problème. Un graphe <O, X> est un graphe de désactivation pour π quand X $\subseteq$ O $\times$ O est l'ensemble des arcs orientés tels que <o, o'> $\in$ X si :

- il existe un état E tel que E est atteignable à partir de I par les opérateurs dans O et o et o' peuvent être appliqués sur E
- o affecte o'

Nous pouvons donc déduire en utilisant le Lemme 1 que la relaxation du parallélisme utilisée dans les codages $\exists$-step est la relation d'autorisation forte. En comparaison, nos derniers codages ne nécessitent pas la contrainte sur les effets contradictoires ce qui nous permet d'utiliser la relation d'autorisation faible moins contraignante et permettant une relaxation plus importante. Par ailleurs, l'intégration de la linéarisation aux codages $\exists$-step, produit un codage de taille $O(k.n^3)$. Un autre codage, plus compact, est également proposé mais il produit encore dans le pire des cas un nombre de clauses en $O(n^2.\log_2 n)$. De plus, ces deux codages imposent le remplacement de clauses binaires de la règle 4 par des clauses de taille 3. Notre approche produit un codage de taille $O(n^2)$, conserve les clauses binaires et extrait la linéarisation qui s'exécute a posteriori en temps polynomial. L'inconvénient est qu'il faut relancer la recherche lorsque la linéarisation du plan est impossible, mais ce cas ne s'est jamais présenté sur l'ensemble des benchmarks classiques.

Un codage linéaire $\exists$-step est également proposé par [Rintanen, Heljanko, Niemelä, 2004, 2006] en fixant a priori un ordre arbitraire sur les opérateurs. Ce codage exclut par ce choix arbitraire un grand nombre d'autres ordonnancements possibles des actions et ainsi, de nombreux ensembles d'opérateurs parallèles qui auraient pu être appliqués simultanément. Il donne de bonnes performances en pratique mais n'offre aucune garantie en terme de qualité pour le plan-solution, que ce soit en nombre d'opérateurs ou en nombre d'étapes.

[Wehrle, Rintanen, 2007] étendent la définition du graphe de désactivation à un graphe d'activation / désactivation qui permet d'activer des opérateurs supplémentaire sur une étape ne permettant pas de déduire leurs préconditions si des opérateurs qui les produisent sont activés. Cette approche, appelée R$\exists$-step, permet parfois d'améliorer les performances des codages $\exists$-step en diminuant le nombre d'étapes mais n'apporte pas de relaxation

supplémentaire du parallélisme au niveau des interactions entre opérateurs (comme pourrait le faire l'introduction de l'autorisation faible).

6.4. Un codage compact utilisant la relation d'autorisation faible

[Robinson, Gretton, Pham, Sattar, 2008] introduisent un nouveau codage basé sur le principe de BLACKBOX qui gère les exclusions mutuelles en utilisant la relation d'indépendance. La différence essentielle repose sur un découpage des opérateurs selon leurs arguments, découpage proposé par [Kautz, Selman, 1992] et déjà utilisé par [Ernst, Millstein, Weld, 1997] sans introduction du parallélisme. Ce découpage permet d'obtenir un codage beaucoup plus compact. Sur certains domaines des compétitions IPC ce codage se montre plus efficace que ∃-step ou R∃-step bien que nécessitant un plus grand nombre de niveaux. Serait-il alors possible de bénéficier conjointement des avantages apportés par un codage compact utilisant le découpage des opérateurs et des avantages de la linéarisation pour réduire le nombre d'étapes dans la recherche d'un plan-solution ?

Nous proposons un nouveau codage dans les espaces d'états LLCS (Lifted Least Committed State-space encoding) qui remplisse ces deux conditions en utilisant la relation d'autorisation faible.

Pour tout opérateur o, on définit une variable booléenne o[i] pour chaque instanciation d'un argument de o. Par exemple MOVE[1](A) représente le fait que le premier argument d'un opérateur MOVE est instancié avec A. Pour toute action a, Φ_a représente une conjonction de variables qui exprime l'instanciation complète de a. Pour tout fluent f, $\Phi_a(f)$ désigne le fragment de Φ_a qui est en rapport avec le fluent f.

Par exemple si $\Phi_a \equiv$ MOVE(A,B,C).
Prec(MOVE(A,B,C)) = { CLEAR(A), CLEAR(C), ON(A,B) }
Add(MOVE(A,B,C)) = { CLEAR(B), ON(A,C) }
Del(MOVE(A,B,C)) = { CLEAR(C), ON(A,B) }

On a alors :
Φ_a(CLEAR(B)) = MOVE[2](B)
Φ_a(CLEAR(C)) = MOVE[3](C)
Φ_a(ON(A,B)) = MOVE[1](A) ∧ MOVE[2](B), etc.

Les principales règles du codage LLCS sont identiques à celles de notre codage dans les espaces d'états avec frame-axiomes explicatifs et relation d'autorisation faible. La seule différence est le remplacement des actions instanciées par le fragment en rapport avec les fluents traités dans chacune des règles selon le découpage défini précédemment.

- **Etat initial et but :**

1. $$\left(\bigwedge_{f\in I} f^0\right)\wedge\left(\bigwedge_{f\in(F-I)} \neg f^0\right)\wedge\left(\bigwedge_{f\in G} f^k\right)$$

- **Préconditions et effets des actions :**

2.1 $$\bigwedge_{i\in[1,k]} \bigwedge_{a\in A} \left[\left(\bigwedge_{f\in Pre c(a)} \left(\Phi_a(f^i) \Rightarrow f^i\right)\right) \wedge \left(\bigwedge_{f\in Add(a)} \left(\Phi_a(f^i) \Rightarrow f^i\right)\right)\right]$$

2.3 $$\bigwedge_{i\in[1,k]} \bigwedge_{a\in A} \bigwedge_{f\in Del(a)} \left[\left(\Phi_a(f^i) \wedge \bigwedge_{\substack{b\in A/(b\neq a)\wedge(f\in Add(b)) \\ \wedge(Add(a)\cap Del(b)=\emptyset)\wedge(Del(a)\cap Pre c(b)=\emptyset)}} \neg\Phi_b(f^i)\right) \Rightarrow \neg f^i \right]$$

- **Frame-axiomes explicatifs :**

3.1 $$\bigwedge_{i\in[1,k]} \bigwedge_{f\in((I\cup F_a)\cap F_d)} \left[\left(f^{i-1} \wedge \neg f^i\right) \Rightarrow \left(\bigvee_{a\in A/\ f\in Del(a)} \Phi_a(f^i)\right)\right]$$

3.2 $$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(((F-I)\cup F_d)\cap F_a)} \left[\left(\neg f^{i-1} \wedge f^i\right) \Rightarrow \left(\bigvee_{a\in A/\ f\in Add(a)} \Phi_a(f^i)\right)\right]$$

- **Exclusions mutuelles (inter-opérateur et intra-opérateur) :** pour tout couple d'actions (a_m, a_n), si a_m n'autorise pas a_n du fait d'un fluent f_1 et a_n n'autorise pas a_m du fait d'un fluent f_2, alors les fragments des actions instanciées $\Phi_{am}(f_1)$ et $\Phi_{an}(f_2)$ ne peuvent être vérifiés simultanément.

5. $$\bigwedge_{i\in[1,k]} \bigwedge_{(a_m,a_n)\in A^2/\ m\leq n} \bigwedge_{\substack{(f_1,f_2)\in F_d^2/ \\ ((f_1\in Add(a_m)\cap Del(a_n))\vee(f_1\in Del(a_m)\cap Pre c(a_n))) \\ \wedge((f_2\in Del(a_m)\cap Add(a_n))\vee(f_2\in Pre c(a_m)\cap Del(a_n)))}} \left(\neg\Phi_{a_m}(f_1^i) \vee \neg\Phi_{a_n}(f_2^i)\right)$$

- **Restriction des opérateurs (optionnel) :** on peut ajouter une règle de restriction des opérateurs lorsque le problème ne permet pas une instanciation totale des opérateurs (donnée par un graphe de planification relaxé par exemple). On définit alors Arity(o) comme étant le nombre d'arguments de o. On définit Dom(o[i]) comme l'ensemble des i^{èmes} arguments possibles de o et Dom(o[j]/o[i](X)) comme l'ensemble des j^{èmes} arguments possibles de o lorsque le i^{ème} argument de o est X.

6. $$\bigwedge_{i\in[1,k]} \bigwedge_{o\in O} \bigwedge_{(m,n)\in\{1,\ldots,Arity(o)\}^2/\ m<n} \left[\begin{array}{l} \bigwedge_{X\in Dom(o[m])} \left(o^i[m](X) \Rightarrow \bigvee_{Y\in Dom(o[n]/o[m](X))} o^i[n](Y)\right) \\ \wedge \bigwedge_{Y\in Dom(o[n])} \left(o^i[n](Y) \Rightarrow \bigvee_{X\in Dom(o[m]/o[n](Y))} o^i[m](X)\right) \end{array}\right]$$

Dans ce chapitre nous avons proposé d'utiliser des restrictions sur les ensembles de fluents pour améliorer les codages basés sur les sémantiques ∀-Step et ∃-Step de [Rintanen, Heljanko, Niemelä, 2004, 2006]. Nous avons également utilisé l'idée du découpage des opérateurs introduite par [Robinson, Gretton, Pham, Sattar, 2008] et la relation d'autorisation pour proposer un codage dans les espaces d'états a priori très compact. Les améliorations que nous proposons n'ont pas été validées avec TSP car leur codage en TSPL aurait nécessité son adaptation pour la prise en compte du découpage d'opérateurs.

7. Conclusion

Notre travail sur les codages SAT de problèmes de planification nous a amené à proposer plusieurs améliorations à de nombreux codages de l'état de l'art. Nous dresserons un bilan des ces travaux dans la conclusion (cf. partie V). Après nous être intéressés à la planification SAT, nous avons travaillé à l'amélioration des techniques de planification dans le cadre temporel, toujours en utilisant des solveurs. C'est ce travail que nous présentons dans la partie suivante.

IV. Planification temporelle : le système TLP-GP

1. Introduction

Pour résoudre des problèmes de planification réels, l'une des principales difficultés à surmonter consiste à prendre en compte la dimension temporelle. En effet, de nombreux problèmes du monde réel nécessitent, pour être résolus ou exécutés plus efficacement, la prise en compte de la durée des actions, des instants d'apparition des effets ou de nécessitation des préconditions, ou encore la concurrence d'actions. La gestion d'aéroports, de gares ferroviaires, la cuisson de céramiques, la cuisine… sont quelques exemples de tels problèmes.

Même si un grand nombre de planificateurs temporels ont été mis au point puis comparés, en particulier lors des compétitions IPC, des études théoriques récentes mettent en évidence les limites des approches actuelles de la planification temporelle. [Cushing et al., 2007.b] montrent ainsi que les problèmes et domaines qui ont été utilisés jusqu'ici dans les compétitions peuvent tous être résolus par des plans séquentiels. En effet, les planificateurs qui ont remporté les compétitions IPC dans la catégorie temporelle, même s'ils sont très efficaces dans un cadre temporel restrictif, ne peuvent pas résoudre des problèmes pour lesquels toutes les solutions possibles nécessitent des actions parallèles (problèmes temporellement expressifs). Ils ne peuvent résoudre que les problèmes pour lesquels il existe au moins une solution séquentielle (problèmes temporellement simples). Ils sont donc, pour l'instant, encore loin de pouvoir résoudre des problèmes réels. L'évaluation objective de ces systèmes nécessite maintenant la mise au point de nouveaux benchmarks temporellement expressifs et de leur intégration aux jeux de tests utilisés pendant les compétitions.

Dans cette partie, nous commençons par dresser un état de l'art (chapitre 2) des techniques utilisées par les planificateurs temporels actuels. Dans la section 2.1, nous présentons les principaux langages de représentation de problèmes temporels et leur expressivité. Nous dressons ensuite, dans les sections 2.2 et 2.3, un état de l'art des premiers planificateurs temporels, puis des planificateurs temporels actuels. Nous terminons ce chapitre par une taxinomie des planificateurs temporels (section 2.4).

Nous présentons ensuite, dans le chapitre 3, l'algorithme TLP-GP (Temporally Lifted Progression GraphPlan) que nous avons développé pour résoudre des problèmes temporellement expressifs représentés dans un langage dont l'expressivité temporelle est supérieure à celle de PDDL2.1. TLP-GP est complet pour les sous-langages temporellement expressifs de PDDL2.1 et utilise une méthode similaire à celle utilisée par les planificateurs de la famille de BLACKBOX [Kautz, Selman, 1999] puisqu'il est basé sur l'utilisation d'un graphe de planification simplifié et d'un solveur Sat Modulo Theory (SMT). TLP-GP commence par construire un graphe de planification atemporel et sans mutex, sans prendre en compte ni la durée, ni l'instant de début des actions (section 3.3.1). Il cherche ensuite à extraire un plan solution, et il peut pour cela utiliser deux méthodes différentes que nous comparons entre elles :

- TLP-GP-1 (section 3.3.2) effectue une recherche arrière dans le graphe de planification pour sélectionner les actions du plan solution. Il utilise également un solveur de problèmes temporels disjonctifs (DTP) pour vérifier, à chaque étape du chaînage arrière, la consistance d'un ensemble de contraintes temporelles.
- TLP-GP-2 (section 3.3.3) code la totalité du graphe dans une logique QF-RDL (Quantifier-Free Real Difference Logic) et l'ensemble de formules correspondant est donné en entrée à un solveur SMT qui lui cherche un modèle. Ce modèle, lorsqu'il existe, fournit un plan solution.

Dans la section 3.5, nous décrivons ensuite les extensions à l'expressivité de PDDL2.1 que nous avons introduites dans TLP-GP. Nous montrons que l'expressivité de notre langage de représentation permet de prendre en compte des préconditions qui peuvent être requises et des effets qui peuvent apparaître sur n'importe quel intervalle temporel relatif à l'instant de début ou de fin d'une action. TLP-GP peut aussi prendre en compte, de manière simple, des événements exogènes, des buts temporellement étendus, des fenêtres d'activation… ainsi que plusieurs modalités qui permettent d'étendre significativement l'expressivité de son langage de représentation. Pour mener notre étude expérimentale, nous avons mis au point un ensemble de nouveaux benchmarks temporellement expressifs que nous décrivons dans la section 3.6. Dans la section 3.7 :

- Nous comparons les performances de TLP-GP avec celles des planificateurs d'IPC'2008 sur les benchmarks temporellement simples de la compétition. TLP-GP s'avère, sur ces problèmes, bien moins performant que les autres planificateurs d'IPC'2008.
- Nous montrons que ces même planificateurs sont cependant incapables de résoudre le petit problème temporellement expressif de [Cushing et al., 2007.a, Figure 3] alors que TLP-GP le résout sans difficulté.
- Nous comparons alors TLP-GP avec trois planicateurs temporellement expressifs représentatifs de l'état de l'art actuel (LPGP [Long, Fox, 2003], VHPOP 2.2 [Younes, Simmons, 2003] et CRIKEY3 [Coles, Fox, Long, Smith, 2008]). Cette étude expérimentale montre que TLP-GP est, pour des problèmes temporellement expressifs, plus performant que ces trois planificateurs.

2. Etat de l'art de la planification temporelle

2.1. Représentation de problèmes de planification temporels

Les premiers planificateurs temporels sont basés sur des logiques temporelles qui prennent en compte le temps de manière spécifique. On y dispose d'une "ligne temporelle" explicite ; planifier revient alors à contraindre, par rapport à cette ligne temporelle, des actions de durée non nulle résolvant les buts donnés. Pour cela, il faut pouvoir prédire leurs effets et vérifier leurs conditions d'applicabilité. Une "base de données temporelle" est utilisée pour mémoriser les effets des actions et les contraintes imposées par le problème. Pour décrire temporellement les effets des actions, deux principaux types de logiques temporelles sont employés : la logique des instants (originellement proposée par [McDermott, 1982]) ou la logique des intervalles (originellement proposée par [Allen, 84]). Dans ces logiques, les contraintes entre les qualifications temporelles sont gérées séparément du planificateur (mais pas

indépendamment) grâce à une "carte temporelle" (ou Time Map), en utilisant des algorithmes de propagation spécifiques.

2.1.1. Les logiques temporelles pour la planification

Dans une logique temporelle, la synthèse de plans peut être considérée comme un double processus : la recherche d'actions susceptibles d'aboutir au but et leur ordonnancement dans le temps. Il peut alors exister deux types d'occurrences d'actions primitives :

- L'occurrence d'une action primitive instantanée, qui sera représentée par un point,
- L'occurrence d'une action primitive de durée non nulle, qui sera représentée par un intervalle temporel.

Un planificateur temporel traitera un ensemble d'occurrences d'actions et un ensemble de contraintes temporelles. La solution sera consistante s'il existe au moins un ordonnancement dans le temps des occurrences qui satisfait l'ensemble des contraintes temporelles. Le planificateur doit donc disposer d'un algorithme lui permettant de tester la satisfiabilité du plan courant pour la logique temporelle qu'il utilise.

D'un point de vue logique, on peut distinguer trois grandes méthodes permettant de modéliser le temps et le changement :

- On peut considérer des opérateurs temporels qui sont généralement introduits comme des opérateurs modaux F et P [Prior, 1957]. Par exemple, l'opérateur F est un opérateur de possibilité interprété ainsi : Fp signifie "il existe un moment dans le Futur où p est vrai" et P est l'opérateur associé pour le Passé : "il existe un moment dans le Passé où p a été vrai".
- Une deuxième méthode, utilisée en planification, consiste à réifier les instants ou les intervalles de temps sur lesquels des propositions logiques sont valides. C'est ce que réalisent le calcul des situations [McCarthy, Hayes, 1969], ou la théorie de l'action de [Allen, 1984] et les théories qui l'ont suivie. Le langage utilisé peut par exemple être formé d'expression du type Holds(p,I) pour indiquer que la proposition p est valide sur l'intervalle I. Cette imbrication d'une logique dans une autre implique une forme de quantification sur les propositions à cause de la complexité introduite dans le raisonnement. Elle a été critiquée par [Ghallab, 1991] qui, pour répondre à ce problème, a mis au point des algorithmes spécifiques basés sur des treillis d'instants.
- Une autre approche consiste, pour contourner ces problèmes, à réifier les entités temporelles (qui peuvent être, suivant la logique considérée, des instants ou des intervalles) en les introduisant comme arguments des prédicats dans une théorie du premier ordre. On peut alors traduire : "Le train part de la gare de Pamiers en direction de Toulouse à 8h21" par *départ(train,pamiers,toulouse,t)* où *t* correspond à l'instant où le train part de la gare. On peut aussi réifier les événements que l'on veut caractériser comme le propose [Davidson, 1967] : "Fred a mangé un sandwich à midi dans le train" peut être représenté par la formule *mange(fred,sandwich,e)* $\wedge$ *à(midi,e)* $\wedge$ *dans(train,e)* ce qui permet d'inférer séparément les différentes caractéristiques de l'événement, par exemple que "Fred a mangé un sandwich à midi". Cette dernière approche à l'avantage d'introduire explicitement des individus ayant une extension temporelle et de rester dans un cadre de logique du premier ordre.

Pour comparer les structures temporelles, il faut considérer les propriétés des ordres qui caractérisent diverses théories à base d'instants. Si l'on discrétise le temps, on peut considérer

un intervalle comme un ensemble de points correspondants à des instants successifs. On parle alors d'ordre sous-jacent à une représentation d'intervalles en parlant de l'ordre correspondant à cette structure d'instants. Nous allons donc commençer par présenter les logiques temporelles basées sur les instants avant de consacrer une deuxième section aux logiques d'intervalles.

a) Logiques temporelles d'instants

Les structures d'instants sont des structures du type $I = (T, <)$ où T est un ensemble d'instants et où $<$ est un ordre partiel strict (irréflexif et transitif) qui peut avoir, selon la logique considérée, certaines des propriétés suivantes :

- **Linéarité :** $\forall x \forall y \ ((x < y) \vee (y < x) \vee (y = x))$
- **Succession :** $(\forall x \exists y \ (y < x)) \wedge (\forall x \exists y \ (x < y))$
  (infinité vers le passé et le futur)
- **Densité :** $\forall x \forall y \ ((x < y) \rightarrow (\exists z \ (x < z < y)))$
- **Ordre Discret :**
  $$\forall x \forall y \ (((x < y) \rightarrow (\exists z \ ((x < z) \wedge \neg \exists u \ (x < u < z)))) \wedge ((x < y) \rightarrow (\exists z \ ((z < y) \wedge \neg \exists u \ (z < u < y)))))$$

Les deux derniers axiomes (densité et ordre discret) sont incompatibles. De nombreuses théories supposent la linéarité de l'ordre sous-jacent, sauf les approches qui, comme le calcul des situations [McCarthy, Hayes, 1969], considèrent le futur comme indéterminé et modélisable par une structure munie de branches représentant les alternatives possibles à partir d'un point, ou bien celles qui considèrent le passé comme mal connu avec diverses histoires envisageables. Il faut noter que la théorie des ordres munis de la linéarité, la succession et la densité d'une part, et les théories des ordres munis de la linéarité, la succession et la discrétion d'autre part sont syntaxiquement complètes (tout ce que l'on peut énoncer dans ces théories est soit un théorème, soit la négation d'un théorème).

L'étendue des logiques qu'il est possible de construire en utilisant une représentation temporelle basée sur les intervalles est bien plus vaste que celle que l'on peut obtenir en utilisant des représentations basées sur les instants. Plusieurs catégories de logiques d'intervalles ont ainsi été développées, comme par exemple des structures basées sur l'inclusion ou le recouvrement. L'approche utilisée originellement en planification a été introduite par [Allen, 1984] et se base sur une relation de "contact" temporel (meets).

b) Raisonnement sur des intervalles : la logique d'Allen

Dans ce système, le temps est représenté sous forme d'intervalles temporels, chacun étant formé par un ensemble T ordonné de points tel que $\exists t^- \ (\forall t \in T) \ (t^- < t)$ et $\exists t^+ \ (\forall t \in T) \ (t^+ > t)$. La paire $\{t^-, t^+\}$ représente les points limites ("end-points") de l'intervalle. Toutes les relations intuitives souhaitables pour pouvoir parler d'ordre temporel sont dérivées de l'unique primitive "meets" qui exprime que deux intervalles $T = \{t^-, t^+\}$ et $S = \{s^-, s^+\}$ se touchent sans se recouvrir ($t^+ = s^-$). Cette relation est axiomatisée comme suit dans [Allen, Ferguson, 1994], en notant $x \parallel y \equiv meets(x,y)$ et $x \parallel y \parallel z \equiv meets(x,y) \wedge meets(y,z)$:

- Les intervalles définissent une classe d'équivalence des intervalles qui les rencontrent :

$$((x \parallel y) \wedge (x \parallel z) \wedge (t \parallel y)) \rightarrow (t \parallel z)$$

- Deux intervalles qui rencontrent les mêmes intervalles sont égaux :

$$[\forall x \forall z((x \parallel y \parallel z) \wedge (x \parallel u \parallel z))] \rightarrow (u = y)$$

- Le temps est infini et les intervalles sont finis :

$$\forall x \exists y \exists z\ ((x \parallel y) \wedge (z \parallel x))$$

- Intervalles "sommes" de deux intervalles y et z qui se rencontrent :

$$(x \parallel y \parallel z \parallel t) \rightarrow \exists u\ (x \parallel u \parallel t)$$

- Linéarité de l'ordre sous-jacent : pour deux paires d'intervalles qui se rencontrent, soit le "point" de rencontre est le même, soit il existe un intervalle entre ces deux points de rencontre (et l'un de ces points est avant l'autre) :

$$((x \parallel y) \wedge (u \parallel v)) \rightarrow ((x \parallel v) \oplus (\exists z\ (x \parallel z \parallel v)) \oplus (\exists z\ (u \parallel z \parallel y)))$$

On peut alors définir 13 relations (cf. Tableau 1) qui expriment tous les cas possibles d'ordre entre les extrémités d'un intervalle de temps connexe. Dans cette logique, une formule est une conjonction de contraintes temporelles de la forme I $\Re$ J où I et J représentent des intervalles et $\Re$ représente une relation temporelle qui est une disjonction de relations élémentaires appelées les 13 primitives d'Allen.

	Relation	**Inverse**	**Sémantique de la relation**	**Schéma**
equal	t = s (symétrique)		$(t^- = s^-) \wedge (t^+ = s^+)$	
before	t < s	t > s	$t^+ < s^-$	
overlap	t <u>o</u> s	t <u>oi</u> s	$(t^- < s^-) \wedge (t^+ > s^-) \wedge (t^+ < s^+)$	
meets	t <u>m</u> s	t <u>mi</u> s	$t^+ = s^-$	
during	t <u>d</u> s	t <u>di</u> s	$(t^- > s^-) \wedge (t^+ < s^+)$	
starts	t <u>s</u> s	t <u>si</u> s	$(t^- = s^-) \wedge (t^+ < s^+)$	
finishes	t <u>f</u> s	t <u>fi</u> s	$(t^- > s^-) \wedge (t^+ = s^+)$	

Tableau 1 : les 13 primitives d'Allen

La composition des relations se fait toujours par conjonctions ou disjonctions de primitives temporelles avec les mêmes règles de distributivité et de factorisation qu'en logique classique. L'ensemble des relations muni de la loi de composition est une algèbre de relations sur les intervalles qui, au delà du cadre de la planification, est très largement utilisée en IA (traitement du temps en langage naturel, raisonnements temporels divers).

Malheureusement, la logique d'Allen ne permet pas de représenter des événements instantanés. Des structures hybrides ont donc vu le jour comme les treillis d'instants [Ghallab, 1989] qui permettent de prendre en compte des contraintes ou des relations entre instants pour

le traitement des événements et entre intervalles de temps pour le traitement de faits vérifiés sur une certaine durée.

Comme nous venons rapidement de le voir, les logiques temporelles autorisent une expressivité importante. Cependant, la nécessité de comparer objectivement différents algorithmes de planification temporelle, en particulier pendant les compétitions IPC, a rapidement conduit à l'élaboration d'un langage de représentation des domaines de planification (le langage PDDL) qui soit commun aux systèmes participant à ces compétitions.

Nous allons maintenant présenter rapidement les principaux aspects de l'extension temporelle de ce langage.

2.1.2. PDDL 2.1 : une extension pour la prise en compte de domaines temporels

Comme nous l'avons vu au chapitre II.2, à l'origine, le langage PDDL n'intégrait pas d'aspects temporels. Il a d'abord été modifié pour intégrer la notion de durée au cadre de la planification STRIPS classique. Cependant, la notion d'action durative y demeure encore limitée et reflète une position fondamentale prise par [Fox, Long, 2003] qui pose que les actions sont vraiment instantanées mais qu'elles peuvent lancer et terminer des processus continus. Selon ce postulat, les actions duratives sont décomposées en triplets comprenant une action de début, un processus, et une action d'arrêt. Ce type de modélisation impose que les préconditions soient vraies sur la totalité des intervalles spécifiés, et que les effets puissent avoir lieu à des instants arbitraires durant le déroulement de l'action. Chacune de ces actions (action de début, processus, action d'arrêt) est ainsi découpée en actions plus simples, chacune ayant seulement des effets au début (at start), à la fin (at end), et ayant seulement des préconditions au début (at start), à la fin (at end), ou sur l'ensemble de la durée de l'action (over all).

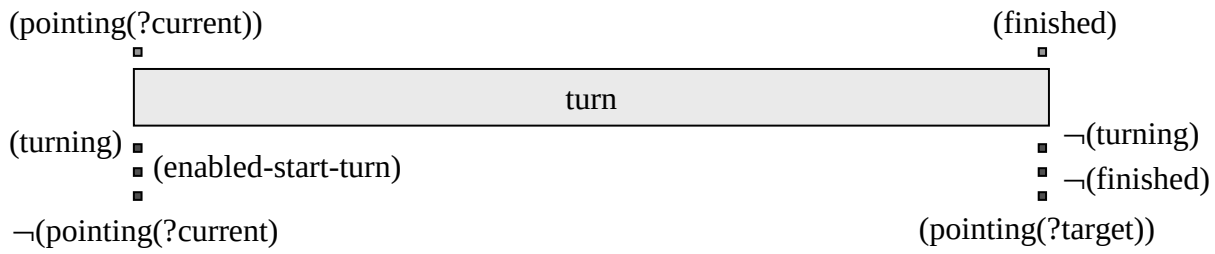
Cette méthode est cependant coûteuse car elle oblige le planificateur à effectuer un travail supplémentaire pour relier entre elles ces actions de plus faible granularité et qu'elle impose à l'agent le contrôle d'actions plus simples mais artificielles.

Pour illustrer la notion d'action durative en PDDL2.1, considérons l'exemple [Smith, 2003] d'un vaisseau spatial qui doit pivoter afin d'orienter un instrument vers une cible donnée. Pour faire tourner ce vaisseau, des réacteurs sont mis à feu afin d'augmenter la vitesse angulaire. Le vaisseau spatial pivote alors jusqu'à ce qu'il soit dirigé dans la bonne direction. A ce moment là, les réacteurs sont à nouveau mis à feu afin d'arrêter la rotation. Mettre le feu aux réacteurs consomme du carburant, et exige que le contrôleur soit entièrement consacré à cette tâche. En outre, quand les réacteurs sont mis à feu, le vaisseau vibre, et certaines autres actions ne peuvent pas être exécutées. Tandis que les mises à feu des réacteurs sont relativement rapides, la phase de rotation ne l'est pas. Généralement la rotation d'un vaisseau spatial est un processus lent qui peut prendre plusieurs minutes. En effet, une rotation rapide exige une accélération et une décélération plus importantes qui consomment donc plus de carburant. On peut modéliser cette tâche comme une action durative en PDDL2.1 par le code suivant :

```
(:durative-action turn
 :parameters (?current-target ?new-target - target)
 :duration (= ?duration (/ (angle ?current-target ?new-target) (turn-rate)))
 :condition (and (at start (pointing ?current-target))
                 (at start (>= (propellant) propellant-required))
                 (at start (not (controller-in-use))))
```

```
 :effect (and (at start (not (pointing ?current-target)))
              (at start (decrease (propellant) propellant-required))
              (at start (controller-in-use))
              (at start (vibration))
              (at end (not (controller-in-use)))
              (at end (not (vibration)))
              (at end (pointing ?new-target))))
```

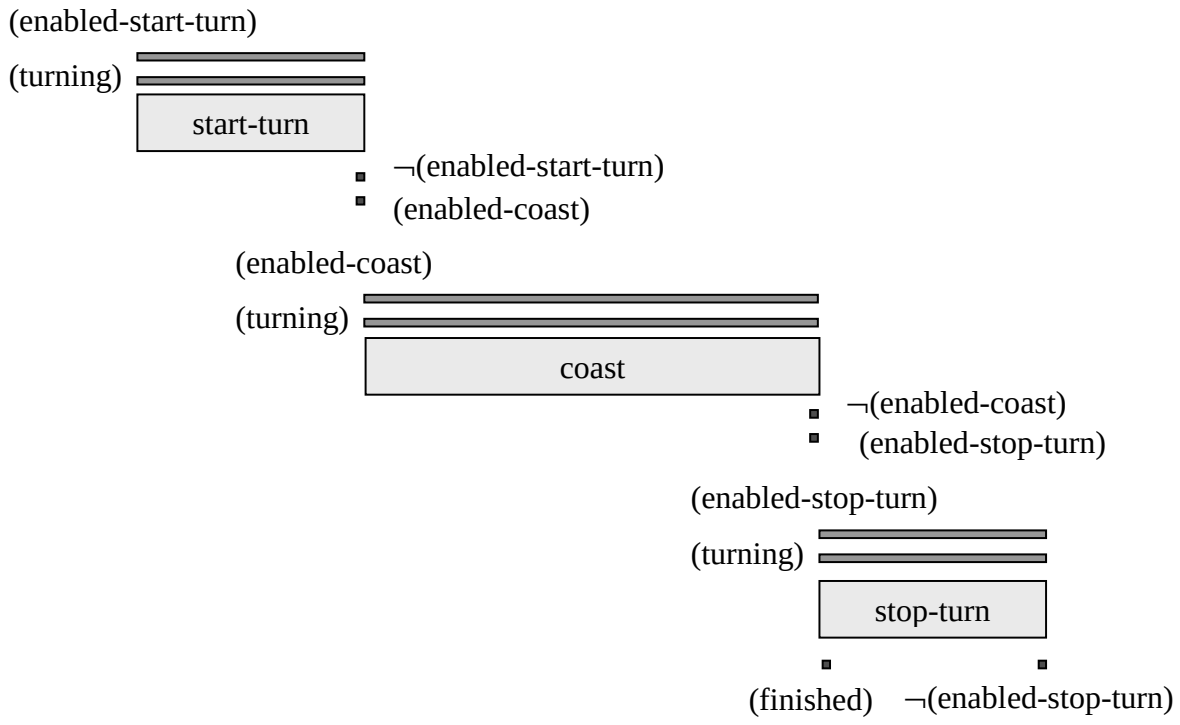
Cette action peut être représentée par le schéma suivant :



En PDDL2.1, cette représentation n'est pas suffisante pour modéliser l'action telle que nous l'avions précédemment décrite. Si l'on reste dans le cadre du langage PDDL2.1, la seule manière de procéder pour modéliser cette action consiste alors, comme le suggèrent [Fox, Long, 2003] et comme le réalise [Smith, 2003], à la compléter par plusieurs actions plus simples.

```
(:durative-action start-turn
 :parameters ()
 :duration (= ?duration (start-turn-duration))
 :condition (and (at start (not (controller-in-use)))
                 (at start (>= (propellant) (/ propellant-required 2)))
                 (over all (turning))
                 (over all (enabled-start-turn)))
 :effect (and (at start (decrease (propellant) (/ propellant-required 2)))
              (at start (controller-in-use))
              (at start (vibration))
              (at end (not (controller-in-use)))
              (at end (not (vibration)))
              (at end (not (enabled-start-turn)))
              (at end (enabled-coast))))

(:durative-action coast
 :parameters ()
 :duration (= ?duration (coast-duration))
 :condition (and (over all (turning))
                 (over all (enabled-coast)))
 :effect (and (at end (not (enabled-coast)))
              (at end (enabled-stop-turn))))

(:durative-action stop-turn
 :parameters ()
 :duration (= ?duration (RCS-duration))
 :condition (and (at start (not (controller-in-use)))
                 (at start (>= (propellant) (/ propellant-required 2)))
                 (over all (turning))
                 (over all (enabled-stop-turn)))
 :effect (and (at start (decrease (propellant) (/ propellant-required 2)))
              (at start (controller-in-use))
              (at start (vibration))
              (at end (not (controller-in-use)))
              (at end (not (vibration)))
              (at end (not (enabled-stop-turn)))
              (at start (finished))))
```

Ces actions peuvent être représentées par le schéma suivant :



Si le but est de s'orienter vers une cible particulière, une action *turn* sera nécessaire. L'action *turn* a une precondition *at end* (*finished*), qui ne peut être satisfaite qu'en ajoutant une action *stop-turn*. Stop-turn a une précondition *over all* (*enabled-stop-turn*) qui ne peut être satisfaite que par l'effet *at end* de l'action *coast*. L'action *coast* a également une précondition *over all* (*enabled-coast*) qui ne peut être satisfaite que par un effet *at end* de l'action *start-turn*. L'action *start-turn* a une precondition *over all* (*enabled-start-turn*) qui ne peut être satisfaite que par un effet *at start* de l'action *turn*. Comme conséquence, l'action *turn* contraint les trois sous-actions dans le plan, et chaque sous-action contraint la sous-action précédente dans le plan. De plus, chacune des sous-actions a une précondition *over all* (*turning*) qui ne peut être satisfaite que pendant la durée de l'action *turn*. Ainsi, la seule possibilité d'obtenir une solution est que la séquence de ces trois sous-actions soit imbriquée pendant l'exécution de l'action *turn*.

[Smith, 2003] constate que cette manière de procéder complexifie inutilement la représentation des problèmes et propose la prise en compte, dans le langage PDDL, d'intervalles temporels. L'action *turn* pourrait ainsi être simplement modélisée :

```
(:durative-action turn
 :parameters (?current-target ?new-target - target)
 :duration (= ?duration (/ (angle ?current-target ?new-target) (turn-rate)))
 :condition (and (at start (pointing ?current-target))
                 (at start (>= (propellant) propellant-required))
                 (at start (not (controller-in-use)))
                 (at (- end RCS-duration) (>= (propellant) (/ propellant-required 2)))
                 (at (- end RCS-duration) (not (controller-in-use))))
```

```
:effect (and (at start (not (pointing ?current-target)))
             (at start (decrease (propellant) (/ propellant-required 2)))
             (over [start (+ start RCS-duration)] (controller-in-use))
             (over [start (+ start RCS-duration)] (vibration))
             (at (- end RCS-duration) (decrease (propellant) (/ propellant-required 2)))
             (over [(- end RCS-duration) end] (controller-in-use))
             (over [(- end RCS-duration) end] (vibration))
             (at end (pointing ?new-target))))
```

A ce jour, aucun des domaines proposés aux compétitions IPC n'a encore pris en compte l'extension proposée par [Smith, 2003].

2.1.3. Expressivité temporelle

Alors que l'on pensait avoir fait de grands progrès sur le traitement de problèmes de planification temporels (cf. les résultats des compétitions IPC'2002, IPC'2004, IPC'2006), et qu'un grand nombre de planificateurs temporels étaient capables de prendre en compte les problèmes décrits en PDDL2.1, les travaux récents de [Cushing et al., 2007.b] ont mis en évidence les limites des approches utilisées. Ils ont en effet montré qu'aucun des domaines et problèmes qui avaient été utilisés jusqu'ici dans les compétitions IPC ne nécessitait d'actions concurrentes pour sa résolution et qu'ils pouvaient tous être résolus par des plans séquentiels. Or un grand nombre de problèmes réels, pour être résolus, nécessitent des actions parallèles (gérer une aérogare, une gare, cuire des céramiques, faire la cuisine, planter un clou...). Les planificateurs primés dans les compétitions IPC, même s'ils sont performants dans un cadre temporel limité, sont donc loin de pouvoir être utilisés pour la résolution de tels problèmes. L'évaluation objective de ces systèmes nécessite maintenant la mise au point de nouveaux benchmarks représentant des problèmes temporellement expressifs. Ce travail indispensable n'a malheureusement toujours pas été réalisé pour la compétition IPC'2008. Dans cette section, nous résumons les travaux de [Cushing et al., 2007.b] et nous étendons leurs notations pour prendre en compte des langages plus expressifs.

[Cushing et al., 2007.a] commencent par définir la notion de simultanéité requise comme la nécessité, pour résoudre un problème ou un domaine particulier, d'utiliser des actions concurrentes. Ils posent ensuite les définitions suivantes :

Définition 38 (plan concurrent / séquentiel)

Un plan est concurrent si deux actions au moins sont concurrentes, sinon le plan est séquentiel.

Définition 39 (problème temporellement expressif)

Un problème temporellement expressif est un problème qui nécessite des actions concurrentes pour sa résolution.

Définition 40 (problème temporellement simple)

Un problème temporellement simple est un problème qui peut être résolu grâce à un plan séquentiel.

Définition 41 (domaine temporellement expressif)

Un domaine temporellement expressif est un domaine qui nécessite des actions concurrentes pour la résolution de tous ses problèmes.

Définition 42 (domaine temporellement simple)

Un domaine temporellement simple (ou intrinsèquement séquentiel) est un domaine pour lequel tous les problèmes peuvent être résolus grâce à un plan séquentiel.

[Rintanen, 2007.a] montre que la résolution de problèmes temporellement expressifs est EXPSPACE-complète alors que la complexité de la recherche d'une solution d'un problème temporellement simple peut se ramener à celle d'un problème de planification dans le cadre classique et est donc PSPACE-complète d'après [Bylander, 1994].

Définition 43 (domaine temporellement simple mixte)

Un domaine temporellement simple mixte est un domaine dans lequel tous les problèmes peuvent être résolus par un plan séquentiel mais où il peut exister, pour certains problèmes, des plans-solutions concurrents qui n'ont pas d'équivalent séquentiel.

[Cushing et al., 2007.b] proposent également une méthode permettant de prouver qu'un domaine est temporellement expressif. Toutes ces notions leur permettent naturellement de diviser l'espace des langages temporels en deux parties :

- Les langages temporellement expressifs, c'est-à-dire ceux qui permettent de représenter des problèmes pour la résolution desquels la simultanéité de certaines actions est requise ;
- Les langages temporellement simples qui ne permettent pas de représenter de tels problèmes.

Notation de [Cushing, Kambhampati, Mausam, Weld, 2007]

Cushing et al. Utilisent la notation $L_{\text{effets}}^{\text{préconditions}}$ pour représenter l'expressivité d'un langage de planification temporel, où <préconditions> et <effets> représentent les instants auxquels les préconditions doivent être vraies et ceux pendant lesquels les effets peuvent se produire. Les valeurs prises par <préconditions> et <effets> peuvent être les suivantes :

- s : instant de démarrage d'une action (*at start*) ;
- e : instant de fin d'une action (*at end*) ;
- o : sur toute la durée de l'action (*over all*).

Avec cette notation, $L_{s,e}^{o}$ désigne ainsi un langage pour lequel les préconditions d'une action doivent être vraies durant toute l'exécution, et les effets doivent survenir à l'instant de début ou à l'instant de fin d'une instance de l'action.

Extension de cette notation

Nous complétons cette notation par la prise en compte d'intervalles temporels. [s, e] représentera ainsi le fait que les préconditions (respectivement les effets) peuvent être

requises (respectivement produits) sur n'importe quel intervalle (ouvert, fermé ou mixte) compris entre l'instant de démarrage et l'instant de fin d'une action. Cette notation peut encore être étendue en notant par [s+, e+] le fait que les préconditions ou effets peuvent être requises (ou produits) sur n'importe quel intervalle (ouvert, fermé ou mixte) relatif à une action. Ces intervalles peuvent donc maintenant précéder l'instant de démarrage ou suivre l'instant de fin d'une action.

Suivant cette notation, les langages PDDL2.1 et PDDL3 sont des langages ayant pour expressivité $L_{s,e}^{s,o,e}$. Ces langages sont temporellement expressifs.

Dans le cadre des sous-langages temporels de PDDL 2.1/3, [Cushing et al., 2007.a] démontrent également un théorème qui permet de caractériser leur expressivité, et par conséquent celle des domaines et des problèmes de planification qu'ils permettent de représenter :

Définition 44 (condition-avant / condition-après)

- Une *condition-avant* est une précondition qui est nécessaire au moment, ou avant, qu'au moins l'un des effets de l'action soit produit.
- Une *condition-après* est une précondition qui est nécessaire au moment, ou après, qu'au moins l'un des effets de l'action soit produit.

Définition 45 (trou temporel / instant critique)

- Un trou entre deux intervalles temporels disjoints et qui ne se rencontrent pas (au sens de la primitive "meets" d'Allen) est l'intervalle qui se trouve entre eux (tel que l'union de ces trois intervalles est un intervalle unique).
- Une action a un *trou temporel* s'il existe un trou entre une *condition-avant* et un effet, ou un trou entre une *condition-après* et un effet, ou un trou entre deux effets.
- Les actions qui n'ont pas de trou temporel possèdent un *instant critique* : l'unique instant où tous les effets sont produits.

Théorème [Cushing, Kambhampati, Mausam, Weld, 2007] :
Un sous-langage de PDDL 2.1/3 est temporellement simple si et seulement si il interdit les trous temporels (la preuve de ce théorème est fournie en Annexe 4).

Grâce à ce théorème, ils peuvent dresser une taxinomie de l'expressivité des différents langages temporels, taxinomie que nous donnons à la Figure 13, ci-après.

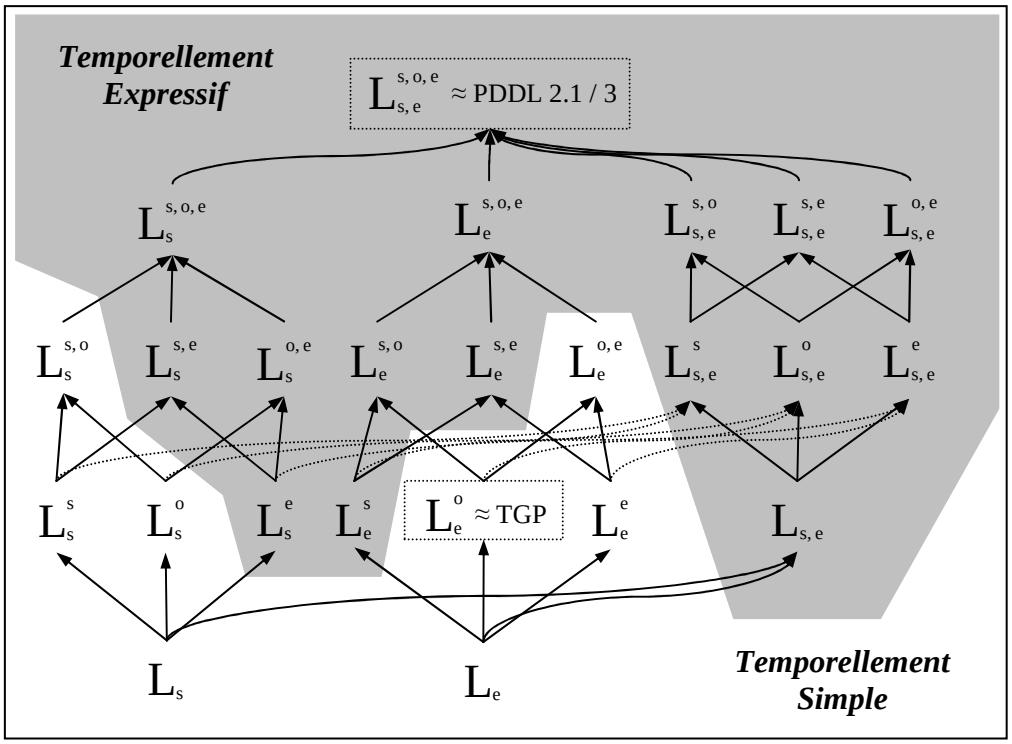


Figure 13 : Taxinomie des langages temporels et leur expressivité d'après [Cushing et al., 2007.a]

En nous basant sur les définitions précédentes, et pour faciliter la compréhension de la suite du document, nous définissons deux classes de planificateurs temporels :

Définition 46 (planificateur temporellement simple)

Un planificateur est dit temporellement simple lorsque son langage de représentation ne permet d'exprimer que des problèmes temporellement simples ou qu'il n'est pas complet pour son langage de représentation lorsque ce dernier est temporellement expressif.

Définition 47 (planificateur temporellement expressif)

Un planificateur est dit temporellement expressif lorsque son langage de représentation est temporellement expressif et qu'il est complet pour ce langage.

Après avoir présenté les principaux langages de représentation de problèmes temporels et leur expressivité, nous allons maintenant voir comment les algorithmes de planification utilisés dans le cadre classique ont été adaptés au cadre temporel. Nous commencerons par évoquer les premiers systèmes qui étaient généralement basés sur des algorithmes de recherche dans les espaces de plans partiels et qui utilisaient les logiques d'instants ou d'intervalles.

2.2. Les premiers planificateurs temporels

Dans le cadre classique de la planification, la seule prise en compte du temps est réalisée par les seules relations de précédence entre actions de durée nulle (càd par la définition d'un ordre partiel ou total basé sur des relations de causalité entre des actions ponctuelles). On ne peut donc pas exprimer explicitement des contraintes temporelles telles que (par exemple) "l'action A1 doit s'exécuter pendant le déroulement de l'action A2" ou "l'exécution de l'action A1 doit avoir lieu entre 3h40 et 4h20". Pour tenir compte du temps d'une façon qui soit plus explicite et plus réaliste, la première solution employée a consisté à rajouter des attributs temporels aux représentations classiques basées sur les opérateurs de changement d'état.

Le système DEVISER [Vere, 1981] a ainsi été le premier à rajouter des fenêtres temporelles, des dates d'activation et des durées à des opérateurs de type STRIPS. Il a été développé pour gérer la sonde spatiale VOYAGER et utilise un formalisme dérivé des réseaux PERT [Kaufman, 1969] avec une représentation du temps basée sur les instants. Une fenêtre d'activation définit les frontières temporelles minimales (EST) et maximales (LST) pour la réalisation d'une action (activité). Elle est la transcription des méthodes des réseaux PERT : ces frontières sont des dates soit de début, soit de fin, au plus tôt ou au plus tard d'une activité. Une fenêtre peut aussi contenir une date idéale optionnelle (IDEAL) indiquant un instant de réalisation préféré (cf. Figure 14). Alors que jusque là, dans les planificateurs traditionnels, les actions ne possédaient pas de durée, dans DEVISER, les préconditions d'une activité doivent être vraies au début et doivent le rester pendant toute la durée d'exécution. Les effets deviennent vrais dès que l'activité se termine. Ce formalisme utilisé dans DEVISER, dont l'expressivité temporelle est L_e^o, sera largement repris dans de nombreux planificateurs et reste très utilisé aujourd'hui. Malheureusement, le choix d'un langage de représentation L_e^o engendre des planificateurs temporellement simples (cf. sous-section 2.1.3).

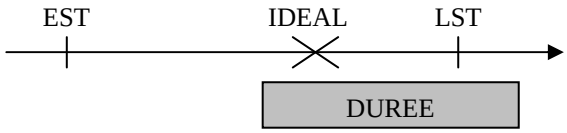


Figure 14 : représentation d'une action (activité) dans le système DEVISER

Après le développement de DEVISER et durant les 15 années suivantes, les planificateurs temporels se sont partagés en deux catégories :

- Les planificateurs qui utilisent une logique temporelle basée sur les instants : FORBIN [Dean, Firby, Miller, 1988], IxTeT [Ghallab, Alaoui, 1989.a/b], [Alaoui, 1990], TRIPTIC [Rutten, Hertzberg, 1993], TEST [Reichgelt, Shadbolt, 1990]…
- Les planificateurs qui utilisent une logique temporelle basée sur les intervalles : TIMELOGIC [Allen, Koomen, 83], TLP [Tsang, 87]…

Tous ces planificateurs étaient, comme ceux développés dans le cadre classique à cette époque, basés sur des algorithmes de recherche dans des espaces de plans. Certains d'entre eux ont introduit des technique novatrices qui seront ensuite largement réutilisées :

- FORBIN (First Order RoBot INtender) [Dean, Firby, Miller, 1988], planificateur basé sur une **logique d'instants**, utilise un **module indépendant**, appelé TMM ("Time-Map

Manager"), chargé de la gestion des contraintes temporelles et développé par [McDermott, Dean, 1987]. Tous les premiers planificateurs basés sur une logique des instants reprendront cette idée d'un module de gestion de base de donnée temporelle indépendant.

- [Allen, 1983] développe TIMELOGIC, le premier planificateur utilisant une modélisation du temps basée sur la **logique des intervalles** (cf. section 2.1.1). Les relations temporelles y sont également gérées dans un module séparé, pour lequel se pose le problème du maintien d'une base temporelle cohérente.
- IxTeT (Indexed Time Table) [Ghallab, Alaoui, 1989], [Ghallab, Laruelle, 1994], [Laborie, Ghallab, 1995], utilise des procédures spécifiques pour gérer quasi-linéairement l'ajout de contraintes temporelles dans un **treillis d'instants**. Comme les planificateurs précédents, IxTeT sépare les mécanismes de raisonnement logique des mécanismes de raisonnement temporel. IxTeT est un système hybride dans lequel un phénomène temporel peut être représenté, si l'on fait abstraction de la sémantique du problème abordé, par un ensemble d'entités élémentaires qui sont soit des instants, soit des intervalles et qui sont reliées par des contraintes symboliques ou numériques.
- ZENO [Penberthy, Weld, 1994] peut prendre en compte des **événements extérieurs**, des **buts temporellement étendus**… Mais les actions ne peuvent y être concurrentes que lorsque leurs effets n'interfèrent pas.

A partir de 1992, l'apparition de nouvelles techniques de planification efficaces dans le cadre classique a conduit à leur chercher des extensions dans le cadre temporel. Dans la section suivante, nous allons voir comment les planificateurs temporels actuels s'appuient sur ces nouvelles techniques de recherche pour prendre en compte le temps.

2.3. Algorithmique des planificateurs temporels actuels

En 1992 sont apparues les techniques de planification SAT [Kautz, Selman, 1992] (cf. partie III), puis GRAPHPLAN [Blum, Furst, 1995] (cf. chapitre II.5) et la recherche heuristique dans les espaces d'états initiée par ASP [Bonnet, Loerincs, Geffner, 1997] et HSP [Bonnet, Geffner, 1998]. Cette dernière technique demeure aujourd'hui la plus performante dans le cadre classique (cf. chapitre II.3). Les planificateurs temporels actuels sont essentiellement basés sur trois types d'algorithmes : la recherche dans les espaces d'états, la recherche dans les espaces de plans et GRAPHPLAN. La majorité de ces systèmes utilise le langage de planification PDDL [McDermott, 1998] qui permet maintenant, lors des compétitions bi-annuelles IPC, de les comparer de manière plus objective. Depuis la compétition IPC'2002, la version 2.1 du langage PDDL [Fox, Long, 2003] (cf. section 2.1.2) autorise la prise en compte de la dimension temporelle.Nous allons maintenant décrire les principes algorithmiques essentiels de ces trois familles de planificateurs temporels.

2.3.1. Recherche dans les espaces d'états étendus

Pour prendre en compte certains aspects temporels, les planificateurs basés sur un algorithme de recherche dans les espaces d'états étendus utilisent tous des nœuds de recherche représentés par un état du monde (comme dans le cas des algorithmes atemporels) et un ensemble d'actions en cours d'exécution associées à leurs instants de démarrage. A partir de cette représentation, deux méthodes peuvent être envisagées :

- Diriger la recherche en se basant d'abord sur les instants auxquels un événement peut se produire (requête d'une précondition, production d'un effet d'une action, événement extérieur, but temporellement étendu…). Chaque décision du type "*quand* exécuter une

action" est alors prise avant toute décision du type "*quelle* action exécuter". Cette approche est appelée **planification avec époques de décisions** (Decision Epoch Planning).

- Diriger la recherche en se basant d'abord sur les actions à utiliser avant de décider de leur ordonnancement dans le temps. Cette approche consiste à ne prendre les décisions du type "*quand* exécuter une action" qu'après que toutes les décisions du type "*quelle* action exécuter" aient été prises. Elle est appelée **planification à progression temporelle flottante** (Temporally Lifted Progression Planning).

a) Planification avec époques de décision (DEP)

La planification DEP utilise comme attribut central d'un état temporel N, l'état du monde *state*(N). Cet état est responsable du succès de la planification DEP car il permet l'utilisation d'heuristiques très efficaces développées dans le cadre de la planification classique atemporelle.

Définition 48 (état temporel)

Un état temporel N est défini par :

- l'état du monde *state*(N),
- un instant t(N),
- et *agenda*(N), une structure qui enregistre les actions qui ne sont pas encore terminées et leurs instants de démarrage.

Pour décrire le fonctionnement des algorithmes de planification dans les espaces d'états étendus nous utilisons la simulation d'une étape de plan définie par [Fox, Long, 2003]. Une étape d'un plan P = $<s_1, \ldots, s_n>$ est convertie en une séquence de transitions, c'est-à-dire un plan séquentiel classique. La simulation de l'exécution de P est alors réalisée en appliquant les transitions, en séquence, en partant de l'état initial. La simulation échoue lorsque la séquence de transitions n'est pas exécutable. La simulation échoue également si l'une des conditions "over all" est violée. Dans ces deux cas, P n'est pas exécutable. P est une solution lorsque le but est vérifié dans le modèle résultant de la simulation.

Dans un algorithme de planification DEP, il existe deux manières pour générer un nœud fils :

- **Développement (fattening) :** Pour un état temporel donné N, on génère un nœud fils N_A pour chaque action A qui diffère de N uniquement par l'ajout d'une nouvelle étape contenant uniquement l'action A. L'instant correspondant à l'état N_A reste le même que pour N.
- **Avance dans le temps :** On génère un seul nœud fils en simulant une avance dans le temps juste après la prochaine transition dans l'agenda à d + ε, où d correspond à la durée restant jusqu'au prochain début ou fin d'une étape dans l'agenda.

Malgré ses avantages, la planification DEP ne permet pas de résoudre l'ensemble des problèmes exprimés en PDDL2.1 car elle n'est pas complète pour les langages temporellement expressifs. Par ailleurs, elle est complète pour les sous langages temporellement simples de PDDL2.1, mais ne permet pas d'obtenir des plans solutions optimaux en temps d'exécution. Une variante appelée DEP+, formalisée dans [Cushing et al.,

2007.a], permet d'obtenir cette optimalité en ajoutant des époques de décisions à des instants où il ne se produit pas d'effets d'actions.

Dans un algorithme de planification DEP+, il existe deux manières pour générer un nœud fils :

- **Développement (fattening) :** Pour un état temporel donné N, on génère deux nœuds fils N^e_A et N^s_A pour chaque action A qui diffèrent de N uniquement par l'ajout d'une nouvelle étape contenant uniquement l'action A. L'instant correspondant à l'état N^e_A est, par rapport à celui de N, avancé dans le temps du maximum de la durée des actions du problème. L'instant correspondant à l'état N^s_A est, par rapport à celui de N, avancé dans le temps du maximum de la durée des actions du problème et retardé de la durée de A.
- **Avance dans le temps :** On génère un seul nœud fils en simulant une avance dans le temps de d + ε, où d correspond à une durée telle que l'instant de N augmenté de d et du maximum de la durée des actions du problème correspond au prochain début ou fin d'une étape dans l'agenda.

La planification DEP+ est complète et optimale en temps d'exécution pour les sous-langages temporellement simples de PDDL2.1, mais elle n'est pas complète pour ses sous-langages temporellement expressifs. La planification à progression temporelle flottante, que nous allons maintenant décrire, a été récemment développée pour pallier ce problème.

b) Planification à progression temporelle flottante

Dans la planification DEP, chaque décision du type "*quand* exécuter une action" est prise avant toute décision du type "*quelle* action exécuter". DEP essaie de créer des plans en démarrant des instances d'actions uniquement aux instants où des effets se produisent.

La planification DEP impose par conséquent les deux hypothèses suivantes (DEP+ impose uniquement la seconde) :

- Chaque action démarre immédiatement après le début ou la fin d'autres actions ;
- Les seuls conflits qui empêchent l'exécution d'une action plus tôt, impliquent une action commençant plus tôt.

La planification à progression temporelle flottante propose une approche différente qui consiste à retarder les décisions du type "*quand* exécuter une action" après que toutes les décisions du type "*quelle* action exécuter" aient été prises.

Définition 49 (état temporel flottant)

Un état temporel flottant, N, est défini par :

- une variable temporelle courante τ(N),
- un modèle *state*(N),
- un plan flottant agenda(N),
- et un ensemble de contraintes temporelles *constraints*(N).

L'agenda utilisé dans un état temporel flottant est différent de celui utilisé dans un état temporel classique. La répartition exacte dans le temps t( ) est remplacée par des variables temporelles τ_{begin}( ) et impose des contraintes au travers de *constraints*(N). En fait, à chaque étape s, sont associées deux variables $\tau_{begin}(s)$ et $\tau_{end}(s)$. Toutes les contraintes de durée $\tau_{end}(s) - \tau_{begin}(s) = \delta(action(s))$ et d'exclusion mutuelle $\tau_x(s_1) \neq \tau_y(s_2)$ pour les transitions mutuellement exclusives x(*action*(s_1)) et y(*action*(s_2)) (où x, y $\in$ {begin, end}) font implicitement partie de *constraints*( ).

Dans un algorithme à progression temporelle flottante tel que TEMPO [Cushing et al., 2007.a], il existe deux manières pour générer un nœud fils :

- **Développement (fattening) :** Pour un état temporel flottant donné N, on génère un nœud fils N_A pour chaque action A qui correspond au démarrage d'une instance de A et qui diffère de N uniquement par l'ajout à *agenda*(N) d'une nouvelle étape s contenant uniquement l'action A. On ajoute également $\tau_{begin}(s) \geq \tau(N)$ à *constraints*(N). Contrairement à DEP, on simule immédiatement l'exécution du plan à partir de l'état flottant N jusqu'à $\tau_{begin}(s)$.
- **Avance dans le temps :** Pour chaque étape contenant une action A dans *agenda*(N), on génère un nœud fils qui correspond à l'arrêt d'une instance de A. On ajoute $\tau_{end}(s) \geq \tau(N)$ à *constraints*(N') et on démarre la simulation à partir de l'état flottant N jusqu'à $\tau_{end}(s)$.

Parmi les planificateurs à progression temporelle flottante, l'algorithme TEMPO est complet et optimal en temps d'exécution pour tous les sous-langages temporellement expressifs (ou simples) de PDDL2.1.

2.3.2. Recherche dans les espaces de plans

Comme nous l'avons vu dans la section 2.2, les toutes premières techniques de planification temporelles étaient basées sur une recherche dans les espaces de plans partiels de type HTN (Hierarchical Tasks Network). Par la suite, les algorithmes de recherche dans les espaces de plans partiels non hiérarchiques (POP) ont également été étendus avec succès au cadre temporel.

Pour réaliser cette extension, on utilise généralement des intervalles temporels pour la représentation des actions et des propositions, et on remplace la relation de causalité entre actions par un ordre temporel dans les plans partiels. La gestion des conflits est ensuite assurée par un système de satisfaction de contraintes temporelles simples (STN : Simple Temporal Network) [Dechter, Mieri, Pearl, 1991] ou disjonctives (DTP : Disjunctive Temporal Problem) [Stergiou, Koubarakis, 2000]. Comme nous le verrons dans la section 2.3.3, les STN ont également été utilisés pour étendre certains planificateurs de type GRAPHPLAN au cadre temporel.

a) Systèmes de satisfaction de contraintes temporelles simples et disjonctives

Les définitions nécessaires pour décrire un système de satisfaction de contraintes temporelles (TCSP) dérivent de celles développées dans le cadre des CSP classiques [Montanari, 1974]. Un TCSP contient un ensemble de variables $x_1, \ldots, x_n$ qui ont des domaines continus. Chaque variable représente un instant. Chaque contrainte est représentée par un ensemble d'intervalles : $\{I_1, \ldots, I_m\} = \{[a_1, b_1], \ldots, [a_m, b_m]\}$. Nous ne donnons ici les définitions que pour des intervalles fermés mais les mêmes traitements s'appliquent sur des intervalles ouverts

ou mixtes. Une contrainte unaire T_i restreint le domaine de la variable x_i à l'ensemble d'intervalles donné et représente la disjonction $(a_1 \leq x_i \leq b_1) \vee \ldots \vee (a_m \leq x_i \leq b_m)$. Une contrainte binaire T_{ij} restreint les valeurs autorisées pour la distance $x_j - x_i$ et représente la disjonction $(a_1 \leq x_j - x_i \leq b_1) \vee \ldots \vee (a_m \leq x_j - x_i \leq b_m)$. Les contraintes sont données sous forme canonique lorsque les intervalles sont disjoints deux à deux. Un réseau de contraintes binaires (TCSP binaire) contient un ensemble de variables $x_1, \ldots, x_n$ et un ensemble de contraintes unaires et binaires.

Définition 50 (STP/STN)

Un TCSP pour lequel toutes les contraintes ne comportent qu'un seul intervalle est appelé problème temporel simple (STP) ou réseau temporel simple (STN). Dans un tel problème, chaque contrainte $(a_{ij} \leq x_j - x_i \leq b_{ij})$ peut être exprimée comme une paire d'inégalités $x_j - x_i \leq b_{ij}$ et $x_i - x_j \leq - a_{ij}$.

Dans sa structure, un STP est similaire aux "Time Maps" comme celle de [Dean, McDermott, 1987] qui ont été utilisées par les premiers planificateurs temporels. [Dechter, Mieri, Pearl, 1991] montrent que la résolution d'un tel problème est polynomiale en $o(n^3)$.

[Stergiou, Koubarakis, 2000] étendent le cadre des STP en considérant des contraintes temporelles disjonctives, pour obtenir des problèmes temporels disjonctifs (DTP), ce qui offre un cadre plus général que celui des TCSP. Ce nouveau cadre permet d'exprimer des contraintes temporelles comme "A apparaît avant ou après B".

Définition 51 (DTP)

Un problème temporel disjonctif (DTP) est un problème de satisfaction de contraintes dans lequel on considère des contraintes temporelles de la forme $(x_1 - y_1 \leq r_1) \vee \ldots \vee (x_n - y_n \leq r_n)$ où $x_1, \ldots, x_n, y_1, \ldots, y_n$ sont des variables réelles, $r_1, \ldots, r_n$ des constantes et $n \geq 1$.

Contrairement aux TCSP, dans une même contrainte disjonctive, il n'y a plus de restriction aux mêmes paires de variables. Une même variable peut également apparaître dans plusieurs disjonctions. Pour résoudre un DTP, une méthode simple consiste à tester tous les STP sous-jacents. Si la complexité de la résolution de chaque STP est polynomiale, dans le pire des cas, résoudre ou vérifier la consistance d'un DTP est NP-difficile [Dechter, Mieri, Pearl, 1991]. Même la résolution d'un DTP avec au plus deux littéraux par clause est encore NP-complète contrairement à 2-SAT. En pratique, des méthodes de résolution efficaces ont été développées [Tsamardinos, Pollack, 2003], [Liu, Jiang, 2008], et de bons résultats sont connus pour certaines classes de DTP [Kumar, 2005]. La plus grande partie des planificateurs temporellement expressifs utilisent les STP pour contourner la complexité de la résolution des DTP mais le traitement des disjonctions est alors réalisé par le développement de branches supplémentaires dans l'arbre de recherche.

b) Planificateurs utilisant les STN

Le planificateur UCPOP [Penberthy, Weld, 1992], planificateur fonctionnant dans le cadre classique a été étendu au cadre temporel pour donner le planificateur VHPOP (Versatile Heuristic Partial-Order Planner) [Younes, Simmons, 2003]. Il utilise un STN [Dechter, Mieri, Pearl, 1991] pour gérer les contraintes temporelles associées au plan partiel courant. Pour planifier dans les espaces de plans en prenant en compte des actions duratives, [Younes, Simmons, 2003] substituent à l'ordre partiel O de la représentation de plan des algorithmes

POP, un STN (Simple Temporal Network) T. Chaque action a_i d'un plan, sauf les actions factices Init et Goal, est représentée dans le STN par deux nœuds $t_s(a_i)$ (l'instant de début) et $t_e(a_i)$ (l'instant de fin). Dans la suite, $t_x(a_i)$ ou $t_y(a_i)$ seront utilisés pour représenter indifféremment $t_s(a_i)$ et $t_e(a_i)$. T peut être représenté d'une manière compacte par un d-graphe [Dechter, Mieri, Pearl, 1991] qui est un graphe orienté complet dans lequel chaque arc $t_x(a_i) \rightarrow t_y(a_j)$ est étiqueté par la distance temporelle la plus courte, $d(t_x(a_i), t_y(a_j))$, entre les deux nœuds temporels $t_x(a_i)$ et $t_y(a_j)$ (c'est-à-dire $t_y(a_j) - t_x(a_i) \leq d(t_x(a_i), t_y(a_j))$). Un instant supplémentaire, t_0, est utilisé comme point de référence pour représenter l'instant initial. Par défaut, $d(t_x(a_i), t_y(a_j)) = \infty$ pour tout $i \neq j$ ($d(t_x(a_i), t_x(a_i)) = 0$), ce qui signifie qu'il n'y a pas de borne supérieure pour la différence $t_y(a_j) - t_x(a_i)$.

Des contraintes sont ajoutées à T lors de l'ajout de chaque nouvelle action, à la liaison d'une condition ouverte et à l'ajout d'une contrainte d'ordre entre les instants de fin de deux actions. La durée δ_i d'une action a_i est spécifiée comme une conjonction de contraintes de durée simples δ_i R c, où c est une constante réelle évaluée et $R \in \{ =, \leq, \geq \}$. Chaque contrainte de durée simple implique des contraintes temporelles entre les noeuds temporels $t_s(a_i)$ et $t_e(a_i)$ de T lors de l'ajout de a_i à un plan. Les contraintes temporelles, en terme de distance minimale $d(t_x(a_i), t_y(a_j))$ entre deux instants, sont les suivantes :

Contrainte de Durée	Contraintes Temporelles
$\delta_i = c$	$(d(t_s(a_i), t_e(a_i)) = c) \wedge (d(t_e(a_i), t_s(a_i)) = - c)$
$\delta_i \leq c$	$d(t_s(a_i), t_e(a_i)) \leq c$
$\delta_i \geq c$	$d(t_e(a_i), t_s(a_i)) \leq - c$

La sémantique de PDDL2.1 impose que chaque action doit démarrer strictement après l'instant zéro. Soit ε la plus petite fraction de temps requise pour séparer deux instants. Pour garantir qu'une action ajoutée a_i démarre après l'instant zéro, la contrainte temporelle $d(t_s(a_i), t_0) \leq - \varepsilon$ est ajoutée. Une annotation temporelle $\tau \in \{start, inv, end\}$ est ajoutée à la représentation des conditions ouvertes : *start* représente une condition qui doit être vérifiée au début de l'action, *inv* une condition invariante et *end* une condition qui doit être vérifiée à la fin de l'action. Une annotation équivalente est ajoutée à la représentation de liens causaux. La liaison d'une condition ouverte annotée τ à un effet associé à un instant $t_x(a_j)$ implique la contrainte temporelle $d(t_k(a_i), t_x(a_j)) \leq - \varepsilon$ (k = s si τ = *end*, sinon k = e). Les liens causaux menacés sont fondamentalement résolus de la même façon que dans le cadre classique, mais au lieu d'ajouter des contraintes d'ordre entre les actions ce sont des contraintes temporelles entre les instants qui sont ajoutées. Pour garantir que l'instant $t_x(a_i)$ précède l'instant $t_y(a_j)$ il faut ajouter la contrainte temporelle $d(t_y(a_j), t_x(a_i)) \leq - \varepsilon$. A chaque fois qu'une contrainte temporelle est ajoutée à un plan, les distances $d(t_x(a_i), t_y(a_j))$ qui peuvent avoir été affectées sont mises à jour. La complexité en temps de cette propagation de contraintes est quadratique en fonction du nombre d'actions du plan.

Dès qu'un plan sans défaut est trouvé, il faut ordonner les actions dans le plan, c'est-à-dire assigner un temps de début et une durée pour chaque action. Un ordonnancement des actions est une solution du STN T. La représentation du d-graphe fournit une solution qui assigne un instant de début le plus tôt possible à chaque action. Le temps de début d'action a_i est fixé à $-d(t_s(a_i), t_0)$ [Dechter, Mieri, Pearl, 1991, Corollaire 3.2] et la durée à $d(t_s(a_i), t_0) - d(t_e(a_i), t_0)$.

L'espace mémoire nécessaire à la planification POP temporelle est bien plus important que celui de la planification POP classique. En effet, dans le cadre classique, l'ordre partiel O peut être stocké comme une matrice de bits qui représente la fermeture transitive des contraintes d'ordre dans O. Un plan partiel de n actions nécessite donc n^2 bits. Si l'on considère un plan partiel avec n actions duratives, il est alors nécessaire de stocker $4n^2$ nombres à virgule flottante qui représentent le d-graphe de T. Chacun de ces nombres nécessitent au moins 32 bits sur une machine moderne, ce qui représente un espace plus de 100 fois supérieur pour représenter des contraintes temporelles à la place de contraintes classiques. Le fait que chaque raffinement ne change que peu d'entrées dans le d-graphe permet néanmoins de répartir les informations entre les différents plans partiels. La taille de la mémoire nécessaire peut alors être réduite grace à une représentation compacte des matrices, ce qui rend cette méthode utilisable en pratique comme l'ont montré les résultats honorables de VHPOP lors de la compétition IPC'2002.

Ces techniques de type POP sont aussi particulièrement intéressantes pour chercher des plans optimaux ; le planificateur CPT2 [Vidal, Geffner, 2006] est ainsi l'un des planificateurs temporels qui obtient les meilleurs résultats. Il a été distingué lors de la compétition IPC'2006 dans la catégorie planificateurs temporels optimaux mais la représentation des actions choisie le limite aux langages temporellement simples.

c) Planificateurs utilisant les DTP

Une autre approche très récente et utilisée pour la première fois dans le planificateur DT-POP (Disjunctive Temporal Partial-Order Planner) [Schwartz, Pollack, 2004] consiste à réduire les branchements dans l'algorithme POP classique en utilisant un ensemble de contraintes temporelles disjonctives à la place du STN. Un système de résolution de DTP (Disjunctive Temporal Problem) doit alors être utilisé. Cette technique permet une forme de moindre engagement car, pour chacune des contraintes disjonctives, elle laisse au solveur le choix de la contrainte simple à satisfaire. DT-POP introduit la notion d'action temporelle disjonctive. Une dt-action <*Pre*, *Eff*, *D*> est un triplet où *Pre* est un ensemble de préconditions, *Eff* un ensemble d'effets, et *D* un ensemble de contraintes disjonctives (disjonctions de contraintes temporelles simples) sur les instants ou les bornes d'intervalles de *Pre* et *Eff*. Si les préconditions de *Pre* sont établies à des horaires consistants avec *D* et que l'action est exécutée, alors les effets de *Eff* sont produits à des horaires consistants avec *D*. DT-POP permet d'obtenir des plans-solutions temporels disjonctifs. Un dt-plan <*Need*, *Have*, *D*> est un triplet où *Need* est un ensemble de conditions qui représentent les buts et les préconditions des actions, *Have* un ensemble de conditions qui représentent les événements exogènes et les effets des actions, et *D* un ensemble de contraintes disjonctives sur les instants où les bornes d'intervalles de *Need* et *Have*. Dans DT-POP, les disjonctions temporelles ne sont utilisées que pour résoudre les menaces car leur utilisation pour la résolution des préconditions ouvertes imposerait l'ajout dans chaque nœud fils des disjonctions représentant toutes les manières de les établir. Cette façon de procéder produirait des dt-plans avec des contraintes temporelles trop complexes pour pouvoir être résolues dans un temps raisonnable par les solveurs DTP actuels.

Comme les planificateurs de type POP, GRAPHPLAN [Blum, Furst, 1995] (cf. chapitre II.5) a aussi fait l'objet d'extensions permettant de lui intégrer une dimension temporelle. Certains planificateurs temporels comme LPG-TD [Gerevini, Saetti, Serina, 2006] utilisent ainsi une méthode classique pour trouver un plan-solution séquentiel avant d'ordonner les actions dans le temps. Nous nous intéressons ici uniquement aux méthodes qui utilisent un graphe de

planification en intégrant la dimension temporelle à sa structure et à la phase d'extraction de solution.

2.3.3. Extensions temporelles de GRAPHPLAN

Les deux principales approches basées sur une utilisation d'un graphe de planification temporel sont une construction du graphe guidée par le temps ou une construction guidée par les actions comme dans le cadre classique.

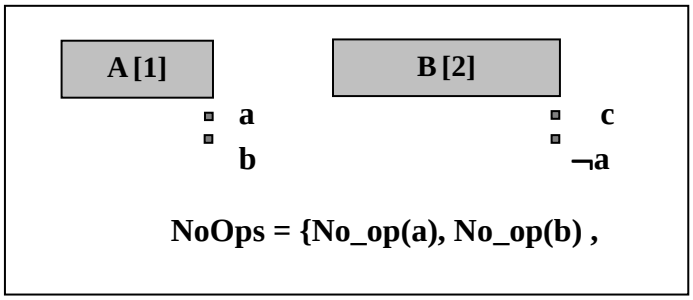
a) Construction d'un graphe de planification guidée par le temps

TGP (Temporal GraphPlan) [Smith, Weld, 1999] est le premier planificateur temporel basé sur GRAPHPLAN. Plusieurs systèmes comme TPSYS [Garrido, Fox, Long, 2002] ou CPPlanner [Dinh, Smith, 2003] ont ensuite utilisé un graphe de planification temporel comparable à celui de TGP. Dans tous ces systèmes, la construction du graphe est guidée par le temps, d'une manière comparable à ce que réalise la planification DEP en utilisant les époques de décision. Lors de la phase d'extraction, le choix des actions est d'abord guidé par le temps avant d'être guidé par les buts à atteindre.

TGP utilise un graphe de planification compact et une généralisation des relations d'exclusion mutuelles. Il augmente de manière incrémentale une représentation compacte du graphe de planification comprenant des actions de différentes durées. Les noeuds d'actions et de propositions sont annotés avec un label numérique représentant le premier niveau auquel elles apparaissent. Un label est également attaché aux mutex et aux nogoods pour représenter le dernier niveau pour lequel la relation est vérifiée. Avec cette représentation, les actions n'ont plus nécessairement une durée d'exécution unitaire ou discrète. Les labels peuvent être des nombres réels représentant les instants de démarrage plutôt que des nombres entiers qui marquent les niveaux d'un graphe de planification. Dans TGP, les préconditions doivent perdurer tout au long d'une instance d'action et ses effets ne doivent apparaître qu'en fin d'exécution. Cela signifie que deux actions ne peuvent en aucun cas être concourantes lorsque un effet ou une précondition de l'une est la négation d'un effet ou d'une précondition de l'autre.

Notations	Sémantiques
$\|A\|$	durée de l'action A
$[A$	instant auquel l'action A apparaît pour la première fois dans le graphe
$A]$	l'instant auquel une action A peut se terminer pour la première fois
$\lfloor A$ $\lfloor p$	instant auquel une instance de l'action A est démarrée instant où la proposition p devient vraie
$A\rceil$	instant de fin d'exécution d'une instance de l'action A
Propriétés : (1) $A] = [A + \|A\|$ (2) $A\rceil = \lfloor A + \|A\|$ (3) $[A \leq \lfloor A$	

Pour illustrer la nécessité d'un nouveau type de mutex dans le cadre temporel, prenons l'exemple simple suivant proposé par [Smith, Weld, 1999].



Ce problème simple illustre la nécessité des mutex action/proposition (Cf. Figure 15, ci-après). Dans la suite, une distinction est faite entre les mutex présents de manière permanente que l'on appelle emutex (eternal mutex) et ceux qui sont conditionnels appelés cmutex (conditional mutex) et qui disparaîtront au cours de l'expansion du graphe. Puisque les actions A et B produisent respectivement a et ¬a, elles ne peuvent être exécutées en même temps. Cette notion est formalisée par l'existence d'une emutex labellisée par ∞ sur le graphe de planification. Le seul moyen d'obtenir b et c consiste à exécuter A et B en série (l'ordre n'est pas important). Le mutex entre b et c devrait donc disparaître au temps t = 3. Mais la propagation classique de GRAPHPLAN est insuffisante pour déduire ce fait car les actions ont des durées d'exécution différentes. Alors que l'approche originale de GRAPHPLAN fonctionne lorsque les niveaux de propositions et d'actions se succèdent de façon régulière, les actions ayant des durées variables cassent cette symétrie. Par exemple, si b est établie et persiste jusqu'au temps t = 2, l'action qui génère c doit être développée suffisamment en amont pour se superposer à l'établisseur de b, et puisque B est superposée à A, il y a conflit. Cette limitation est éliminée grâce à l'introduction de mutex action/proposition. Nous pouvons remarquer que dans notre exemple, il est impossible d'avoir b vraie et l'action B en cours d'exécution au temps t = 2 si B commence son exécution avant que b devienne vraie. Intuitivement, les mutex action/proposition permettent de déduire un nombre plus important d'inconsistances parce qu'elles connectent mieux les mutex action/action aux mutex proposition/proposition dans le cas où les actions se superposent.

Les labels sur les cmutex représentent les conditions pour que ces mutex soient vérifiés : M_1 = "A est cmutex avec c si ⌊A < [c est vérifiée" et M_2 = "B est cmutex avec b si ⌊B < [b est vérifiée". La définition complète des mutex généralisés utilisés dans TGP est fournie en Annexe 5.

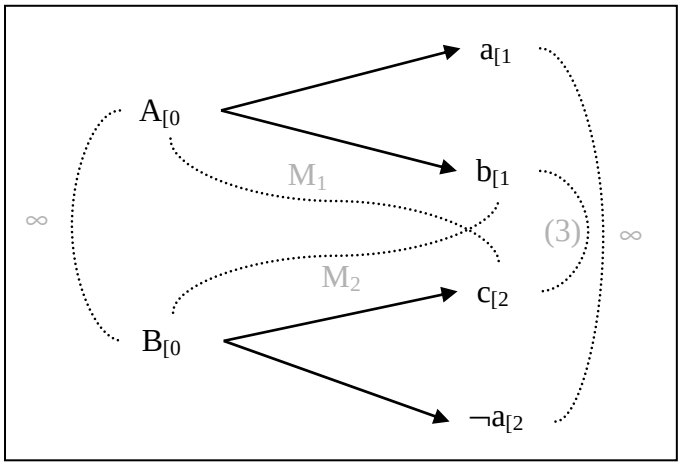


Figure 15 : Graphe de planification temporel (TGP)

En utilisant la représentation compacte du graphe de planification, il est possible de mettre à jour ce graphe en démarrant au temps t = 0 et en incrémentant progressivement le temps. Le planificateur peut garder une trace des modifications dans le graphe et examiner uniquement les propositions, actions et mutex qui peuvent être affectés par le changement à l'aide d'un algorithme récursif.

Pour cette phase d'expansion, TGP utilise deux listes principales ordonnées dans le temps *NewSupp* et *EndPPMutex*. *NewSupp* contient des triplets < A, p, t > représentant le fait que la proposition p a un nouveau support apporté par l'action A au temps t. *EndPPMutex* contient des couples < M, t > représentant le fait que M est un cmutex proposition/proposition qui disparaît au temps t. Pour plus d'efficacité, TGP gère également une liste temporaire *NewProps* qui est le sous-ensemble des propositions de *NewSupp* qui sont nouvelles (qui n'avaient pas, antérieurement, de support).

Une fois que le graphe de planification a été étendu à un temps t_G, où tous les buts sont présents, TGP effectue une recherche d'un plan solution par chaînage arrière. Cette recherche est implémentée en utilisant deux structures de données : *Agenda* et *Plan*. *Agenda* est une liste prioritaire de paires $< p_i, t_i >$, où p_i est une proposition du but (ou un sous-but) à asserter et t_i est le temps auquel elle doit être vérifiée. *Agenda* est initialisé en ajoutant $< g_i, t_G >$ à la liste pour chaque but de niveau supérieur g_i, et la liste est triée dans l'ordre temporel décroissant. La seconde structure de données, *Plan*, qui est initialement vide, enregistre le plan en construction comme un ensemble de paires $< A_i, s_i >$, où s_i est l'instant de démarrage de l'action A_i. Des actions de persistance pour le but, notées *persist-g*, qui correspondent aux No_ops (N_g) de GRAPHPLAN (cf. chapitre II.5), sont considérées explicitement puisqu'elles n'ont pas été ajoutées pendant l'expansion du graphe.

La boucle d'extraction de solution de TGP réalise les étapes suivantes tant que *Agenda* n'est pas vide :

1. Extraire < g, t > de *Agenda*.
2. Si (t = 0) $\wedge$ (g n'appartient pas à l'état initial), alors échec (backtrack). Si (t > 0) alors, soit S l'ensemble des actions A_i telles que chaque A_i a g pour effet et $A_i] \leq t$.
3. Choisir A dans S $\cup$ { *persist-g* } telle que A n'est mutex avec aucune des actions de *Plan*. Ajouter < A, t – |A| > à *Plan*, et pour chaque précondition p de A, ajouter < p, t - |A| > à *Agenda*. Si aucun A n'existe, backtrack. Si A est l'action spéciale "*persist-g*", |A| est égal au plus grand commun diviseur des durées de l'ensemble des actions et le test de mutex s'effectue avec la proposition g.

Dans TGP, les actions ont des effets seulement à la fin de l'exécution et les préconditions se maintiennent durant toute l'exécution. Dans TPSYS [Garrido, Fox, Long, 2002], PDDL 2.1 [Fox, Long, 2001a] est employé et les actions peuvent avoir des effets au début et à la fin de leur exécution. Dans CPPlanner [Dinh, Smith, 2003], la représentation d'actions a été étendue pour que les effets puissent se produire n'importe quand pendant l'exécution. Avec cette extension, les relations de mutex sont mémorisées comme contraintes pendant la phase d'expansion du graphe et elles sont revérifiées lors de l'extraction de la solution.

Une autre approche consiste, comme nous allons le voir, à construire le graphe de planification de manière classique [Blum, Furst, 1995] en étant guidé non pas par les instants auxquels une action peut être démarrée mais uniquement par le choix des actions. La

dimension temporelle doit alors être intégrée de manière différente : on ne doit pas pouvoir empêcher une action dont les préconditions sont satisfaites d'être ajoutée au graphe. Ceci n'est pas le cas lorsque l'on est guidé par le temps car le déclenchement des actions dépend alors uniquement de l'instant considéré.

b) Construction d'un graphe de planification guidée par les actions

Nous détaillons ici les méthodes employées dans LPGP [Long, Fox, 2003] et TM-LPSAT [Shin, Davis, 2004] dans lesquelles les actions duratives sont décomposées en trois actions instantanées. Les données temporelles y sont intégrées en ajoutant une variable représentant la durée pour chaque changement de niveau du graphe et un ensemble de contraintes entre ces variables. LPGP et TM-LPSAT diffèrent par la méthode choisie pour l'extraction de la solution. Si ces planificateurs font tous les deux appel à un solveur, LPGP utilise un algorithme de type GRAPHPLAN en recherche arrière dans le graphe de planification tout en maintenant la consistance d'une base de contraintes temporelles, alors que TM-LPSAT code entièrement le graphe de planification et les contraintes temporelles.

LPGP (Linear Programming and Graph Plan) [Long, Fox, 2003] est un planificateur qui utilise un modèle de programmation linéaire pour capturer les durées des actions. Chaque action A est décomposée en trois sous-actions : A-start, A-invariant et A-end. Pendant la construction, si l'action A-start apparaît à un niveau k du graphe, alors l'action A-invariant sera ajoutée au niveau k+1 et l'action A-end au niveau k+2. A chaque changement de niveau du graphe est associé une variable représentant sa durée. Lors de la phase d'extraction, les valeurs de ces variables seront alors contraintes en fonction des sous-actions sélectionnées. On pourrait être amené à sélectionner plusieurs fois la sous-action A-invariant (cf. Figure 16).

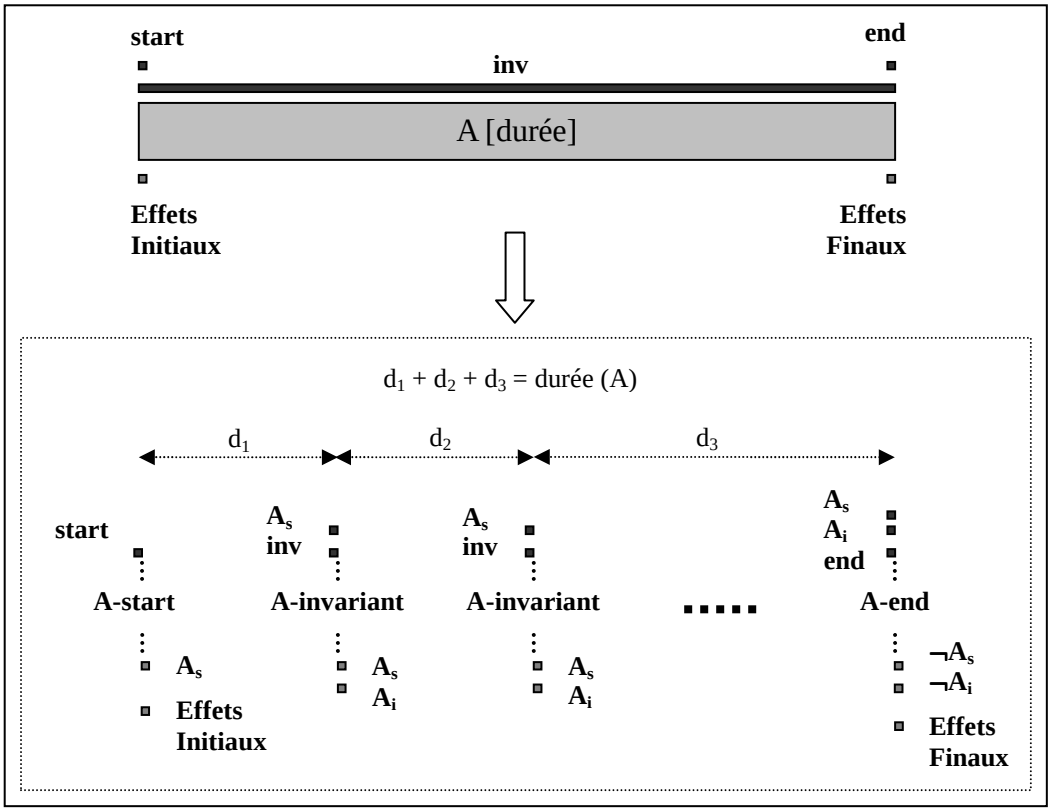


Figure 16 : décomposition d'une action durative selon le modèle temporel de LPGP

Cet ensemble de contraintes peut être codé sous la forme d'un STP (Simple Temporal Problem) et peut être résolu en temps polynomial par rapport au nombre de niveaux. Un inconvénient de cette méthode est que le nombre de niveaux augmente rapidement avec la taille des problèmes à cause de la décomposition de chaque action en trois sous-actions. Par contre, comme la dimension temporelle est gérée au travers d'une base de contraintes et non grace au graphe lui-même, on peut utiliser les relations de mutex classiques de GRAPHPLAN.

TM-LPSAT (Temporal Metric – LPSAT) [Shin, Davis, 2004] est aussi basé sur les graphes de planification et utilise, dans le cadre temporel, l'approche de la planification SAT (cf. partie III). Le système fait appel au solveur LPSAT [Wolfman, Weld, 1999] (utilisé à l'origine pour la planification avec ressources) qui intègre un solveur SAT couplé à un solveur de contraintes arithmétiques linéaires. TM-LPSAT utilise un codage spécifique dans lequel le temps est représenté comme une ressource particulière. Comme dans LPGP, les actions y sont découpées en trois sous-actions. Par contre, l'extraction d'une solution ne nécessite pas de procédure de recherche spécifique : le graphe de planification est codé en totalité et la recherche d'une solution entièrement confiée au solveur de contraintes. TM-LPSAT est composé de trois modules :

- Un compilateur, qui prend en entrée un domaine et un problème décrits en PDDL+ [Fox, Long, 2001, 2003] et renvoie un ensemble de contraintes sur des variables propositionnelles et numériques.
- Le solveur LPSAT qui est utilisé pour obtenir, si elle existe, une affectation des variables respectant l'ensemble des contraintes.
- Un décodeur qui prend en entrée l'affectation des variables et retourne un plan-solution.

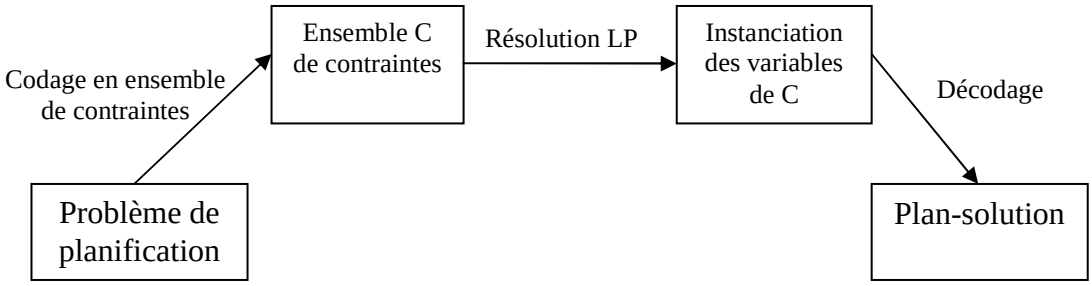


Figure 17 : schéma de principe de TM-LPSAT

TM-LPSAT est capable de prendre en compte des préconditions et des effets sur des ensembles d'instants ou d'intervalles quelconques, et des événements exogènes. Les actions peuvent aussi engendrer des effets continus dépendant d'une fonction temporelle linéaire. Les tests effectués sur le système avec les benchmarks de la catégorie "simple-time" d'IPC'2002 montrent de bonnes performances sur la résolution par LPSAT ou MathSat[1] de problèmes encodés [Shin, Davis, 2004]. TM-LPSAT reste cependant peu efficace si l'on considère les temps de compactification nécessaires au compilateur pour générer, à partir des ensembles de contraintes bruts (centaines de milliers de clauses), des ensembles de contraintes

[1] http://mathsat.itc.it/

raisonnablement exploitables (temps de compactification : Satellite-1 : 2 109 sec. / Zeno-2 : 7 711 sec. / Zeno-3 : 15 026 sec.). Si l'on considère le temps global de recherche de solution, LPGP est donc largement plus efficace sur les problèmes testés (temps global de résolution : Satellite-1 : 0,2 sec. / Zeno-2 : 5,5 sec. / Zeno-3 : 13,2 sec.).

Une méthode comparable à celle utilisée dans TM-LPSAT est décrite dans [Hu, 2007]. L'algorithme présenté réalise une décomposition des actions en deux actions simples et effectue un codage en CSP du problème selon une sémantique basée sur des opérateurs modaux. Dans ce codage, qui englobe le codage du graphe de planification, les invariants sont codés sous forme de contraintes. Cette approche permet de capturer une grande partie de l'expressivité de PDDL2.1/3, au-delà du cadre temporel. Elle n'a cependant pas été comparée à des planificateurs expressifs performants comme VHPOP ou LPGP.

Dans notre planificateur TLP-GP (cf. chapitre 3) nous utiliserons des méthodes similaires à celles de LPGP et TM-LPSAT, en donnant également la priorité au choix des actions sur les instants de démarrage. TLP-GP utilisera une représentation plus compacte des actions, une prise en compte différente de la dimension temporelle et déléguera une part plus importante du travail de recherche au solveur.

Après avoir présenté les techniques utilisées par les planificateurs temporels pour la prise en compte du temps, nous allons maintenant dresser une taxinomie de ces planificateurs. Les critères que nous considèrerons sont le type d'algorithme qu'ils utilisent et l'expressivité maximale du langage de représentation pour lequel la complétude de l'algorithme est garantie. Ce deuxième critère est pertinent car il donne une bonne indication de la capacité qu'ont ces planificateurs à résoudre des problèmes proches de problèmes réels.

2.4. Taxinomie des principaux planificateurs temporels

Indépendamment de l'algorithme utilisé, les planificateurs temporels peuvent être classés en deux catégories en terme d'expressivité temporelle (cf. section 2.1.3). Nous dressons ici un inventaire des différents types de planificateurs temporellement simples et temporellement expressifs.

2.4.1. Planificateurs temporellement simples

Si la recherche dans les espaces d'états étendus est actuellement la technique la plus performante, l’expressivité des langages de représentation employés reste faible. De nombreux planificateurs temporels sont basés sur cette technique car tout planificateur atemporel peut être modifié pour prendre en compte le temps en utilisant la technique DEP (cf. section 2.3.1) et / ou un réordonnancement d'une séquence d'actions. Ces méthodes permettent d’employer les techniques éprouvées de recherche heuristique utilisées en planification atemporelle.

a) Recherche dans les espaces d'états

TALPLAN [Kvarnström, Doherty, 2000], TLPLAN [Bacchus, Ady, 2001], TP4 [Haslum, Geffner, 2001], METRIC-FF [Hoffmann, 2002], SAPA [Do, Kambhampati, 2003], et MIPS [Edelkamp, Helmert, 2001] sont des planificateurs temporellement simples. C'est également le cas de SGPLAN [Chen, Wah, Hsu, 2006], qui a remporté les compétitions IPC'2004, IPC'2006 et IPC'2008, dans la catégorie des planificateurs temporels, ou de TFD (Temporal Fast Downward) [Röger, Eyerich, Mattmüller, 2008] qui a terminé deuxième de la

compétition IPC'2008. [Cushing et al., 2007.a] ont montré que les techniques DEP ou les techniques classiques suivies d'un réordonnancement des actions ne permettent pas de résoudre des problèmes temporellement expressifs. [Halsey, Long, Fox, 2004] proposent dans le planificateur CRIKEY une approche plus complète en terme d'expressivité. CRIKEY considère uniquement des solutions séquentielles et les réordonne pour obtenir des plans parallèles. Il essaie d'abord de planifier de manière classique mais, contrairement aux systèmes précédents, lorsqu'il ne trouve pas de plan, il bascule sur un algorithme à progression temporelle flottante. Malheureusement, [Cushing et al., 2007.a] montrent que, dans ce dernier cas, les règles d'élagage de l'arbre de recherche de CRIKEY peuvent supprimer des choix menant à un plan-solution et qu'il n'est donc pas complet pour les langages temporellement expressifs.

b) Recherche dans les espaces de plans partiels

Dans DEVISER [Vere, 1983], les préconditions d'une activité doivent être vraies au début et doivent le rester pendant toute la durée d'exécution. Les effets deviennent vrais dès que l'activité se termine. L'expressivité temporelle de ce précurseur des planificateurs temporels est donc du type L_e^o ce qui le classe dans la catégorie des planificateurs temporellement simples. Dans ZENO [Penberthy, Weld, 1994], les actions ne peuvent être concurrentes que lorsque leurs effets n'interfèrent pas. Certains problèmes temporellement expressifs ne peuvent donc pas être résolus, ce qui rend l'algorithme incomplet pour des problèmes nécessitant, pour leur résolution, des actions concurrentes de ce type. CPT [Geffner, Vidal, 2004] [Tabary, Vidal, 2006] [Vidal, 2008] est un planificateur POP temporel optimal en temps d'exécution qui est complet pour les sous-langages temporellement simples de PDDL2.1. Son langage de représentation est du type L_e^o ce qui le rend incapable de résoudre des problèmes temporellement expressifs. CPT n'est cependant optimal que pour des plans séquentiels (ou qui résultent de la projection des plans parallèles produits). Il est en effet facile de construire des problèmes temporellement simples mixtes pour lesquels les solutions optimales en temps d'exécution nécessitent des actions concurrentes et ne peuvent donc être trouvées que par un planificateur temporellement expressif optimal. DAE (Divide And Evolve) [Bibai, Savéant, Schoenauer, Vidal, 2008] décompose le problème en sous-problèmes et fait appel à CPT pour la résolution de ces derniers. CPT2 [Tabary, Vidal, 2006] a été distingué lors de la compétition IPC'2006 dans la catégorie "planificateurs temporels optimaux".

c) Extensions temporelles de GRAPHPLAN

La plupart des planificateurs temporels basés sur les graphes de planification sont temporellement simples. Certains d'entre eux, comme LPG-TD [Gerevini, Saetti, Serina, 2006], ont utilisé une méthode classique pour trouver un plan-solution séquentiel avant d'ordonner les actions dans le temps. TGP [Smith, Weld, 1999] et TPSYS [Garrido, Fox, Long, 2002] utilisent une approche comparable à la planification DEP dans les espaces d'états étendus en ce qui concerne le choix des instants possibles pour le démarrage d'une action (qui correspondent à l'état initial ou à des instants de production d'effets). TGP est donc complet pour son langage de représentation temporellement simple L_e^o. Par contre, TPSYS n'est pas complet puisqu'il utilise un langage $L_{s,e}^o$ temporellement expressif. Dans CPPLANNER [Dinh, Smith, 2003] les instants auxquels les préconditions doivent être vérifiées ne doivent pas obligatoirement correspondre à des instants de production d'effets. Le choix d'instants possibles est dans ce cas similaire à celui des époques de décisions de l'approche DEP+ utilisée dans les espaces d'états (cf. section 2.3.1). Le planificateur fournit donc des plans-

solutions optimaux en temps d'exécution pour des problèmes temporellement simples, mais il n'est pas complet pour son langage de représentation $L^{o}_{[s,e]}$ qui est temporellement expressif (super-langage de $L^{o}_{s,e}$). TPSYS et CPPLANNER sont néanmoins complets pour les sous-langages L^{o}_{s} et L^{o}_{e} qu'ils sont capables d'interpréter.

2.4.2. Planificateurs temporellement expressifs

a) Recherche dans les espaces d'états

CRIKEY3 [Coles, Fox, Long, Smith, 2008] est aujourd'hui le seul système implémenté de planification temporellement expressif basé sur une recherche dans les espaces d'états étendus. L'algorithme TEMPO proposé par [Cushing et al., 2007.a] permettra certainement dans le futur le développement de planificateurs utilisant une progression temporelle flottante. L'avantage principal de cet algorithme est qu'il permet d'utiliser les techniques de recherche heuristique existantes et très performantes dans les espaces d'états, tout en conservant l'accès aux informations sur un état temporel flottant.

b) Recherche dans les espaces de plans partiels

La grande majorité des planificateurs temporellement expressifs utilisent un algorithme de recherche dans les espaces de plans hiérarchiques de type HTN. La plupart de ces planificateurs utilisent une logique et un Time Map Manager (TMM) qui leur permet d'autoriser le recouvrement des actions et donc le parallélisme dans les plans-solutions. A partir du moment où ces systèmes sont complets pour un langage de représentation qui autorise le parallélisme, ils sont temporellement expressifs. C'est le cas des planificateurs basés sur une logique d'instants comme FORBIN [Dean, Firby, Miller, 1988], IXTET [Ghallab, Alaoui, 1989], [Ghallab, Laruelle, 1994], [Laborie, Ghallab, 1995], [Alaoui, 1990], TRIPTIC [Rutten, Hertzberg, 1993], TEST [Reichgelt, Shadbolt, 1990]. C'est également le cas des planificateurs basés sur une logique d'intervalles comme TIMELOGIC [Allen, Koomen, 1983], TLP [Tsang, 1987]. Leurs langages de représentation sont très expressifs mais présentent de nombreuses différences par rapport à PDDL2.1, ce qui les rend très difficiles à comparer aux autres planificateurs avec les benchmarks temporels classiques.

c) Extensions temporelles de GRAPHPLAN

Une approche différente de celle des systèmes GRAPHPLAN temporels comme TGP consite à choisir des instants de démarrage d'une action arbitrairement loin dans le temps. C'est l'approche utilisée dans LPGP [Long, Fox, 2003] et TM-LPSAT [Shin, Davis, 2004] qui sont par conséquent complets pour leurs langages de représentation respectifs $L^{s,o,e}_{s,e}$ et $L^{[s+,e+]}_{[s+,e+]}$.

Notre système TLP-GP est décrit en détail dans le chapitre 3. C'est un planificateur temporellement expressif. Il est complet pour son langage de représentation temporellement expressif $L^{[s+,e+]}_{[s+,e+]}$, super-langage de $L^{s,o,e}_{s,e}$ qui est lui même temporellement expressif d'après le théorème de [Cushing et al., 2007.a] (cf. section 2.1.3).

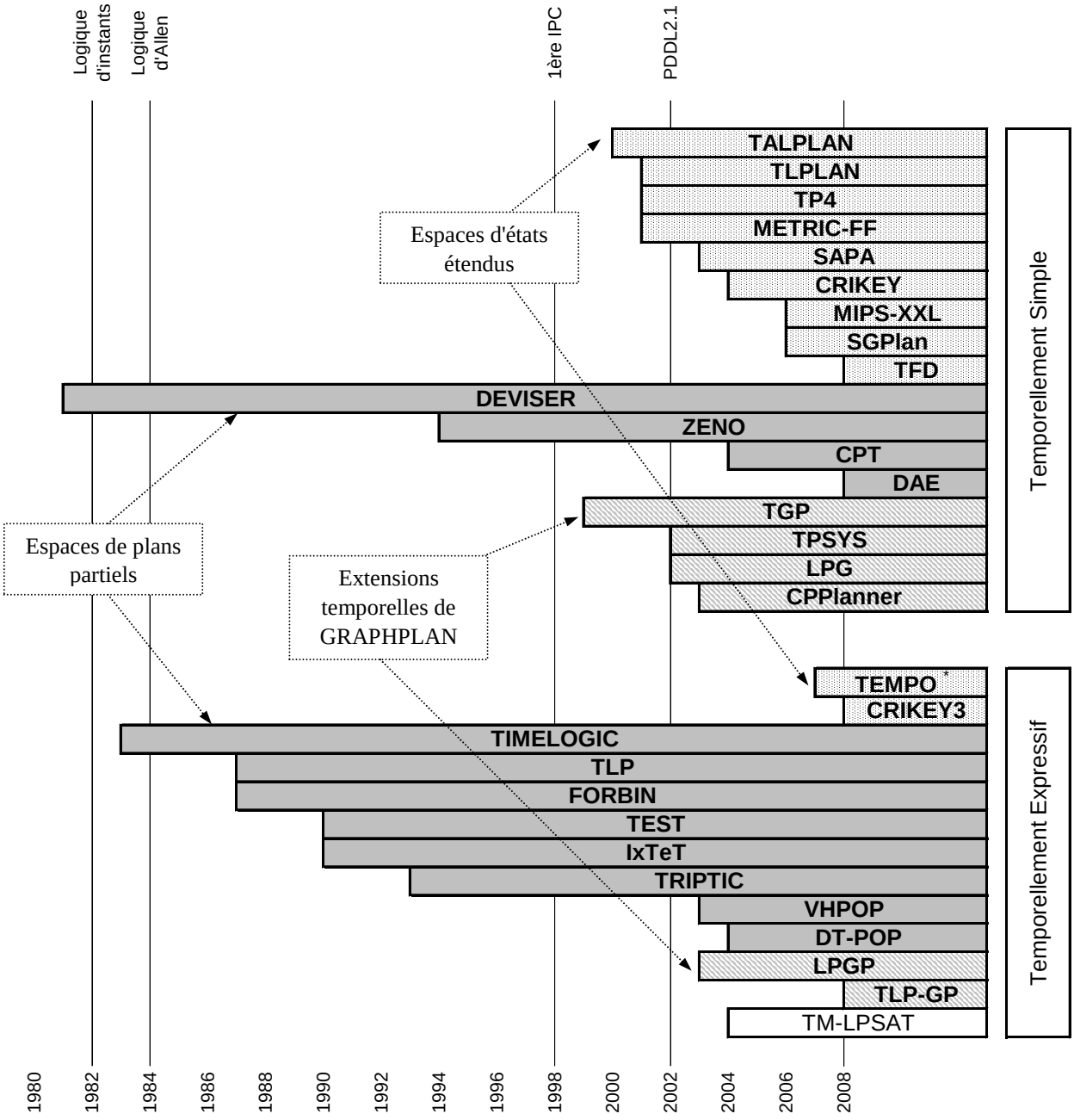
2.4.3. Synthèse

Voici maintenant un tableau récapitulatif des principaux planificateurs temporels en fonction du type d'algorithme utilisé et de l'expressivité maximale du langage de représentation assurant la complétude de l'algorithme. Les cellules grisées indiquent qu'il ne peut exister de planificateur (par exemple un planificateur qui utilise une technique DEP ne peut en aucun cas être temporellement expressif).

		Temporellement Simple	Temporellement Expressif	
		L_{e}^{o}	$L_{s,e}^{s,o,e}$	$L_{[s+,e+]}^{[s+,e+]}$
Espaces d'états étendus	*DEP*	TLPLAN TALPLAN TP4 METRIC-FF SAPA CRIKEY MIPS SGPLAN		
	DEP+			
	Progression Temporelle Flottante	(CRIKEY)*	CRIKEY3	(TEMPO)**
Espaces de plans	*HTN*	DEVISER ZENO		TIMELOGIC TLP FORBIN TEST IXTET TRIPTIC
	POP	CPT DAE	VHPOP	DT-POP
Extensions temporelles de GRAPHPLAN	*Guidée par le temps*	TGP TPSYS CPPLANNER		
	Guidée par les actions	LPG	LPGP	**TLP-GP-1**
Autres méthodes par compilation	*CSP*		Système de Hu	
	SAT / SMT		TM-LPSAT	**TLP-GP-2**

* complet uniquement pour les sous-langages temporellement simples de PDDL2.1
** algorithme non implémenté

Le schéma suivant montre que la large majorité des planificateurs temporellement expressifs actuels sont basés sur une recherche dans les espaces de plans, en particulier les anciens systèmes HTN basés sur un Time Map Manager (TMM). Bien qu'étant très expressifs, ces planificateurs sont lents et ne permettent pas d'envisager une résolution efficace de problèmes réels. Des algorithmes temporellement expressifs basés sur des techniques plus performantes comme la recherche dans les espaces d'états ou les graphes de planification commencent à voir le jour.



* algorithme non implémenté

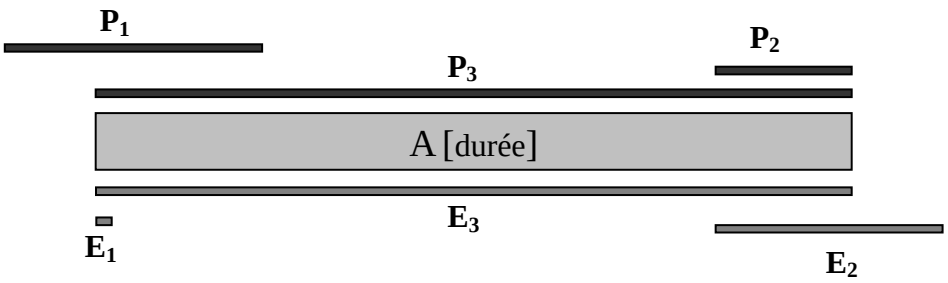
3. TLP-GP : un planificateur pour résoudre des problèmes temporellement-expressifs

3.1. Introduction

En dehors des premiers systèmes de type HTN (TIMELOGIC, TLP, FORBIN, TEST, IXTET, TRIPTIC…), il existe peu de planificateurs temporellement expressifs. Parmi ces premiers systèmes, aucun n'est capable de prendre en compte des problèmes codés en langage PDDL2.1. D'autre part, l'utilisation de solveurs DTP n'a été précédemment employée que dans le planificateur DT-POP [Schwartz, Pollack, 2004] sans résultats convaincants, son développement n'ayant pas été poursuivi. De la même manière, seul un petit nombre de travaux ont étudié l'utilisation de solveurs SMT (Sat Modulo Theory) pour résoudre des problèmes temporellement expressifs (LPGP, TM-LPSAT). Notre objectif étant le traitement de problèmes proches de problèmes réels, nous avons orienté notre travail sur l'extension de l'expressivité du langage de représentation de problèmes et sur l'utilisation de solveurs pour la résolution. Nous avons donc développé le système TLP-GP, un planificateur sain et complet pour l'ensemble des langages $L_{[s+,e+]}^{[s+,e+]}$. Ces langages sont temporellement expressifs puisque ce sont des super-langages de $L_{s,e}^{s,o,e}$ qui est lui-même temporellement expressif. TLP-GP est donc un planificateur temporellement expressif. Par contre, TLP-GP n'est pas optimal en temps d'exécution des plans-solutions puisqu'il s'arrête au premier plan-solution trouvé grace à l'utilisation d'une heuristique de type DVO (Domains Variables Ordering).

3.2. Extension de l'expressivité de PDDL2.1 aux intervalles temporels

[Smith, 2003] propose des extensions au langage PDDL 2.1 pour représenter des préconditions et des effets pouvant intervenir sur des intervalles quelconques liés au début ou à la fin de l'action (cf. section 2.1.2). Une action peut alors être représentée de la manière suivante :

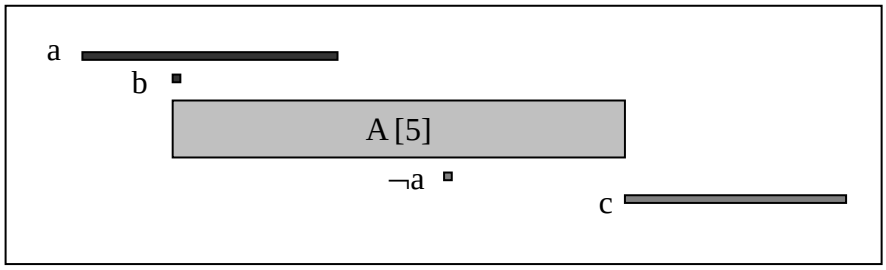


3.2.1. Prise en compte des extensions de [Smith, 2003]

Dans un premier temps, nous avons choisi d'intégrer au langage de TLP-GP les extensions proposées par [Smith, 2003]. Pour l'instant, aucune de ces extensions n'a encore été implémentée dans la version officielle du langage PDDL.

Dans TLP-GP, les actions sont représentées par des quadruplets (<nom-action>, <préconditions>, <effets>, <durée>) où <préconditions> et <effets> sont des ensembles de propositions associées à un label temporel. Ce label représente un intervalle sur les réels, relatif à l'instant de début de l'action (start) sur lequel une précondition doit être vérifiée ou un effet produit. Un label [t, t] ne contenant que la valeur t sera noté [t]. Pour une action, <durée> représentera sa durée. On notera $\tau_s(A)$ la variable temporelle correspondant à l'instant de démarrage (Start) de l'action A. De plus, nous utiliserons deux actions factices et instantanées A_I et A_G permettant respectivement d'établir l'état initial et de requérir les buts (Goals). $\tau_I = \tau_s(A_I)$ et $\tau_G = \tau_s(A_G)$ seront respectivement les instants de début et de fin du plan.

Exemple : soit l'action (A, $\{a_{[-1, 2]}, b_{[0]}\}$, $\{\neg a_{[3]}, c_{[5, 7]}\}$, 5). Si $\tau_s(A)$ est l'instant de début de A, la durée de A est 5, la proposition a doit être vérifiée entre $\tau_s(A)$-1 et $\tau_s(A)$+2, b doit être vérifiée à $\tau_s(A)$, ¬a apparaît à $\tau_s(A)$+3 et c apparaît à $\tau_s(A)$+5 et reste vraie au moins jusqu'à $\tau_s(A)$+7.



Dans la suite, nous utiliserons l'exemple de [Cushing et al., 2007.a, figure 3] modifié pour lui intégrer la prise en compte des intervalles (cf. Figure 18). Il nous servira ensuite, tout au long de cette partie à expliquer le fonctionnement de TLP-GP. Ce problème simple est particulièrement intéressant car il met en défaut tous les planificateurs temporellement simples.

Dans ce problème de planification $\prod$ = <O, I, G>, l'état initial est I = {}, le but est G = {b, d, e} et l'ensemble O des actions est : {(A, {}, $\{a_{[0\,;\,5[}, \neg a_{[5]}, b_{[5]}, \neg d_{[5]}\}$, 5) ; (B, $\{a_{[0]}\}$, $\{c_{[0\,;\,4[}, d_{[4]}, \neg c_{[4]}\}$, 4) ; (C, $\{c_{[0]}\}$, $\{\neg b_{[1]}, e_{[1]}\}$, 1)}.

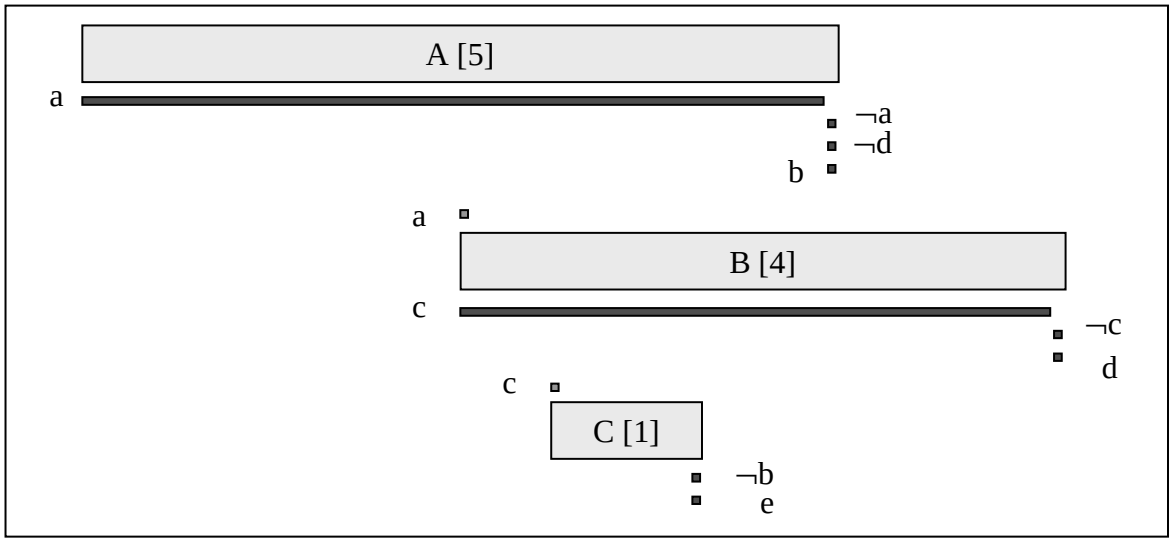


Figure 18 : exemple de problème temporellement expressif

Nous allons maintenant présenter la manière dont TLP-GP prend en compte les intervalles temporels pour résoudre ce type de problème.

3.2.2. Exemples d'utilisation de cette expressivité

Grâce à la prise en compte des intervalles temporels, le langage de TLP-GP permet de représenter de manière simple des buts temporellement étendus, des événements extérieurs, des fenêtres d'activation...

- Buts temporellement étendus : La prise en compte de buts temporellement étendus est réalisée en codant chacun de ces buts par une précondition de l'action terminale A_G, chacun d'entre eux étant labellisé par rapport à τ_G.
- Evènements extérieurs : La prise en compte d'événements extérieurs (ou de propositions initiales datées) est réalisée dans TLP-GP par l'ajout, à l'action factice A_I, d'effets ayant un intervalle temporel distinct de l'instant initial. Il suffit d'ajouter ces effets et ces données temporelles à la description de l'état initial du problème.
- Fenêtres d'activation : La prise en compte de fenêtres d'activation peut naturellement être formulée par l'ajout, à l'action factice A_I, d'effets factices positifs p en début de fenêtre et négatifs ¬p en fin de fenêtre. Les actions que l'on désire activer dans la fenêtre devront alors avoir p comme précondition factice.
- Délai entre actions : La prise en compte d'un délai minimum d_{min} entre deux actions A et B peut être réalisée par l'ajout d'un effet factice ¬p sur un intervalle temporel démarrant à la fin de l'action A (over [end (+ end d_{min})] (not p)), d'un effet factice p à la fin de cet intervalle (at (+ end d_{min}) p), et d'une précondition factice p en début d'action B (at start p). Un délai maximum d_{max} peut également être envisagé en ajoutant encore un effet factice ¬p à l'action A (at (+ end d_{max}) (not p)).

3.3. Algorithmique de TLP-GP

Le planificateur TLP-GP commence par construire un graphe de planification atemporel et sans mutex, sans prendre en compte ni la durée, ni l'instant de début des actions (section 3.3.1). Il cherche ensuite à extraire un plan solution, et il peut pour cela utiliser deux méthodes différentes que nous comparons entre elles :

- TLP-GP-1 (section 3.3.2) effectue une recherche arrière dans le graphe de planification en utilisant un solveur de problèmes temporels disjonctifs (DTP) pour vérifier, à chaque étape du chaînage arrière, la consistance d'un ensemble de contraintes temporelles. A chaque étape, ce système de contraintes est donné à un prouveur qui vérifie sa consistance.
- TLP-GP-2 (section 3.3.3) code le graphe dans une logique QF-RDL (Quantifier-Free Real Difference Logic) et l'ensemble de formules correspondant est donné en entrée à un solveur SMT qui lui cherche un modèle.

Quelle que soit la méthode utilisée, en cas d'échec, le graphe est étendu d'un niveau supplémentaire et l'opération renouvelée.

3.3.1. Expansion du graphe de planification temporel

Contrairement à ce que réalisent des planificateurs comme TGP ou LPGP, TLP-GP utilise un graphe de planification construit sans exclusions mutuelles et de manière atemporelle, sans

tenir compte dans un premier temps, ni de la durée, ni de l'instant de démarrage des actions. Les conflits entre actions, y compris les exclusions mutuelles, sont totalement gérés lors de la phase d'extraction de solution par un système de satisfaction de contraintes. Cet usage a minima du graphe de planification permet à TLP-GP de ne pas souffrir des mêmes limitations que d'autres planificateurs : époques de décisions qui limitent la complétude aux problèmes temporellement simples ou nombre de niveaux du graphe trop important pour permettre la résolution pratique de problèmes temporellement expressifs (LPGP). La solution que nous avons retenue consiste à confier une grande partie du travail de résolution à un solveur DTP. Elle permet de produire des plans temporels flottants et d'augmenter considérablement l'expressivité du langage de représentation utilisable. Cette stratégie s'est avérée très efficace dans le cadre de la planification classique puisque, grâce aux progrès considérables des solveurs SAT, le système SATPLAN'04[2], successeur du système BLACKBOX [Kautz, Selman, 1999] était toujours, sept ans plus tard, le premier planificateur optimal de la compétition IPC'06.

Comme une action peut produire et détruire une même proposition à des instants différents, on mémorise également dans le graphe les négations des propositions. Enfin, contrairement à ce qui est réalisé dans les autres planificateurs temporels basés sur GRAPHPLAN, les niveaux ne sont pas liés à une échelle temporelle.

Le niveau 0 comprend l'action factice A_I, sans préconditions, qui produit toutes les propositions de l'état initial et les arcs d'effets correspondants (dans notre exemple nous l'omettrons car I = {}). Pour chaque niveau $n \geq 1$, on applique ensuite les actions dont toutes les préconditions sont présentes au niveau n-1 et on ajoute les arcs préconditions au graphe. On ajoute alors les effets de ces actions au niveau n avec les arcs d'effets. Le graphe est ainsi étendu niveau par niveau jusqu'à l'apparition de l'ensemble des buts. Durant la construction, les arcs sont associés à un label temporel correspondant à leur nature :

- **arc de précondition** (proposition → action) : le label représente l'intervalle, relatif au début de l'action, sur lequel la précondition doit être vérifiée ;
- **arc d'effet** (action → proposition) : le label représente un intervalle, relatif au début de l'action, au début duquel l'effet apparaît, pendant lequel il est vrai, et à la fin duquel il n'est plus garanti qu'il soit maintenu.

On construit enfin un niveau supplémentaire comprenant l'action factice A_G, sans effets, ayant tous les buts pour préconditions, et on ajoute les arcs préconditions associés. L'algorithme d'extraction est alors appelé.

LPGP est le seul planificateur basé sur le graphe de planification qui puisse résoudre le problème de la Figure 18. Il le fait en choisissant des époques de décision en avant et en arrière, arbitrairement loin dans le temps. Cependant, même si les trois actions du plan sont présentes dès le niveau 4, il doit encore étendre le graphe jusqu'au niveau 6 pour obtenir une solution alors que TLP-GP trouve une solution à ce problème dès le niveau 3, sans retour arrière, ni augmentation du graphe. Le graphe construit pour cet exemple est donné ci-après.

[2] http://www.cs.rochester.edu/u/kautz/satplan/index.htm

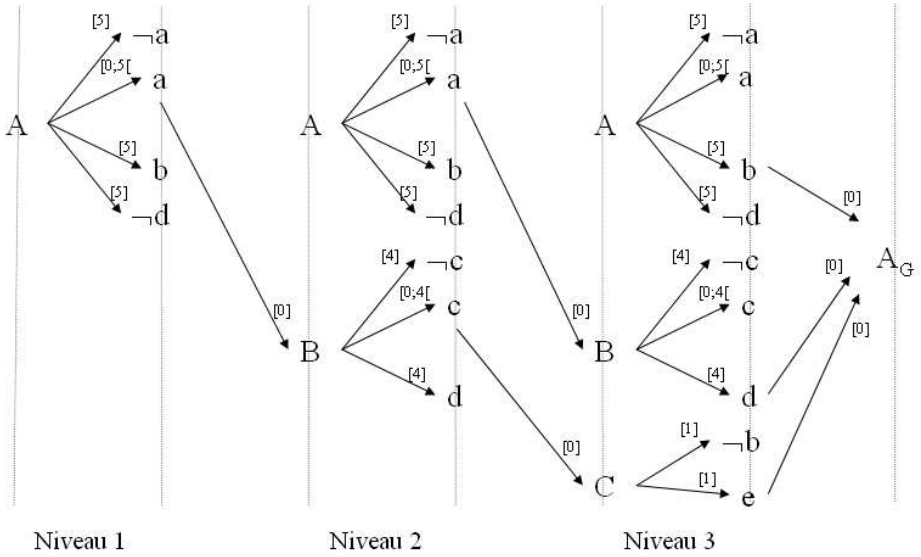


Figure 19 : expansion du graphe de planification de TLP-GP

Nous allons maintenant présenter la première procédure d'extraction de solution qui utilise une méthode classique de recherche arrière et un solveur DTP.

3.3.2. Extraction d'un plan-solution flottant en recherche arrière

Une fois que le graphe de planification a été étendu jusqu'à un niveau où tous les buts sont présents, TLP-GP-1 y recherche un plan-solution en chaînage arrière. Pour cela, il positionne des contraintes temporelles entre actions et, contrairement à des systèmes du type de TGP [Smith, Weld, 1999], à la place de la répartition fixe des actions et des propositions dans le temps, il emploie des variables temporelles sur lesquelles sont définies des contraintes. A chaque étape, ce système de contraintes est donné à un système de satisfaction de contraintes temporelles disjonctives (DTP) qui vérifie sa satisfiabilité. Cette opération se répète jusqu'à l'obtention d'une solution. En cas d'échec, le graphe est étendu d'un niveau supplémentaire et l'opération renouvelée. Comme deux feuilles distinctes de l'arbre de recherche peuvent correspondre à un même plan solution, TLP-GP-1 n'est pas systématique. Nous donnons l'algorithme d'extraction simplifié ci-après. La recherche d'un plan-solution utilise deux structures de données essentielles : *Agenda* et *Contraintes*.

Agenda est un ensemble de listes dont chacune est associée à une proposition. Pour une proposition p, la liste associée, *Agenda*(p), est composée d'intervalles temporels de la forme $[\tau_s(A)+\delta_1 ; \tau_s(B)+\delta_2]$, sur lesquels p doit être vraie. Lorsque l'intervalle ne contient qu'une valeur, on le note $[\tau_s(A)+\delta]$. Deux catégories d'intervalles temporels peuvent être ajoutés à l'*Agenda* :

- Les intervalles qui correspondent à une relation de causalité entre actions (maintien d'une proposition produite par une action pour être la précondition d'une autre action) ;
- Les intervalles qui correspondent à l'apparition des effets d'une action choisie.

Contraintes est une liste de disjonctions (ayant au plus deux littéraux) de contraintes temporelles binaires entre instants caractéristiques :

- Les contraintes correspondant aux relations de causalité représentent le fait qu'une précondition doit être produite par une action A à l'instant $\tau_s(A)+\delta_1$ avant d'être requise par une action B à l'instant $\tau_s(B)+\delta_2$. Elles ne sont pas disjonctives et sont de la forme $\tau_s(A)+\delta_1 \leq \tau_s(B)+\delta_2$.
- Les contraintes interdisant le recouvrement de deux intervalles $[\tau_s(A)+\delta_1, \tau_s(B)+\delta_2]$ et $[\tau_s(C)+\delta_3, \tau_s(D)+\delta_4]$ sur lesquels une proposition et sa négation sont respectivement vraies peuvent être disjonctives. Dans le cas général, ces contraintes sont de la forme $(\tau_s(B)+\delta_2 \leq \tau_s(C)+\delta_3) \vee (\tau_s(D)+\delta_4 \leq \tau_s(A)+\delta_1)$. Les inégalités peuvent être larges ou strictes en fonction du type des intervalles considérés (ouverts, fermés, mixtes). La plupart du temps, ces contraintes se simplifient en un unique littéral.

Le système de contraintes de TLP-GP-1 forme ainsi un problème temporel disjonctif. Dans le pire des cas, résoudre ou vérifier la consistance d'un DTP est NP-difficile [Dechter, Mieri, Pearl, 1991] mais des techniques récentes améliorent considérablement les performances des algorithmes de résolution de ces problèmes.

Algorithme d'extraction

```
Buts ← Pre(A_G) ;
Pour chaque effet e de Eff(A_I) :
   Ajouter un intervalle I d'apparition de la proposition e à Agenda(e) ;
Fin pour;
Tant que Buts ≠ ∅
   Pour chaque précondition p d'une action B, p ∈ Buts :
      Buts ← Buts – p ;
      Sélectionner (* point de backtrack *) en utilisant l'heuristique, une action A
      qui produit p pour B ;
      Buts ← Buts ∪ Pre(A) ;
      Poser une contrainte de précédence entre A et B ;
      Poser un intervalle I de maintien de précondition à Agenda(p) ;
      Pour chaque intervalle I' appartenant à Agenda(¬p) :
         Poser une contrainte interdisant le recouvrement de I et I'.
      Fin pour ;
      Pour chaque effet e de A, (sauf pour p lorsque le label de p est un
      singleton) :
         Ajouter un intervalle I d'apparition de la proposition e à Agenda(e) ;
         Pour chaque intervalle I' de Agenda(¬e) :
            Poser une contrainte interdisant le recouvrement de I et I'.
         Fin pour ;
      Fin pour ;
      Vérifier la consistance de Contraintes (appel au solveur DTP) ;
      En cas d'échec, retour au point de backtrack pour sélectionner une autre
      action A ;
   Fin pour ;
Fin tant que ;
Si Contraintes est consistant
   Alors retourner le plan-solution flottant (actions sélectionnées et Contraintes)
   Sinon il n'y a pas de solution à ce niveau du graphe ;
Fin si ;
Fin.
```

TLP-GP-1 offre la possibilité de guider la recherche par des heuristiques propres aux graphes de planification alors que, pour TLP-GP-2, l'ensemble de la recherche de solutions sera entièrement confié au solveur SMT. L'heuristique que nous avons implémentée pour sélectionner les sous-buts à établir et les actions qui les établissent est la suivante : priorité aux sous-buts qui apparaissent dans les niveaux les plus élevés du graphe et, pour les établir, priorité aux actions qui apparaissent dans les niveaux les moins élevés du graphe.

Pour initialiser la recherche, toutes les préconditions de A_G que l'on doit établir sont ajoutées à la liste des *Buts*. Les intervalles correspondant à l'apparition des effets de l'action A_I (propositions de l'état initial) sont ensuite ajoutés à l'*Agenda* pour optimiser la recherche.

Pour établir le but e (présent au niveau 3), choisissons l'action C au même niveau (cf. Figure 20). C_3 est donc sélectionnée. $[t_s(C_3)+1; t_G]$ est ajouté à *Agenda*(e) pour conserver le sous-but e jusqu'à l'instant t_G. $[t_s(C_3)+1]$ est ajouté à *Agenda*(¬b) pour prendre en compte l'apparition de l'effet ¬b.

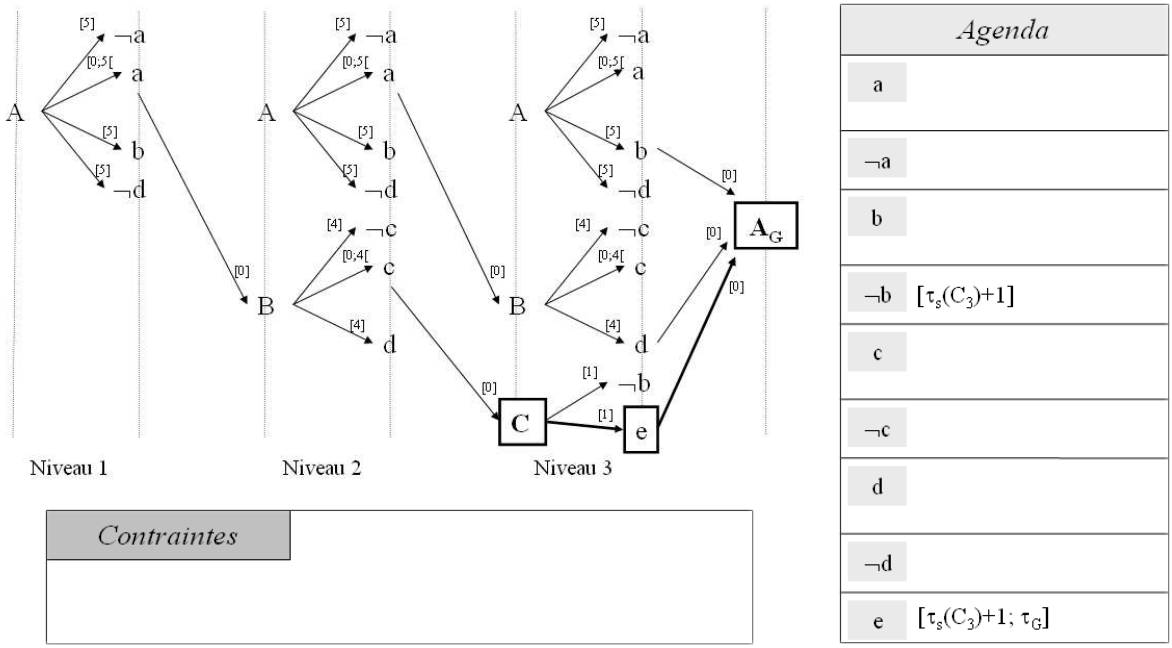


Figure 20 : extraction d'une solution par TLP-GP-1 (étape 1)

Nous choisissons ensuite l'action B au niveau 2 pour établir c, précondition de C_3. B_2 est sélectionnée et $[t_s(B_2); t_s(C_3)]$ est ajouté à *Agenda*(c) pour maintenir la précondition c le temps nécessaire (cf. Figure 21). $t_s(B_2) \leq t_s(C_3)$ est ajoutée aux *Contraintes* pour s'assurer que cette précondition soit produite avant d'être requise. $[t_s(B_2)+4]$ est ajouté à Agenda(¬c) et à Agenda(d) pour prendre en compte l'apparition des effets ¬c et d. Pour prévenir les conflits entre c et ¬c (mis en évidence par l'*Agenda*), nous ajoutons la contrainte disjonctive $(t_s(C_3)<t_s(B_2)+4) \vee (t_s(B_2)+4<t_s(B_2))$ qui se simplifie en $t_s(C_3)<t_s(B_2)+4$.

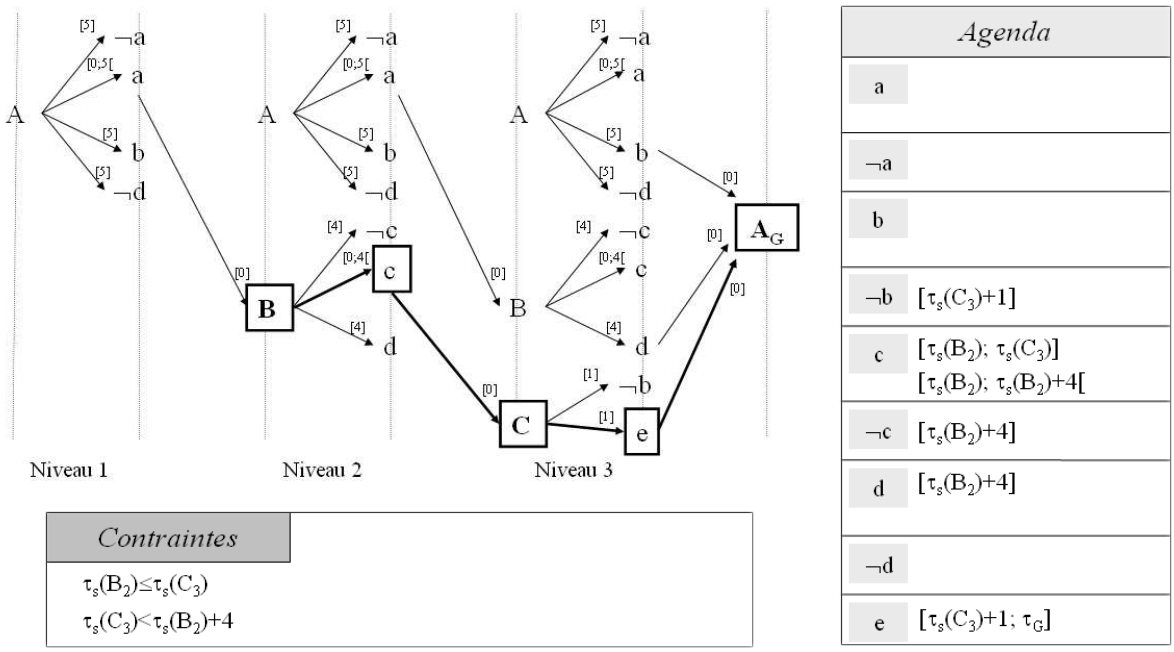


Figure 21 : extraction d'une solution par TLP-GP-1 (étape 2)

Nous sélectionnons ensuite l'action A_1 pour établir a, précondition de B_2. $[t_s(A_1); t_s(B_2)]$ est ajouté à *Agenda*(a) pour maintenir la précondition a et $t_s(A_1) \leq t_s(B_2)$ est ajoutée aux *Contraintes* (cf. Figure 22). $[t_s(A_1)+5]$ est ajouté à Agenda(¬a), à Agenda(b) et à Agenda(¬d) pour prendre en compte l'apparition des effets ¬a, b et ¬d. Pour prévenir le conflit entre a et ¬a, nous ajoutons la contrainte disjonctive $(t_s(B_2)<t_s(A_1)+5) \vee (t_s(A_1)>t_s(A_1)+5)$ qui se simplifie en $t_s(B_2)<t_s(A_1)+5$. Pour prévenir le conflit entre b et ¬b, nous ajoutons la contrainte disjonctive $(t_s(A_1)+5<t_s(C_3)+1) \vee (t_s(A_1)+5>t_s(C_3)+1)$ qui équivaut, pour cet exemple simple, à la contrainte $t_s(A_1)+5 \neq t_s(C_3)+1$. Pour prévenir le conflit entre d et ¬d, nous ajoutons une nouvelle contrainte disjonctive $(t_s(A_1)+5<t_s(B_2)+4) \vee (t_s(A_1)+5>t_s(B_2)+4)$ qui équivaut à la contrainte $t_s(A_1)+5 \neq t_s(B_2)+4$.

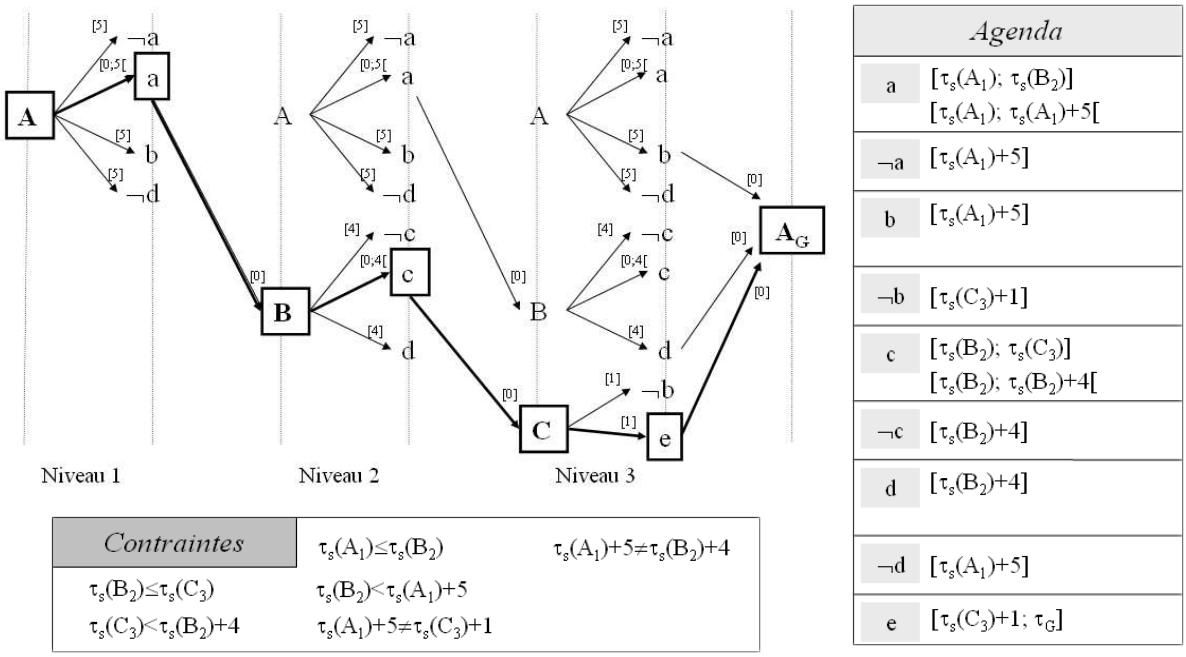


Figure 22 : extraction d'une solution par TLP-GP-1 (étape 3)

Nous allons maintenant chercher à établir le but d (cf. Figure 23) : d'après l'Agenda, cette proposition est déjà établie à $t_s(B_2)+4$. $[t_s(B_2)+4;\ t_G]$ est donc ajouté à *Agenda*(d) pour conserver le sous-but d jusqu'à l'instant t_G. Pour prévenir le conflit entre d et ¬d, nous ajoutons la contrainte disjonctive $(t_s(A_1)+5<t_s(B_2)+4) \vee (t_s(A_1)+5>t_G)$ qui se simplifie en $t_s(A_1)+5<t_s(B_2)+4$.

Etablissons enfin le but b : d'après l'Agenda, cette proposition est déjà établie à $t_s(A_1)+5$. $[t_s(A_1)+5;\ t_G]$ est ajouté à *Agenda*(b) pour conserver b jusqu'à l'instant t_G. Pour prévenir le conflit entre b et ¬b, nous ajoutons la contrainte $t_s(C_3)+1<t_s(A_1)+5$.

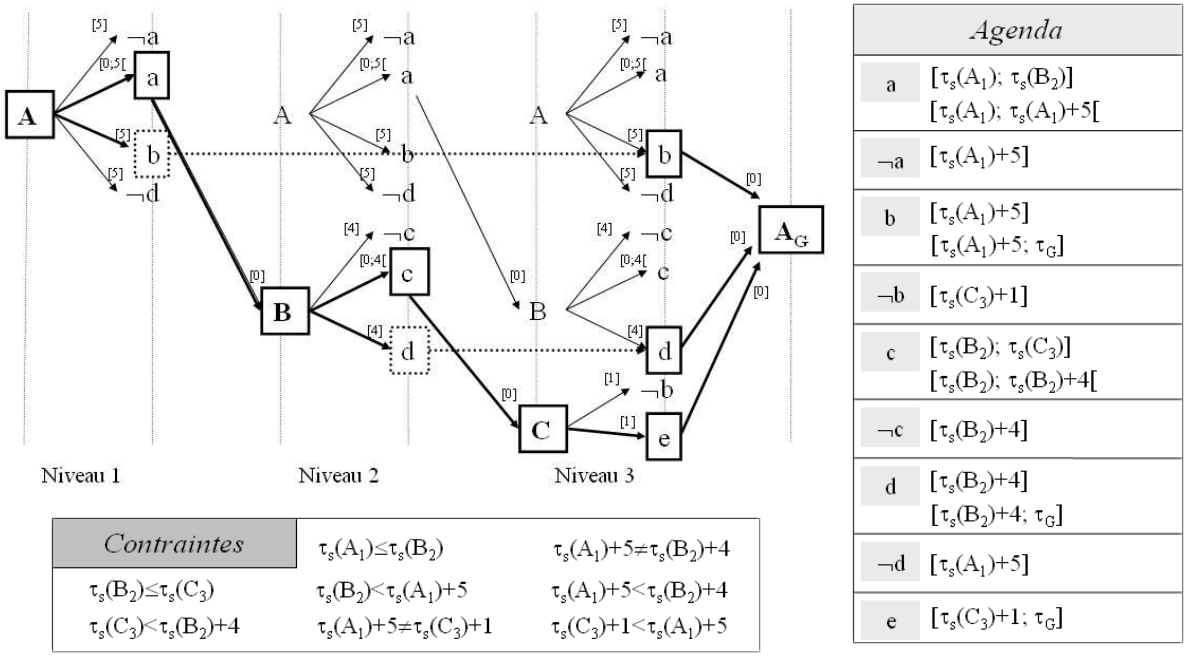


Figure 23 : extraction d'une solution par TLP-GP-1 (étape 4)

Comme à chacune des étapes précédentes, l'ensemble des contraintes temporelles demeure satisfiable, ce qui nous permet d'obtenir un plan-solution flottant.

Définition 52 (plan-solution flottant)

Un plan-solution flottant est un triplet <A, V, T> où A est un ensemble d'actions, V un ensemble de variables caractéristiques des actions de A (en PDDL2.1, start ou end) et T un ensemble consistant de contraintes temporelles simples entre variables de V.

L'algorithme précédent permet de construire un plan-solution flottant (ensemble des actions sélectionnées et des contraintes) dont nous donnons une représentation graphique à la Figure 24. Les instants $\tau_s(x)+\delta$ définis relativement à l'instant de démarrage $\tau_s(x)$ d'une action x qui apparaît dans l'ensemble des contraintes sont représentés par des lignes verticales discontinues. Chaque action est représentée par un intervalle pouvant être décalé dans le temps. Les traits continus entre les instants liés aux actions et intervenant dans le système de contraintes permettent d'indiquer l'amplitude maximale de ces glissements temporels. Dans l'exemple, le début de B_2 pourrait glisser jusqu'au début de A_1 (contrainte représentée par une trait continu entre a et a sur la Figure 24), mais la fin de A_1 doit aussi strictement demeurer

avant la fin de B_2 (contrainte représentée par un trait continu entre a et ¬a sur la Figure 24). Le respect de ces contraintes permet ainsi au but d, établi par B_2, de rester vrai bien qu'étant détruit par A_1.

Le plan-solution flottant est trouvé sans backtrack par TLP-GP au niveau 3. Il possède une grande flexibilité, et on peut facilement optimiser son temps d'exécution en faisant démarrer toutes les actions le plus tôt possible.

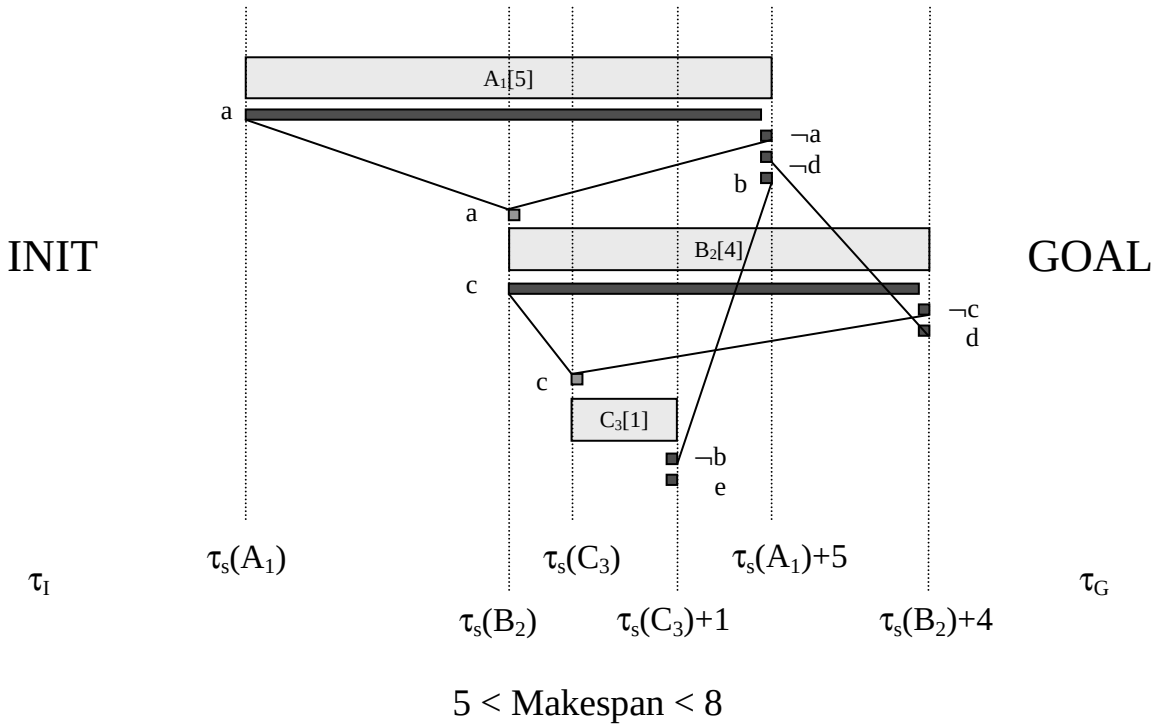


Figure 24 : plan solution flottant

Nous allons maintenant présenter une deuxième méthode d'extraction de solution qui est basée sur le codage de la totalité du graphe de planification. Notre objectif est ensuite de comparer ces deux techniques pour déterminer celle qui est généralement la plus performante.

3.3.3. Extraction d'un plan-solution flottant par codage du graphe

Pour extraire un plan-solution du graphe temporel, une autre méthode, que nous appellerons TLP-GP-2, consiste à produire une base de clauses intégrant à la fois la structure du graphe et les contraintes temporelles. Ceci peut être réalisé en utilisant la logique QF-RDL (quantifier-free, real difference logic) basée sur la théorie arithmétique linéaire sur les réels LA(R). Ces bases sont solubles par les solveurs SMT (Sat Modulo Theory) qui intégrent cette théorie (en particulier MathSat[3]). Si l'avantage de cette approche est de ne pas avoir à relancer systématiquement le solveur à chaque étape de la recherche, il est impossible de diriger la recherche par des heuristiques directement issues de connaissances sur le domaine ou le problème de planification. Les règles qui permettent ce codage sont les suivantes :

[3] http://mathsat.itc.it/

- **Etat initial et But :** les nœuds d'actions factices Init (produisant l'état initial) et Goal (nécessitant le but) sont tous deux vrais.

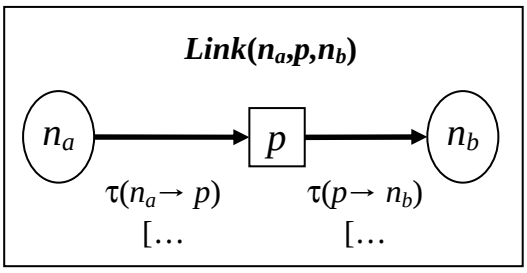
$$1. \quad n_{Init} \wedge n_{Goal}$$

Exemple :

$A_I \wedge A_G$

- **Production des préconditions par liens causaux :** si une action B est active dans le plan, alors pour chacune de ses préconditions p, il existe au moins un lien causal (noté Link(n_A, p, n_B)) de l'action A, qui produit cette précondition, vers B.

$$2. \quad \bigwedge_{(n_p,n_b)\in Arcs\,Prec} \left(n_b \Rightarrow \bigvee_{(n_a,n_p)\in ArcsAdd} Link(n_a, p, n_b) \right)$$



Exemple :

$B_2 \Rightarrow Link(A_1,a,B_2)$
$B_3 \Rightarrow Link(A_1,a,B_3) \vee Link(A_2,a,B_3)$
$C_3 \Rightarrow Link(B_2,c,C_3)$
$A_G \Rightarrow Link(A_1,b,A_G) \vee Link(A_2,b,A_G) \vee Link(A_3,b,A_G)$
$A_G \Rightarrow Link(B_2,d,A_G) \vee Link(B_3,d,A_G)$
$A_G \Rightarrow Link(C_3,e,A_G)$

- **Activation des actions et ordre partiel :** s'il existe un lien causal entre une action A qui produit une précondition p pour une action B, alors A et B sont actives dans le plan et l'instant où A produit certainement p est antérieur ou égal à l'instant où B commence à nécessiter p.

$$3. \quad \bigwedge_{(n_a,n_p)\in ArcsAdd} \bigwedge_{(n_p,n_b)\in Arcs\,Prec} \left(Link(n_a, p, n_b) \Rightarrow \left(\begin{array}{l} n_a \wedge n_b \\ \wedge \left(\tau(n_a \underset{begin}{\rightarrow} p) \leq \tau(p \underset{begin}{\rightarrow} n_b) \right) \end{array} \right) \right)$$

Exemple :

$Link(A_1,a,B_2) \Rightarrow (A_1 \wedge B_2 \wedge (\tau(A_1\rightarrow a) \leq \tau(a\rightarrow B_2)))$
$Link(A_1,a,B_3) \Rightarrow (A_1 \wedge B_3 \wedge (\tau(A_1\rightarrow a) \leq \tau(a\rightarrow B_3)))$
$Link(A_2,a,B_3) \Rightarrow (A_2 \wedge B_3 \wedge (\tau(A_2\rightarrow a) \leq \tau(a\rightarrow B_3)))$
$Link(B_2,c,C_3) \Rightarrow (B_2 \wedge C_3 \wedge (\tau(B_2\rightarrow c) \leq \tau(c\rightarrow C_3)))$

$Link(A_1,b,A_G) \Rightarrow (A_1 \wedge A_G \wedge (\tau(A_1 \rightarrow b) \leq \tau(b \rightarrow A_G)))$
$Link(A_2,b,A_G) \Rightarrow (A_2 \wedge A_G \wedge (\tau(A_2 \rightarrow b) \leq \tau(b \rightarrow A_G)))$
$Link(A_3,b,A_G) \Rightarrow (A_3 \wedge A_G \wedge (\tau(A_3 \rightarrow b) \leq \tau(b \rightarrow A_G)))$
$Link(B_2,d,A_G) \Rightarrow (B_2 \wedge A_G \wedge (\tau(B_2 \rightarrow d) \leq \tau(d \rightarrow A_G)))$
$Link(B_3,d,A_G) \Rightarrow (B_3 \wedge A_G \wedge (\tau(B_3 \rightarrow d) \leq \tau(d \rightarrow A_G)))$
$Link(C_3,e,A_G) \Rightarrow (C_3 \wedge A_G \wedge (\tau(C_3 \rightarrow e) \leq \tau(e \rightarrow A_G)))$

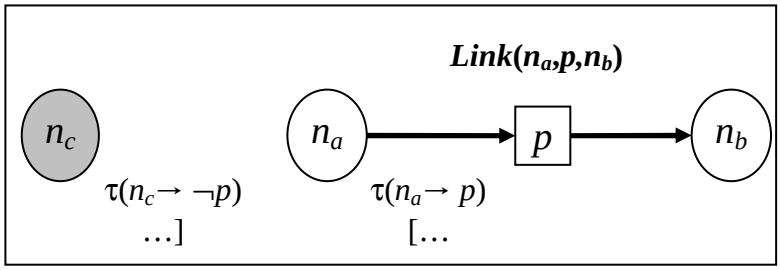
$Link(A_1,a,B_2) \Rightarrow (A_1 \wedge B_2 \wedge (\tau_s(A_1) \leq \tau_s(B_2)))$
$Link(A_1,a,B_3) \Rightarrow (A_1 \wedge B_3 \wedge (\tau_s(A_1) \leq \tau_s(B_3)))$
$Link(A_2,a,B_3) \Rightarrow (A_2 \wedge B_3 \wedge (\tau_s(A_2) \leq \tau_s(B_3))))$
$Link(B_2,c,C_3) \Rightarrow (B_2 \wedge C_3 \wedge (\tau_s(B_2) \leq \tau_s(C_3)))$
$Link(A_1,b,A_G) \Rightarrow (A_1 \wedge A_G \wedge (\tau_s(A_1)+5 \leq \tau(A_G)))$
$Link(A_2,b,A_G) \Rightarrow (A_2 \wedge A_G \wedge (\tau_s(A_2)+5 \leq \tau(A_G)))$
$Link(A_3,b,A_G) \Rightarrow (A_3 \wedge A_G \wedge (\tau_s(A_3)+5 \leq \tau(A_G)))$
$Link(B_2,d,A_G) \Rightarrow (B_2 \wedge A_G \wedge (\tau_s(B_2)+4 \leq \tau(A_G)))$
$Link(B_3,d,A_G) \Rightarrow (B_3 \wedge A_G \wedge (\tau_s(B_3)+4 \leq \tau(A_G)))$
$Link(C_3,e,A_G) \Rightarrow (C_3 \wedge A_G \wedge (\tau_s(C_3)+1 \leq \tau(A_G)))$

- **Exclusions mutuelles temporellement étendues (1) :** si un lien causal assure la protection d'une proposition p et qu'une action produisant sa négation est active dans le plan, alors l'intervalle temporel correspondant au lien causal et l'intervalle temporel correspondant à l'activation de ¬p par l'action sont disjoints.

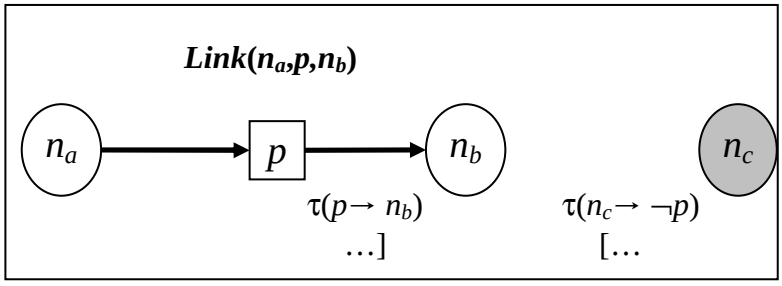
4.1 $$\bigwedge_{(n_a,n_p)\in ArcsAdd} \bigwedge_{(n_p,n_b)\in Arcs\,Prec} \bigwedge_{(n_c,n_{\neg p})\in ArcsDel} \left(\left(Link(n_a, p, n_b) \wedge n_c\right) \Rightarrow \left(\begin{array}{l} \left(\tau(n_c \underset{end}{\rightarrow} \neg p) < \tau(n_a \underset{begin}{\rightarrow} p)\right) \\ \vee \left(\tau(p \underset{end}{\rightarrow} n_b) < \tau(n_c \underset{begin}{\rightarrow} \neg p)\right) \end{array} \right) \right)$$

Comme dans les espaces de plans partiels, le conflit peut être évité par promotion ou rétrogradation.

1er cas : le conflit est évité en avançant l'action c dans le temps



2ème cas : le conflit est évité en reculant l'action c dans le temps



Exemple :

$(Link(A_1,a,B_2) \wedge A_1) \Rightarrow (((\tau(A_1 \rightarrow \neg a) < \tau(A_1 \rightarrow a)) \vee (\tau(a \rightarrow B_2) < \tau(A_1 \rightarrow \neg a)))$
$(Link(A_1,a,B_2) \wedge A_2) \Rightarrow (((\tau(A_2 \rightarrow \neg a) < \tau(A_1 \rightarrow a)) \vee (\tau(a \rightarrow B_2) < \tau(A_2 \rightarrow \neg a)))$
$(Link(A_1,a,B_2) \wedge A_3) \Rightarrow (((\tau(A_3 \rightarrow \neg a) < \tau(A_1 \rightarrow a)) \vee (\tau(a \rightarrow B_2) < \tau(A_3 \rightarrow \neg a)))$
$(Link(A_1,a,B_3) \wedge A_1) \Rightarrow (((\tau(A_1 \rightarrow \neg a) < \tau(A_1 \rightarrow a)) \vee (\tau(a \rightarrow B_3) < \tau(A_1 \rightarrow \neg a)))$
$(Link(A_1,a,B_3) \wedge A_2) \Rightarrow (((\tau(A_2 \rightarrow \neg a) < \tau(A_1 \rightarrow a)) \vee (\tau(a \rightarrow B_3) < \tau(A_2 \rightarrow \neg a)))$
$(Link(A_1,a,B_3) \wedge A_3) \Rightarrow (((\tau(A_3 \rightarrow \neg a) < \tau(A_1 \rightarrow a)) \vee (\tau(a \rightarrow B_3) < \tau(A_3 \rightarrow \neg a)))$
$(Link(A_2,a,B_3) \wedge A_1) \Rightarrow (((\tau(A_1 \rightarrow \neg a) < \tau(A_2 \rightarrow a)) \vee (\tau(a \rightarrow B_3) < \tau(A_1 \rightarrow \neg a)))$
$(Link(A_2,a,B_3) \wedge A_2) \Rightarrow (((\tau(A_2 \rightarrow \neg a) < \tau(A_2 \rightarrow a)) \vee (\tau(a \rightarrow B_3) < \tau(A_2 \rightarrow \neg a)))$
$(Link(A_2,a,B_3) \wedge A_3) \Rightarrow (((\tau(A_3 \rightarrow \neg a) < \tau(A_2 \rightarrow a)) \vee (\tau(a \rightarrow B_3) < \tau(A_3 \rightarrow \neg a)))$
$(Link(B_2,c,C_3) \wedge B_2) \Rightarrow (((\tau(B_2 \rightarrow \neg c) < \tau(B_2 \rightarrow c)) \vee (\tau(c \rightarrow C_3) < \tau(B_2 \rightarrow \neg c)))$
$(Link(B_2,c,C_3) \wedge B_3) \Rightarrow (((\tau(B_3 \rightarrow \neg c) < \tau(B_2 \rightarrow c)) \vee (\tau(c \rightarrow C_3) < \tau(B_3 \rightarrow \neg c)))$
$(Link(B_2,d,A_G) \wedge A_1) \Rightarrow (((\tau(A_1 \rightarrow \neg d) < \tau(B_2 \rightarrow d)) \vee (\tau(d \rightarrow A_G) < \tau(A_1 \rightarrow \neg d)))$
$(Link(B_2,d,A_G) \wedge A_2) \Rightarrow (((\tau(A_2 \rightarrow \neg d) < \tau(B_2 \rightarrow d)) \vee (\tau(d \rightarrow A_G) < \tau(A_2 \rightarrow \neg d)))$
$(Link(B_2,d,A_G) \wedge A_3) \Rightarrow (((\tau(A_3 \rightarrow \neg d) < \tau(B_2 \rightarrow d)) \vee (\tau(d \rightarrow A_G) < \tau(A_3 \rightarrow \neg d)))$
$(Link(B_3,d,A_G) \wedge A_1) \Rightarrow (((\tau(A_1 \rightarrow \neg d) < \tau(B_3 \rightarrow d)) \vee (\tau(d \rightarrow A_G) < \tau(A_1 \rightarrow \neg d)))$
$(Link(B_3,d,A_G) \wedge A_2) \Rightarrow (((\tau(A_2 \rightarrow \neg d) < \tau(B_3 \rightarrow d)) \vee (\tau(d \rightarrow A_G) < \tau(A_2 \rightarrow \neg d)))$
$(Link(B_3,d,A_G) \wedge A_3) \Rightarrow (((\tau(A_3 \rightarrow \neg d) < \tau(B_3 \rightarrow d)) \vee (\tau(d \rightarrow A_G) < \tau(A_3 \rightarrow \neg d)))$

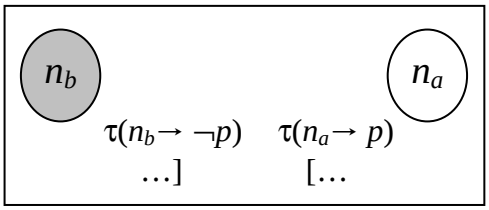
$(Link(A_1,a,B_2) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(A_1)) \vee (\tau_s(B_2) < \tau_s(A_1)+5))$
$(Link(A_1,a,B_2) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(A_1)) \vee (\tau_s(B_2) < \tau_s(A_2)+5))$
$(Link(A_1,a,B_2) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(A_1)) \vee (\tau_s(B_2) < \tau_s(A_3)+5))$
$(Link(A_1,a,B_3) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(A_1)) \vee (\tau_s(B_3) < \tau_s(A_1)+5))$
$(Link(A_1,a,B_3) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(A_1)) \vee (\tau_s(B_3) < \tau_s(A_2)+5))$
$(Link(A_1,a,B_3) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(A_1)) \vee (\tau_s(B_3) < \tau_s(A_3)+5))$
$(Link(A_2,a,B_3) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(A_2)) \vee (\tau_s(B_3) < \tau_s(A_1)+5))$
$(Link(A_2,a,B_3) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(A_2)) \vee (\tau_s(B_3) < \tau_s(A_2)+5))$
$(Link(A_2,a,B_3) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(A_2)) \vee (\tau_s(B_3) < \tau_s(A_3)+5))$
$(Link(B_2,c,C_3) \wedge B_2) \Rightarrow ((\tau_s(B_2)+4 < \tau_s(B_2)) \vee (\tau_s(C_3) < \tau_s(B_2)+4))$
$(Link(B_2,c,C_3) \wedge B_3) \Rightarrow ((\tau_s(B_3)+4 < \tau_s(B_2)) \vee (\tau_s(C_3) < \tau_s(B_3)+4))$
$(Link(B_2,d,A_G) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(B_2)+4) \vee (\tau(A_G) < \tau_s(A_1)+5))$
$(Link(B_2,d,A_G) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(B_2)+4) \vee (\tau(A_G) < \tau_s(A_2)+5))$
$(Link(B_2,d,A_G) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(B_2)+4) \vee (\tau(A_G) < \tau_s(A_3)+5))$
$(Link(B_3,d,A_G) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(B_3)+4) \vee (\tau(A_G) < \tau_s(A_1)+5))$
$(Link(B_3,d,A_G) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(B_3)+4) \vee (\tau(A_G) < \tau_s(A_2)+5))$
$(Link(B_3,d,A_G) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(B_3)+4) \vee (\tau(A_G) < \tau_s(A_3)+5))$

- **Exclusions mutuelles temporellement étendues (2) :** si deux actions produisant respectivement une proposition p et sa négation sont actives dans le plan, alors les intervalles temporels correspondants à l'activation de p et à l'activation de ¬p sont disjoints.

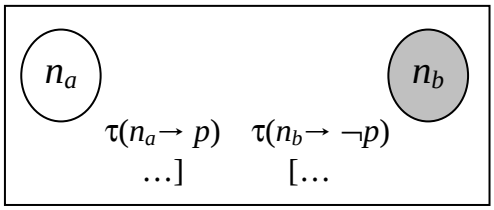
$$4.2 \quad \bigwedge_{(n_a,n_p)\in ArcsAdd} \bigwedge_{(n_b,n_{\neg p})\in ArcsDel} \left((n_a \wedge n_b) \Rightarrow \left(\begin{array}{l} \left(\tau(n_b \underset{end}{\rightarrow} \neg p) < \tau(n_a \underset{begin}{\rightarrow} p) \right) \\ \vee \left(\tau(n_a \underset{end}{\rightarrow} p) < \tau(n_b \underset{begin}{\rightarrow} \neg p) \right) \end{array} \right) \right)$$

Ici encore, le conflit peut être évité par promotion ou rétrogradation.

1er cas : le conflit est évité en avançant l'action b dans le temps



2ème cas : le conflit est évité en reculant l'action b dans le temps



Exemple :

$(A_1 \wedge A_2) \Rightarrow ((\tau(A_1 \rightarrow a) \neq \tau(A_2 \rightarrow \neg a))$
$(A_1 \wedge A_3) \Rightarrow ((\tau(A_1 \rightarrow a) \neq \tau(A_3 \rightarrow \neg a))$
$(A_1 \wedge C_3) \Rightarrow ((\tau(A_1 \rightarrow b) \neq \tau(C_3 \rightarrow \neg b))$
$(A_2 \wedge A_1) \Rightarrow ((\tau(A_2 \rightarrow a) \neq \tau(A_1 \rightarrow \neg a))$
$(A_2 \wedge A_3) \Rightarrow ((\tau(A_2 \rightarrow a) \neq \tau(A_3 \rightarrow \neg a))$
$(A_2 \wedge C_3) \Rightarrow ((\tau(A_2 \rightarrow b) \neq \tau(C_3 \rightarrow \neg b))$
$(A_3 \wedge A_1) \Rightarrow ((\tau(A_3 \rightarrow a) \neq \tau(A_1 \rightarrow \neg a))$
$(A_3 \wedge A_2) \Rightarrow ((\tau(A_3 \rightarrow a) \neq \tau(A_2 \rightarrow \neg a))$
$(A_3 \wedge C_3) \Rightarrow ((\tau(A_3 \rightarrow b) \neq \tau(C_3 \rightarrow \neg b))$
$(B_2 \wedge B_3) \Rightarrow ((\tau(B_2 \rightarrow c) \neq \tau(B_3 \rightarrow \neg c))$
$(B_3 \wedge B_2) \Rightarrow ((\tau(B_3 \rightarrow c) \neq \tau(B_2 \rightarrow \neg c))$
$(B_2 \wedge A_1) \Rightarrow ((\tau(B_2 \rightarrow d) \neq \tau(A_1 \rightarrow \neg d))$
$(B_2 \wedge A_2) \Rightarrow ((\tau(B_2 \rightarrow d) \neq \tau(A_2 \rightarrow \neg d))$
$(B_2 \wedge A_3) \Rightarrow ((\tau(B_2 \rightarrow d) \neq \tau(A_3 \rightarrow \neg d))$
$(B_3 \wedge A_1) \Rightarrow ((\tau(B_3 \rightarrow d) \neq \tau(A_1 \rightarrow \neg d))$
$(B_3 \wedge A_2) \Rightarrow ((\tau(B_3 \rightarrow d) \neq \tau(A_2 \rightarrow \neg d))$
$(B_3 \wedge A_3) \Rightarrow ((\tau(B_3 \rightarrow d) \neq \tau(A_3 \rightarrow \neg d))$

$(A_1 \wedge A_2) \Rightarrow (\tau_s(A_1) \neq \tau_s(A_2)+5)$
$(A_1 \wedge A_3) \Rightarrow (\tau_s(A_1) \neq \tau_s(A_3)+5)$
$(A_1 \wedge C_3) \Rightarrow (\tau_s(A_1)+5 \neq \tau_s(C_3)+1)$
$(A_2 \wedge A_1) \Rightarrow (\tau_s(A_2) \neq \tau_s(A_1)+5)$
$(A_2 \wedge A_3) \Rightarrow (\tau_s(A_2) \neq \tau_s(A_3)+5)$
$(A_2 \wedge C_3) \Rightarrow (\tau_s(A_2)+5 \neq \tau_s(C_3)+1)$
$(A_3 \wedge A_1) \Rightarrow (\tau_s(A_3) \neq \tau_s(A_1)+5)$
$(A_3 \wedge A_2) \Rightarrow (\tau_s(A_3) \neq \tau_s(A_2)+5)$
$(A_3 \wedge C_3) \Rightarrow (\tau_s(A_3)+5 \neq \tau_s(C_3)+1)$
$(B_2 \wedge B_3) \Rightarrow (\tau_s(B_2) \neq \tau_s(B_3)+4)$
$(B_3 \wedge B_2) \Rightarrow (\tau_s(B_3) \neq \tau_s(B_2)+4)$
$(B_2 \wedge A_1) \Rightarrow (\tau_s(B_2)+4 \neq \tau_s(A_1)+5)$
$(B_2 \wedge A_2) \Rightarrow (\tau_s(B_2)+4 \neq \tau_s(A_2)+5)$
$(B_2 \wedge A_3) \Rightarrow (\tau_s(B_2)+4 \neq \tau_s(A_3)+5)$
$(B_3 \wedge A_1) \Rightarrow (\tau_s(B_3)+4 \neq \tau_s(A_1)+5)$
$(B_3 \wedge A_2) \Rightarrow (\tau_s(B_3)+4 \neq \tau_s(A_2)+5)$
$(B_3 \wedge A_3) \Rightarrow (\tau_s(B_3)+4 \neq \tau_s(A_3)+5)$

- **Bornes inférieure et supérieure :** l'instant initial où les propositions de l'état initial sont vraies est antérieur à tous les instants de début des préconditions des actions du plan. L'instant final où les propositions du but sont vraies est postérieur à tous les instants de fin des effets des actions du plan.

$$5.\quad \left(\tau(n_{Init}) \leq \tau(n_{Goal})\right) \quad \wedge \quad \bigwedge_{n_a \in NoeudsAction} \left(\begin{array}{l} \left(\tau(n_{Init}) \leq \underset{p \in Prec(a)}{Min} \left\{ \tau(p \underset{begin}{\rightarrow} n_a) \right\} \right) \\ \wedge \left(\underset{q \in Eff(a)}{Max} \left\{ \tau(n_a \underset{end}{\rightarrow} q) \right\} \leq \tau(n_{Goal}) \right) \end{array} \right)$$

Exemple :

$(\tau(A_I) \leq \tau(A_G))$
$(\tau(A_I) \leq \tau_s(A_1)) \wedge (\tau(A_I) \leq \tau_s(A_2)) \ldots (\tau(A_I) \leq \tau_s(C_3))$
$(\tau_e(A_1) \leq \tau(A_G)) \wedge (\tau_e(A_2) \leq \tau(A_G)) \ldots (\tau_e(C_3) \leq \tau(A_G))$

Pour notre exemple, le codage obtenu par TLP-GP-2 au troisième niveau du graphe contient 18 variables propositionnelles, 8 variables réelles et 88 clauses. Le solveur trouve une solution à ce niveau. Toutes les étapes du codage de cet exemple sont données en Annexe 6. Le problème est résolu (satisfiable) par MathSat[4] en moins d'une seconde. Sur cet exemple, le plan solution flottant qui est extrait est le même que celui obtenu par recherche arrière (cf. Figure 24). Pour pouvoir appréhender plus facilement les plans-solutions construits par TLP-GP, nous avons implémenté une interface graphique spécifique que nous présentons rapidement dans la section suivante.

La comparaison entre les méthodes d'extraction de solution TLP-GP-1 et TLP-GP-2 nécessite également la mise au point de nouveaux benchmarks qui soient temporellement expressifs. Nous les présenterons dans la section 3.6.2. Comme nous le verrons, les tests expérimentaux montrent que la méthode qui s'avère généralement la plus efficace est la recherche arrière de TLP-GP-1.

3.4. Présentation de l'interface graphique

Comme il est extrêmement difficile d'apréhender des plans temporels flottants sous forme textuelle, nous avons implémenté une interface graphique pour TLP-GP qui permet essentiellement :

- de choisir les fichiers des problèmes / domaines du planificateur ;
- de paramétrer puis de lancer TLP-GP ;
- d'afficher le plan-solution fourni par l'algorithme ;
- de modifier ce plan-solution dans la limite des contraintes fixées par celui ci ;
- d'obtenir des informations sur les différentes actions, préconditions, effets du plan-solution ;
- de simuler l'exécution du plan solution dans le temps afin d'observer l'effet de la réalisation des différentes actions ainsi que l'état des différents fluents ;
- d'enregistrer ce plan-solution modifié.

L'interface graphique de TLP-GP est codée en JAVA et peut donc ainsi être exécutée sur tous les systèmes d'exploitation courants comme Windows, Linux, MacOS. Elle permet d'exécuter

[4] http://mathsat.itc.it/

TLP-GP sur un problème particulier, mais peut également être lancée indépendamment du planificateur et permet de manipuler des plans-solutions flottants à partir de ses fichiers de sortie. Nous présentons, Figure 25, un exemple de plan-solution flottant du problème "cooking01" (cf. section 3.6.2) en cours de simulation dans l'interface graphique de TLP-GP. Une description plus complète de cette interface est fournie en Annexe 7.

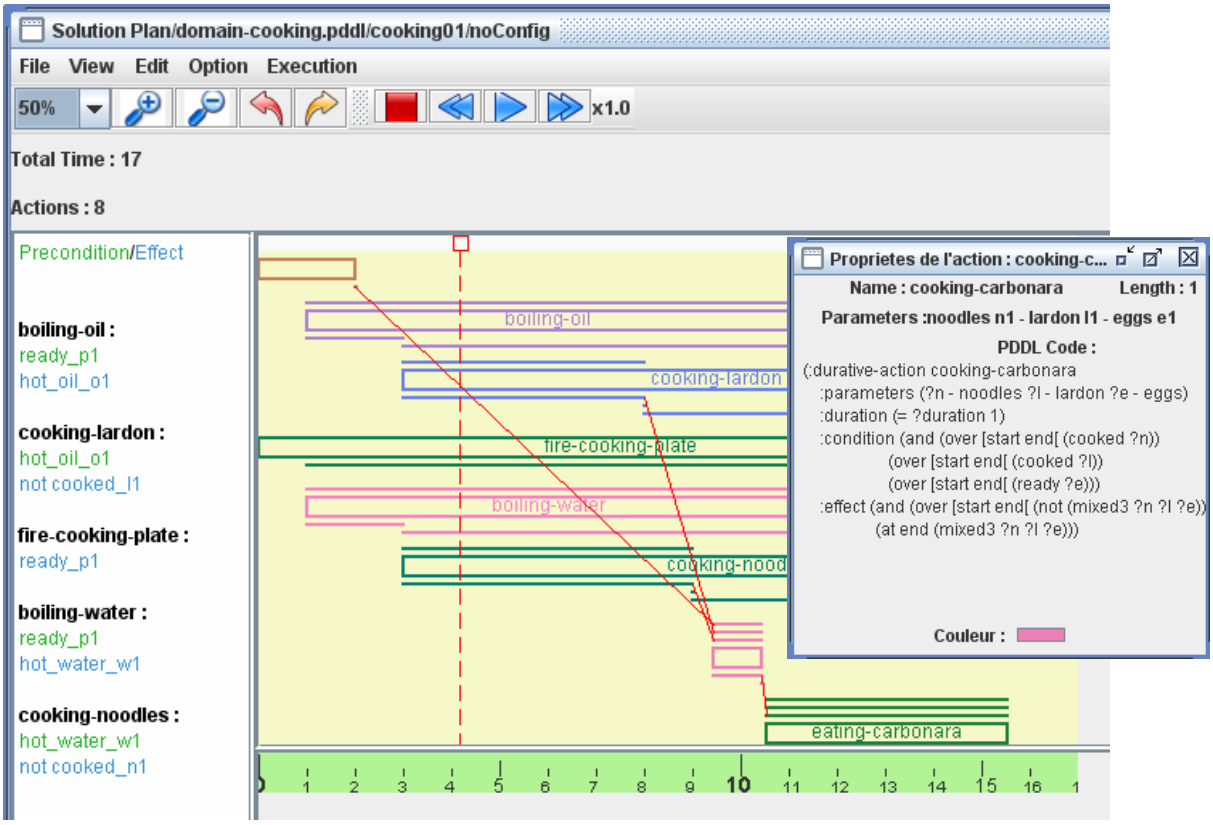


Figure 25 : interface graphique de TLP-GP

Avant de détailler les résultats expérimentaux, nous allons maintenant présenter le travail que nous avons effectué sur l'extension de l'expressivité temporelle du langage PDDL2.1.

3.5. Extension supplémentaire de l'expressivité

Le langage de représentation des problèmes pouvant être traités par TLP-GP est du type $L^{[s+,e+]}_{[s+,e+]}$. Même si ce langage permet une plus grande flexibilité dans la définition de domaines et problèmes temporels, son pouvoir de représention est identique à celui de PDDL2.1, du type $L^{s,o,e}_{s,e}$. Pour pouvoir formaliser des problèmes plus proches de problèmes réels, nous avons implémenté une nouvelle extension qui permet d'augmenter l'expressivité temporelle du langage PDDL2.1 sur deux points essentiels :

- La prise en compte de nouveaux instants caractéristiques des actions, différents de *start* et *end*, ainsi que des ensembles de contraintes associées.
- L'introduction des modalités atomiques *supported* et *forbidden* qui vont nous permettre de formaliser les modalités *at* et *over*, modalités qui existent déjà pour la sémantique des intervalles temporels associés aux effets et préconditions des actions. Elles vont également nous permettre de formaliser de nouvelles modalités comme *somewhere* et *anywhere*.

3.5.1. Définition d'instants caractéristiques pour les actions

Définition 53 (action de PDDL temporel étendu) :

Une action A définie en PDDL temporel étendu comporte trois ensembles supplémentaires par rapport à une action définie en PDDL2.1 :

- **timepoints :** est l'ensemble Ci des instants caractéristiques de l'action. Exemple : *start*, *end*.
- **timealiases :** permet de définir des alias pour les intervalles temporels. Exemple : *all* représente [*start* ; *end*[.
- **timeconstraints :** est l'ensemble des contraintes Cn qui doivent être vérifiées entre instants caractéristiques. Exemple : *end* = *start* + durée(A).

Dans le cas où ces ensembles sont omis dans la définition de l'action, les ensembles par défaut de PDDL2.1 sont alors utilisés :

```
:duration (= ?duration …)
:timepoints (start end)
:timealiases (all [start end[)
:timeconstraints (= (- end start) ?duration)
```

Exemple : l'action *skyjet-fly* permet de reserver une fenêtre de départ depuis une base, et de commencer le vol n'importe quand à l'intérieur de cette fenêtre.

```
(:durative-action skyjet-fly
   :parameters (?s – skyjet ?b - skybase)
   :duration (= ?duration (flying-timewindow ?s))
   :timepoints (start end flypoint)
   :timealiases (all [start end[)
   :timeconstraints ((= (- end start) ?duration)
                     (< start flypoint)
                     (> end flypoint))
   :condition (over all (ready ?s)
   :effect (and (over all (reserved-skybase ?b))
                (over [flypoint (+ flypoint (flying-time ?s))] (flying ?s))
                (at end (not (reserved-skybase ?b)))))
```

3.5.2. Définition de modalités pour les intervalles temporels

a) Modalités atomiques et forme temporelle d'un intervalle

Nous introduisons maintenant deux modalités atomiques qui permettent de définir différentes manières, pour une proposition, d'intervenir sur un intervalle temporel :

- La modalité atomique *supported* indique que f est nécessaire sur un intervalle temporel, elle est équivalente pour un intervalle particulier à la modalité "nécessaire" ($\Box_{[a\,;\,b]}$ f) de la logique modale normale [Kripke, 1959, 1963]. Autrement dit, f doit être supporté par une action sur cet intervalle.
- La modalité atomique *forbidden* indique que l'établissement de f est impossible sur un intervalle temporel. Cette modalité est différente de la négation de la modalité "possible"

de la logique modale normale ($\neg\Diamond_{[a\,;\,b]}$ f), car f peut être vrai sur cet intervalle à condition qu'il soit établi avant l'intervalle.

Définition 54 (modalités atomiques *supported*/*forbidden*) :

Soient [a ; b] un intervalle temporel, et f un littéral. Nous pouvons définir les deux modalités atomiques :

- **`supported [a b] f :`** f est nécessairement **vrai** sur tout l'intervalle [a ; b] ;
- **`forbidden [a b] f :`** f ne peut être **établi** sur l'intervalle [a ; b].

Le résultat de l'application d'une modalité atomique à un littéral sur un intervalle temporel donné sera appelé *atome temporel*.

Dans le cas d'une précondition, $supported_{[a\,;\,b]}$ f signifie qu'une autre action doit produire et protéger f sur [a ; b]. Dans le cas d'un effet, $supported_{[a\,;\,b]}$ f signifie que l'action produit et protège f sur [a ; b]. Autrement dit, la modalité *supported* correspond à l'établissement de f "en tout point" de l'intervalle. Dans les deux cas, $forbidden_{[a\,;\,b]}$ f signifie qu'aucune autre action, distincte de l'action dans laquelle l'atome temporel est utilisé, ne peut produire f sur [a ; b].

Définition 55 (forme temporelle) :

A tout intervalle temporel sera associé un ensemble d'atomes temporels. Cet ensemble sera appelé la *forme temporelle* (ou la particularité temporelle) de l'intervalle. Par extension, la *forme temporelle* (ou la particularité temporelle) d'un ensemble d'intervalles temporels sera l'union des formes temporelles de tous ses éléments.

Définition 56 (effet-cohérence)

La forme temporelle F d'un ensemble d'intervalles sera dite *effet-cohérente* si pour tout couple d'atomes temporels ($mod^1_{[a\,;\,b]}$ f, $mod^2_{[c\,;\,d]}$ $\neg$f) $\in$ F^2, on a ($mod^1 \equiv mod^2 \equiv$ *supported*) $\Rightarrow$ ([a ; b] $\cap$ [c ; d] = $\varnothing$). Concrètement, une forme temporelle qui n'est pas *effet-cohérente* ne pourra pas intervenir dans la définition d'un effet d'une action applicable car l'action ne pourra jamais produire f et sa négation sur un même intervalle.

Définition 57 (précondition-cohérence)

La forme temporelle F d'un ensemble d'intervalles sera dite *précondition-cohérente* si F est *effet-cohérente* et pour tout couple d'atomes temporels ($mod^1_{[a\,;\,b]}$ f, $mod^2_{[c\,;\,d]}$ f) $\in$ F^2, on a ($mod^1 \equiv$ *supported* et $mod^2 \equiv$ *forbidden*) $\Rightarrow$ ([a ; b] $\cap$ [c ; d] = $\varnothing$). Concrètement, une forme temporelle qui n'est pas *précondition-cohérente* ne pourra pas intervenir dans la définition d'une précondition d'une action applicable, d'une part parce que l'action ne pourra pas requérir à la fois f et sa négation sur un même intervalle (effet-cohérence), et d'autre part parce que l'action ne pourra pas requérir la protection de f sur un intervalle et interdire son établissement (son maintien) sur ce même intervalle (précondition-cohérence).

Etant donné deux formes temporelles données, nous pouvons définir P comme étant la partition de l'union des intervalles temporels correspondants. Pour que ces deux ensembles d'intervalles temporels puissent se recouvrir, il faut vérifier, pour chaque intervalle de P, la compatibilité des atomes temporels qui leur sont associés. Nous donnons ici le diagramme qui

permet de déterminer les atomes temporels autorisant le recouvrement des intervalles temporels.

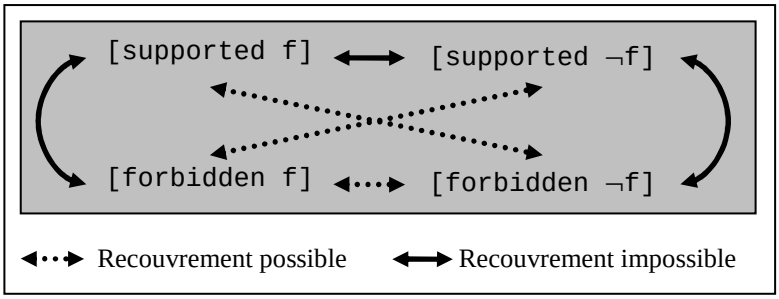


Figure 26 : diagramme d'autorisation de recouvrement des intervalles temporels

Nous allons maintenant redéfinir les modalités courantes utilisées implicitement dans le langage PDDL2.1 en utilisant les modalités atomiques que nous venons de définir.

b) Modalités courantes de PDDL2.1

Nécessité d'une précondition ou apparition d'un effet à un instant donné : at a p

La modalité "at" permet de définir un instant auquel une précondition doit être vérifiée ou un effet doit apparaître :

- p est vraie à l'instant a.
- ¬p ne peut pas être vraie à l'instant a.

Modalité "at" : **at a f**
`supported [a a] f`
	`(:time-modality at ?a ?f`
	`:timevariables ()`
	`:constraints ()`
	`:supported ([?a ?a] ?f)`
	`:forbidden ())`

Maintien d'une précondition ou d'un effet sur une durée : over [a b] p

Cette modalité existait déjà en partie pour les préconditions dans PDDL2.1 (over all) et [Cushing et al., 2007.b] l'ont étendue aux effets des actions. Dans TLP-GP, nous l'étendons à des intervalles quelconques ; c'est cette modalité que nous avons implicitement utilisée dans les trois sections précédentes.

- p est vraie sur l'intervalle [a, b] (maintien d'un effet).
- ¬p ne peut pas être vraie sur l'intervalle [a, b].

Modalité “over” : **over [a b] f**
`supported [a b] f`
	`(:time-modality over [?a ?b] ?f`
	`:timevariables ()`
	`:constraints (<= ?a ?b)`
	`:supported ([?a ?b] ?f)`
	`:forbidden ())`

c) Modalités "somewhere" et "anywhere"

Nous définissons ici, en utilisant les modalités atomiques *supported* et *forbidden*, deux nouvelles modalités : "somewhere" et "anywhere".

Incertitude sur l'instant de production de l'effet (resp. sur l'instant de nécessitation d'une précondition) : somewhere [a b] p

Cette modalité permet d'exprimer que l'on doit protéger tout l’intervalle pour garantir que l’effet puisse s’y produire (resp. pour garantir que la précondition puisse y être satisfaite) : il existe une marge de manoeuvre, mais la valeur finale est incontrôlable.

- p est vraie à l'instant b (incertitude sur l'instant d'apparition de l'effet, certitude sur sa production).
- ¬p ne peut pas être établie sur l'intervalle [a, b] (protection de l'intervalle).

Modalité “somewhere” : **somewhere [a b] f**
`supported [b b] f`
`forbidden [a b] ¬f`
	`(:time-modality somewhere [?a ?b] ?f`
	`:timevariables ()`
	`:constraints (<= ?a ?b)`
	`:supported ([?b ?b] ?f)`
	`:forbidden ([?a ?b] (not ?f)))`

Choix possible de l'instant de production de l'effet (resp. de l'instant de nécessitation d'une précondition) : anywhere [a b] p

Cette modalité permet d'exprimer que l’effet (resp. la précondition) doit impérativement se produire (resp. être produite) dans l'intervalle, mais n’importe où : il existe une marge de manoeuvre, mais la valeur finale est contrôlable. Le planificateur peut réduire l’intervalle, sans toutefois aboutir à l’intervalle vide pour conserver la satisfiabilité. On peut en plus imposer une durée de validité minimale à l’effet, durée pendant laquelle l'effet doit être maintenu dans l'intervalle (syntaxe : **minimal-duration d anywhere [a b] p**).

- p est vraie à un instant c de l'intervalle [a, b]. On peut, en utilisant *minimal-duration d*, le contraindre à rester vrai à l'intérieur de [a, b] durant d.
- ¬p peut être vraie sur une partie de l'intervalle [a, b] mais pas partout, en particulier pas de l'instant c (inclus) jusqu'à c+d (si *minimal-duration d* est utilisé).

Modalité “anywhere” : **anywhere [a b] f**
`supported [i j] f`
	`(:time-modality anywhere [?a ?b] ?f`
	`:timevariables (?i ?j)`
	`:constraints ((<= ?a ?i)`
	`(<= ?j ?b)`
	`(<= ?i ?j))`
	`:supported ([?i ?j] ?f)`
	`:forbidden ())`

Modalité “anywhere” avec durée minimale : **minimal-duration d anywhere [a b] f**
`supported [i j] f`
	`(:time-modality minimal-duration ?d`
	`anywhere [?a ?b] ?f`
	`:timevariables (?i ?j)`
	`:constraints ((<= ?a ?i)`
	`(<= ?j ?b)`
	`(<= ?i ?d)`
	`(<= ?d (- ?j ?i)))`
	`:supported ([?i ?j] ?f)`
	`:forbidden ())`

Remarque : la modalité *anywhere* est identique à *over* ; l’utilisation de cette modalité dans la définition d'une action peut être vue comme l’ajout d’un nouvel instant caractéristique i borné par l'intervalle [a ; b], et de nouvelles contraintes associées.

d) Modalités atomiques incompatibles entre elles dans la définition de la forme temporelle d'un intervalle

Il est possible d’utiliser conjointement des modalités atomiques *supported* et *forbidden* incompatibles entre elles dans la forme temporelle d'un intervalle.

Exemple, effets "over all" de [Cushing et al., 2007.b] :

Dans la forme originale, un tel effet représente la transition d'un fluent f d'une valeur donnée à une autre sur toute la durée de l'action. C'est-à-dire que f est affecté par l'action sur cette période et ne peut donc pas intervenir simultanément dans une autre action. Par exemple, la transition du fluent f de la valeur faux à la valeur vrai est codée :

```
:effect (and (over all (-> f false true) …)
```

Cette représentation présente l'inconvénient d'ajouter implicitement une précondition dans la définition d'un effet puisque, pour passer de la valeur faux à la valeur vrai, le fluent doit nécessairement être faux au début de l'action. Nous adoptons une définition plus large d'une transition sur un intervalle [a, b] : f est indéfini sur [a, b[et f prend une valeur fixée à l'instant b. Cette transition peut être représentée par une nouvelle modalité que nous appelerons "transition over" (notée `->over [a b] f`). Notre exemple de transition peut alors être codé :

```
:condition (and (at start (not f)) …)
:effect (and (->over all f) …)
```

Les avantages de cette représentation sont d'une part qu'il n'est plus permis d'inclure une précondition dans la définition d'un effet et d'autre part qu'il n'est plus nécessaire d'imposer la connaissance de la valeur d'un fluent pour qu'il puisse se trouver en transition vers une autre valeur. Par exemple, si une action consiste à planter un clou dans une planche en donnant plusieurs coups de marteau, il n'est pas nécessaire de savoir si le clou est complètement enfoncé ou pas au début de l'action, mais il le sera certainement à la fin.

Modalité "transition over" : **→over [a b] f**
`supported [b b] f`
`forbidden [a b] f`
`forbidden ]a b] ¬f`
	`(:time-modality (->over [?a ?b] ?f`
	`  :timevariables ()`
	`  :constraints (<= ?a ?b)`
	`  :supported ([?b ?b] ?f)`
	`  :forbidden (([?a ?b] ?f)`
	`  (]?a ?b] (not ?f))))`

La modalité *transition over* code le fait que f est produit à l'instant b et aucune autre action ne peut produire f ou ¬f sur l'intervalle [a ; b]. La forme temporelle de son ensemble d'intervalles associés est effet-cohérente, mais n'est pas précondition-cohérente puisqu'elle contient $supported_{[b\,;\,b]}$ f et $forbidden_{[a\,;\,b]}$ f et que $[b\,;\,b] \cap [a\,;\,b] = \{b\} \neq \emptyset$. Cette modalité ne peut donc être utilisée que pour définir un effet. Si elle est utilisée pour la définition d'une précondition, cette dernière ne pourra jamais être vérifiée.

3.5.3. Exemples d'actions utilisant les instants caractéristiques ou les modalités

Outre l'augmentation importante de l'expressivité du langage, l'introduction conjointe des instants caractéristiques avec leurs contraintes associées et des modalités temporelles permet une grande souplesse dans la définition des actions. Par exemple, si nous souhaitons capturer l'information temporelle de la modalité *anywhere* (anywhere [start end] (flying ?s)), nous pouvons utiliser un nouvel instant caractéristique et la modalité *over*.

```
(:durative-action skyjet-fly
   :parameters (?s – skyjet)
   :duration (= ?duration (flying-timewindow ?s))
   :timepoints (start end flypoint)
   :timealiases (all [start end[)
   :timeconstraints ((= (- end start) ?duration)
                     (< start flypoint)
                     (> end flypoint))
   :condition (over all (ready ?s)
   :effect (at flypoint (flying ?s)))
```

Il est également possible d'utiliser des instants caractéristiques et la modalité *over* pour coder la modalité *anywhere* avec durée minimale (minimal duration 3 anywhere [start end] (flying ?s)).

```
(:durative-action skyjet-fly
   :parameters (?s – skyjet)
   :duration (= ?duration (flying-timewindow ?s))
   :timepoints (start end flypoint-s flypoint-e)
   :timealiases (all [start end[)
   :timeconstraints ((= (- end start) ?duration)
                     (< start flypoint-s)
                     (> end flypoint-e)
                     (>= (- flypoint-e flypoint-s) 3)
   :condition (over all (ready ?s)
   :effect (over [flypoint-s flypoint-e] (flying ?s)))
```

Après avoir proposé de nouvelles extensions de l'expressivité de PDDL2.1, nous allons voir que les différents benchmarks des compétitions IPC sont tous temporellement simples et nous allons proposer un ensemble de nouveau benchmarks temporellement expressifs intégrant progressivement les intervalles temporels de [Smith, 2003], puis les modalités temporelles que nous avons définies.

3.6. Benchmarks temporellement expressifs

3.6.1. Les benchmarks des IPC

La première compétition IPC s'est déroulée en 1998 avec la première version du langage PDDL. Les aspects temporels n'ont été intégrés au langage PDDL2.1 et aux benchmarks qu'en 2002 avec la tenue d'IPC'2002.

Tous les benchmarks[5] temporels des compétitions IPC (y compris IPC'2008) sont des extensions des benchmarks classiques, auxquels on a simplement ajouté une durée aux actions, et où l'on a précisé les instants caractéristiques d'apparition des effets ou de nécessitation des préconditions. Comme nous l'avons déjà expliqué, [Cushing et al, 2007.b] ont démontré que tous ces domaines étaient temporellement simples. Ils proposent des modifications permettant de le vérifier facilement[6]. Une précondition ou un effet d'une action est dit "sûr" s'il ne peut exister de plan exécutable dans lequel cette précondition ou cet effet provoque obligatoirement un conflit entre ordre causal et ordre temporel. [Cushing et al, 2007.b] montrent que si toutes les préconditions et tous les effets des actions d'un domaine sont "sûrs", alors le domaine est temporellement simple. Ils démontrent ensuite que les effets *over all*, qui peuvent être soit des transitions (cf. section 3.5.2.d), soit des affectation de valeurs, sont "sûrs". Il suffit donc de modifier les domaines des compétitions IPC pour obtenir des domaines équivalents ne contenant que des préconditions et effets dont on a montré qu'ils étaient "sûrs". L'action *Drive* du domaine *Depots* est ainsi modifiée en utilisant des fluents multivalués (les modifications apparaissent en gras) :

```
(:durative-action Drive
  :parameters (?x - truck ?y - place ?z - place)
  :duration (= ?duration (/ (distance ?y ?z) (speed ?x)))
  :condition (and     ;;(at start (at ?x ?y)))            ;; Fluent Multivalué
  :effect (and        ;;(at start (not (at ?x ?y)))       ;; Fluent Multivalué
                      ;;(at end (at ?x ?z))               ;; Fluent Multivalué
                      (over all (-> (location ?x) ?y ?z))  ;; Fluent Multivalué
))
```

Cette modification pourrait être réalisée sans utilisation de fluents multivalués et de manière plus simple en utilisant notre modalité "transition over" sur toute la durée de l'action sans inclure implicitement la précondition "le camion ?x se situe en ?y au début de l'action" dans les effets :

```
(:durative-action Drive
  :parameters (?x - truck ?y - place ?z - place)
  :duration (= ?duration (/ (distance ?y ?z) (speed ?x)))
  :condition (and (at start (at ?x ?y)))
  :effect (and        ;;(at start (not (at ?x ?y)))
                      ;;(at end (at ?x ?z))
                      (->over all (at ?x ?z)))
))
```

[5] 2002 : Depots, DriverLog, ZenoTravel, Satellite, Rovers ; 2004 : Satellite, Airport, UMTS ; 2006 : Openstacks, Pathways, Pipesworld, Storage, TPP, Trucks ; 2008 : CrewPlanning, Elevators, ModelTrain, ParcPrinter, PegSol, Sokoban, Transport, WoodWorking.

[6] http://www.public.asu.edu/~ktalamad/temporal/

3.6.2. Elaboration de benchmarks temporellement expressifs

Comme les benchmarks des competitions IPC étaient inappropriés, puisque temporellement simples, nous avons mis au point plusieurs nouveaux benchmarks temporellement expressifs. Les premiers domaines que nous présentons utilisent des actions duratives de PDDL2.1. Nous avons ensuite progressivement développé des benchmarks plus expressifs et plus complexes qui intègrent des intervalles temporels, puis des modalités temporelles. Les caractéristiques des domaines et des problèmes sont aussi de natures différentes et complémentaires. Tous ces benchmarks peuvent être téléchargés à l'adresse donnée ci-dessous[7].

a) Problèmes temporellement expressifs en PDDL2.1

Les trois premiers domaines étendent l'exemple de [Cushing et al., 2007.a, Figure 3] de trois manières différentes :

- les problèmes "tempo-depth-n" étendent ce problème sur un plus grand nombre de niveaux. Les plans-solutions nécessitent n "étapes" composées de triplets d'actions concurrentes.
- les problèmes "tempo-width-m" nécessitent m triplets d'actions concurrentes sur uniquement trois niveaux du graphe pour obtenir le but.
- les problèmes "tempo-matrix-n×m" combinent les difficultés des deux précédents.

Domaine tms-k-t-light : ce domaine (dérivé de temporal machine shop, [Cushing et al., 2007.a]) est inspiré par une application du monde réel. Chaque problème tms-k-t-p, concerne l'utilisation de k fours (kilns), chacun ayant une durée de chauffe différente, pour cuire p morceaux de céramique (*bake-ceramic*) de t types différents. Chacun de ces types de céramique nécessite un temps de cuisson différent. Ces céramiques peuvent ensuite être assemblée pour produire différentes structures (*make-structure*). Les structures résultantes peuvent alors être recuites pour obtenir des structures plus grandes (*bake-structure*). Nous avons développé cette version "light" pour les planificateurs temporellement expressifs qui ne prennent pas en compte les actions duratives plus riches autorisant l'utilisation des intervalles temporels.

b) Problèmes utilisant des intervalles temporels

Ces problèmes utilisent les actions duratives plus riches telles qu'elles sont définies dans [Smith, 2003], pour lesquelles les préconditions ou les effets peuvent se réaliser sur des intervalles quelconques relatifs au début ou à la fin des actions. Nous avons étendu le domaine tms-2-3-light en utilisant ces intervalles temporels (domaine tms-2-3) et développé un nouveau domaine relatif à la cuisine.

Domaine Cooking : ce domaine permet de planifier la préparation d'un repas, ainsi que sa consommation en respectant des contraintes de maintien de température. Les problèmes cooking-carbonara-n que nous avons utilisés pour ces tests permettent de planifier la préparation de n plats de pâtes à la carbonara. Ici encore, la concurrence des actions est obligatoire pour obtenir le but. En effet, il est nécessaire que les plaques électriques fonctionnent pour que l'eau et l'huile soient chaudes afin de cuire les pâtes et les lardons. Il est également nécessaire d'effectuer ces cuissons en parallèle pour pouvoir servir un plat dont

[7] http://tlpgp.free.fr/

tous les éléments soient chauds sur toute la durée de consommation. Nous donnons ici, en utilisant les extensions de [Smith, 2003], le codage de l'action *boiling-water* :

```
(:durative-action boiling-water
    :parameters (?p - cooking-plate  ?w - water-pan)
    :duration (= ?duration 10)
    :condition (and (over [start end[ (ready ?p)))
    :effect (and (over [start (+ start 2)[ (not (hot-water ?w)))
                 (over [(+ start 2) end] (hot-water ?w))))
```

Avec les intervalles de [Smith, 2003], le domaine comprend 8 opérateurs. Sa traduction en PDDL2.1 en décomposant certaines actions en actions plus simples, que nous apellerons **cooking-pddl2.1**, en comprend 18. Nous donnons ici la décomposition en PDDL2.1 de l'action *boiling-water* :

```
(:durative-action boiling-water
    :parameters (?p - cooking-plate  ?w - water-pan)
    :duration (= ?duration 10)
    :condition (and (over all (ready ?p))
                    (at end (boiling-water-finished ?p ?w)))
    :effect (and (at start (boiling-water-ing ?p ?w))
                 (at start (boiling-water-enabled-part1 ?p ?w))
                 (at start (not (hot-water ?w)))))

(:durative-action boiling-water-part1
    :parameters (?p - cooking-plate ?w - water-pan)
    :duration (= ?duration 2)
    :condition (and (over all (boiling-water-ing ?p ?w))
                    (over all (boiling-water-enabled-part1 ?p ?w)))
    :effect (and (at end (not (boiling-water-enabled-part1 ?p ?w)))
                 (at end (boiling-water-enabled-part2 ?p ?w))))

(:durative-action boiling-water-part2
    :parameters (?p - cooking-plate ?w - water-pan)
    :duration (= ?duration 8)
    :condition (and (over all (boiling-water-ing ?p ?w))
                    (over all (boiling-water-enabled-part2 ?p ?w)))
    :effect (and (at start (hot-water ?w))
                 (at end (not (boiling-water-enabled-part2 ?p ?w)))
                 (at end (boiling-water-finished ?p ?w))))
```

c) Problèmes utilisant les modalités sur les intervalles temporels

Domaine temporal-machine-shop-k-t : ce domaine étend le domaine "tms-k-t-light". Pour représenter des domaines plus proches de la réalité nous avons défini un langage plus expressif (cf. section 3.5). Nous donnons ici la description complète du problème tms-2-3-03 issu du domaine temporal-machine-shop-2-3. Si l'on utilise toute l'expressivité de TLP-GP pour décrire ce domaine, on peut introduire une incertitude sur l'instant où un four commencera à être suffisamment chaud après le démarrage d'une action *fire-kiln* (somewhere [start (+ start 1)] (ready ?k)), puis pour garantir le maintien de cet état sur une durée fixée (over [(+ start 1) end[(ready ?k)). Par ailleurs, il est possible que la cuisson d'une céramique prenne moins de temps que prévu, et nous pouvons également représenter l'incertitude sur l'instant exact où cette cuisson va se terminer (somewhere [(- end 5) end] (baked ?p)). L'utilisation d'un état initial daté permet de définir une plage durant laquelle l'énergie nécessaire au fonctionnement des fours sera disponible (over [0 30[(energy)) et après laquelle elle ne le sera plus ((at 30 (not (energy)))). Nous pouvons aussi représenter le fait que les poteries cuites seront disponibles (pour livraison par exemple) sur certaines plages temporelles. La structure p1/p2 devra être prête sur l'ensemble d'un intervalle temporel (over [30 40] (baked-structure p1 p2)), alors que la céramique p3 devra être prête sur un intervalle

de durée 5, mais que le planificateur pourra placer n'importe quand dans un intervalle plus grand fixé (minimal-duration 5 anywhere [30 40] (baked p3)).

```
(define (domain domain-tms-2-3)

(:requirements :strips :typing :durative-
actions)
(:types
 piecetype1 piecetype2 piecetype3 – piece
 kiln8 kiln20 – kiln)

(:predicates
 (energy )
 (ready ?k – kiln)
 (baking ?p – piece)
 (treated ?p – piece)
 (baked ?p – piece)
 (structured ?p1 - piece ?p2 – piece)
 (baked-structure ?p1 - piece ?p2 - piece))

(:durative-action fire-kiln1
 :parameters (?k – kiln8)
 :duration (= ?duration 8)
 :condition (over all (energy ))
 :effect (and
  (somewhere [start (+ start 1)]
             (ready ?k))
  (over [(+ start 1) end[ (ready ?k))
  (at end (not (ready ?k)))))

(:durative-action fire-kiln2
 :parameters (?k – kiln20)
 :duration (= ?duration 20)
 :condition (over all (energy ))
 :effect (and
  (somewhere [start (+ start 2)]
             (ready ?k))
  (over [(+ start 2) end[ (ready ?k))
  (at end (not (ready ?k)))))

(:durative-action bake-ceramic1
 :parameters (?p - piecetype1 ?k – kiln)
 :duration (= ?duration 15)
 :condition (over all (ready ?k))
 :effect (and
  (over [start end[ (not (baked ?p)))
  (over [start end[ (baking ?p))
  (at end (not (baking ?p)))
  (somewhere [(- end 5) end] (baked ?p))))

(:durative-action bake-ceramic2
 :parameters (?p - piecetype2 ?k – kiln)
 :duration (= ?duration 10)
 :condition (over all (ready ?k))
 :effect (and
  (over [start end[ (not (baked ?p)))
  (over [start end[ (baking ?p))
  (at end (not (baking ?p)))
  (somewhere [(- end 3) end] (baked ?p))))

(:durative-action bake-ceramic3
 :parameters (?p - piecetype3 ?k – kiln)
 :duration (= ?duration 5)
 :condition (over all (ready ?k))
 :effect (and
  (over [start end[ (not (baked ?p)))
  (over [start end[ (baking ?p))
  (at end (not (baking ?p)))
  (somewhere [(- end 1) end] (baked ?p))))

(:durative-action treat-ceramic1
 :parameters (?p – piece)
 :duration (= ?duration 3)
 :condition (over all (baking ?p))
 :effect (at end (treated ?p)))

(:durative-action treat-ceramic2
 :parameters (?p – piecetype2)
 :duration (= ?duration 2)
 :condition (over all (baking ?p))
 :effect (at end (treated ?p)))

(:durative-action treat-ceramic3
 :parameters (?p – piecetype3)
 :duration (= ?duration 1)
 :condition (over all (baking ?p))
 :effect (at end (treated ?p)))

(:durative-action make-structure
 :parameters (?p1 - piece ?p2 – piece)
 :duration (= ?duration 1)
 :condition (and
  (over all (baked ?p1))
  (over all (treated ?p1))
  (over all (baked ?p2))
  (over all (treated ?p2)))
 :effect (at end (structured ?p1 ?p2)))

(:durative-action bake-structure
 :parameters
  (?p1 - piece ?p2 - piece ?k – kiln)
 :duration (= ?duration 3)
 :condition (and
  (over all (ready ?k))
  (over all (structured ?p1 ?p2)))
 :effect (and
  (over [start end[
  (not (baked-structure ?p1 ?p2)))
  (at end (baked-structure ?p1 ?p2)))))


(define (problem tms-2-3-03)
 (:domain domain-tms-2-3)
 (:objects
  k1 – kiln8
  k2 – kiln20
  p1 – piecetype1
  p2 – piecetype2
  p3 – piecetype3)
 (:init
  (over [0 30[ (energy ))
  (at 30 (not (energy ))))
 (:goal (and
  (over [30 40] (baked-structure p1 p2))
  (minimal-duration 5 anywhere [30 40]
  (baked p3)))))
```

Nous allons maintenant présenter les résultats des tests expérimentaux que nous avons réalisés en utilisant ces benchmarks pour comparer TLP-GP avec d'autres planificateurs temporels.

3.7. Résultats expérimentaux

3.7.1. Introduction

TLP-GP est implémenté en OCaml 3.09.2. Les exemples ont été testés sur machine Pentium M cadencée à 1,6 GHz avec 512 Mo DDRAM sous Linux Kubuntu (noyau 2.6.17-11). TLP-GP utilise le solveur SMT (Sat Modulo Theory) MathSat[8] 3.4 qui montre de bonnes performances pour la résolution de DTP (cf. compétition SMT-COMP'06[9]). Pour extraire une solution, TLP-GP donne la priorité aux sous-buts qui apparaissent dans les niveaux les plus élevés du graphe et, pour les établir, utilise d'abord les actions qui apparaissent dans les niveaux les moins élevés du graphe.

Nous avons d'abord (cf. section 3.7.2) comparé TLP-GP avec les planificateurs temporels de la compétition IPC'2008, à laquelle nous avons participé. Si ces planificateurs offrent de bien meilleures performances que TLP-GP sur les problèmes temporellement simples de la compétition, ils sont incapables de résoudre des problèmes temporellement expressifs.

Nous avons ensuite (cf. section 3.7.3) comparé TLP-GP à trois planificateurs de référence capables de traiter des problèmes temporellements expressifs (cf. section 2.3) :

- le planificateur LPGP, extension temporelle de GRAPHPLAN,
- le planificateur dans les espaces de plans partiels VHPOP2.2,
- le planificateur dans les espaces d'états étendus CRIKEY3.

Comme LPGP, VHPOP et CRIKEY ont une expressivité temporelle inférieure à celle de TLP-GP, nous avons utilisé une version simplifiée, mais toujours temporellement expressive, de nos benchmarks. Pour tous les tests, nous avons limité le temps de recherche d'une solution à 1h.

3.7.2. Comparaison avec les planificateurs de la compétition IPC'2008

Sur les benchmarks temporellement simples de la compétition, les performances de TLP-GP sont bien en dessous de celles des autres planificateurs (Figure 27).

[8] http://mathsat.itc.it/
[9] http://www.csl.sri.com/users/demoura/smt-comp/

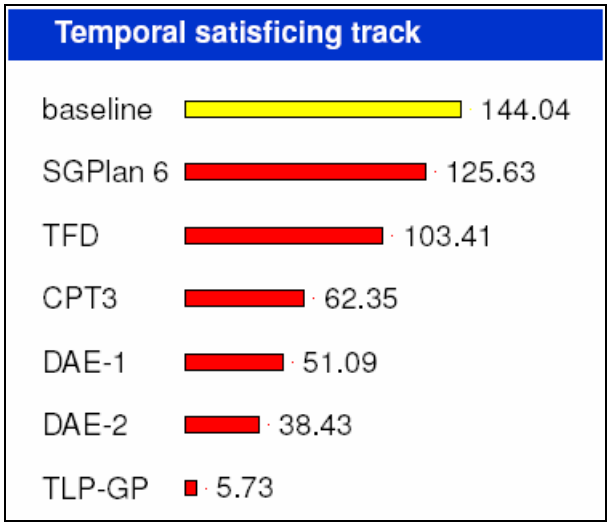


Figure 27 : Résultats de la compétition IPC'2008 (temporal satisficing track)

Cependant, aucun des planificateurs de la compétition, à l'exception de TLP-GP, ne peut résoudre de problème temporellement expressifs. Nous le montrons ici en les exécutants sur le petit problème temporellement expressif de [Cushing et al., 2007a, figure 3].

```
(define (domain domain-tempo)
  (:requirements :strips :durative-actions)
  (:predicates (i ) (a ) (b ) (c )
               (d ) (e ))

  (:durative-action A
      :duration (= ?duration 5)
      :condition (at start (i ))
      :effect (and (at start (a ))
                   (at end (not (a )))
                   (at end (b ))
                   (at end (not (d )))))
  (:durative-action B
      :duration (= ?duration 4)
      :condition (at start (a ))
      :effect (and (at start (c ))
                   (at end (not (c )))
                   (at end (d ))))
  (:durative-action C
      :duration (= ?duration 1)
      :condition (at start (c ))
      :effect (and (at end (not (b )))
                   (at end (e ))))
)


(define (problem tempo)
  (:domain domain-tempo)
  (:init (i ))
  (:goal (and (b ) (d ) (e ))))
```

Le gagnant de la compétition IPC'2008, SGPLAN6 n'est pas un planificateur sain. En effet, sur notre exemple, il renvoie un plan qui n'est pas un plan-solution. Le plan n'est tout simplement pas exécutable !

```
; Time 0.03
; ParsingTime 0.03
; NrActions 5
; MakeSpan
; MetricValue 9.005
; PlanningTechnique Modified-FF(enforced hill-climbing search) as the subplanner

0.001: (A) [5.0000]
0.002: (B) [4.0000]
4.003: (C) [1.0000]
4.004: (A) [5.0000]
5.005: (B) [4.0000]
```

Dans le plan fourni par SGPLAN6 (Figure 28), l'action C ne peut être démarrée à l'instant 4.003 car l'action B détruit sa précondition c à l'instant 4.002 et qu'aucune autre action ne la rétablit dans l'intervalle [4.002 ; 4.003]. L'action B ne peut être démarrée à l'instant 5.005 car

l'action A détruit sa précondition a à l'instant 5.001 et qu'aucune autre action ne la rétablit dans l'intervalle [5.001 ; 5.005]. Ce plan n'est donc pas un plan-solution.

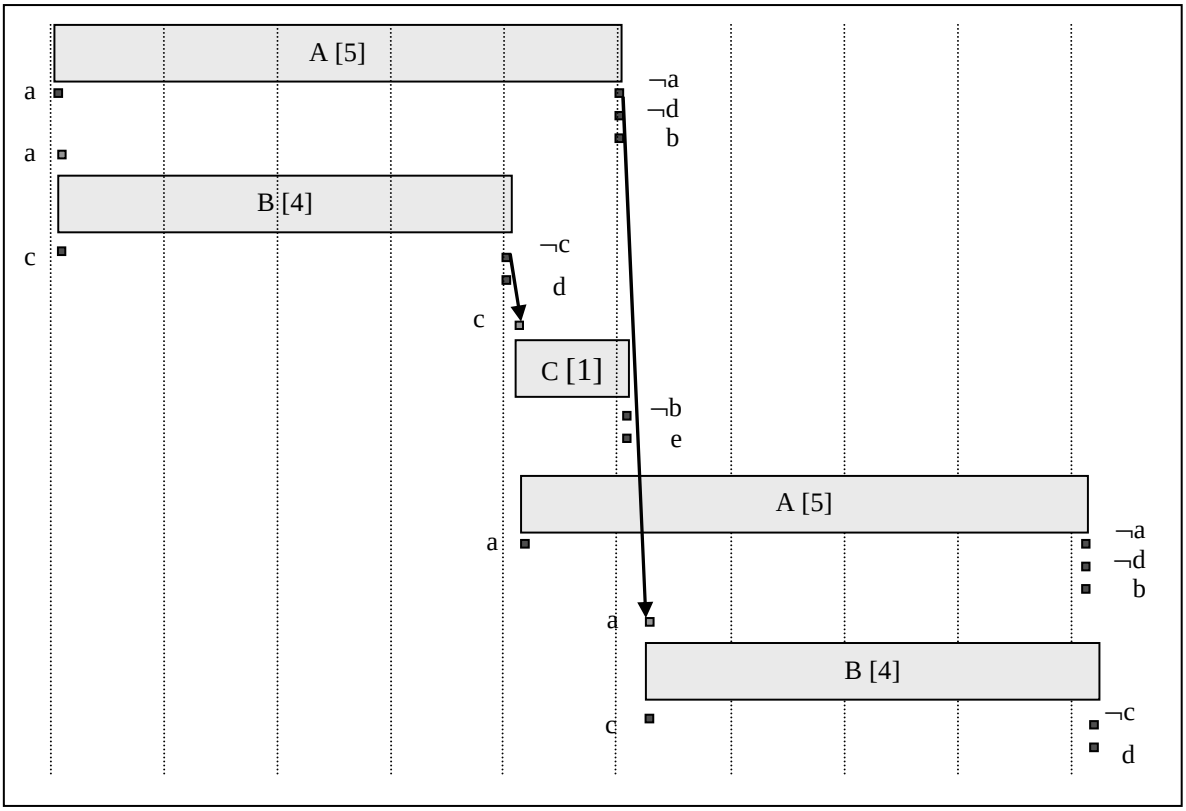


Figure 28 : solution incorrecte fournie par SGPLAN6

Le second planificateur de la compétition IPC'2008, Temporal-Fast-Downward, ne trouve aucune solution :

```
Conducting best first search.
Initializing cyclic causal graph heuristic...done.
Completely explored state space -- no solution!
```

Le planificateur CPT3, le seul de la catégorie "planificateurs temporels optimaux", restreint son langage de représentation aux sous-langages L_e^o de PDDL2.1. Il n'accepte donc que des préconditions *over all* et des effets *at end*, ce qui ne lui permet pas d'interpréter l'exemple. DAE-1 et DAE-2 faisant eux-mêmes appel à CPT3, il ne peuvent pas non plus résoudre ce problème.

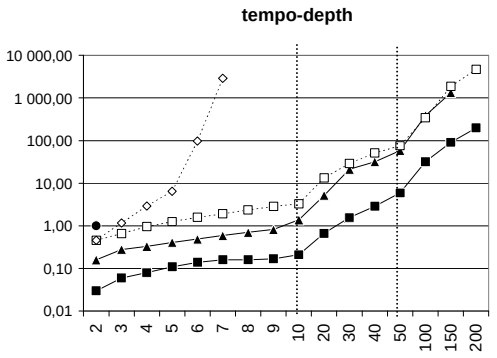
Notre planificateur, TLP-GP, est le seul capable de fournir un plan-solution :

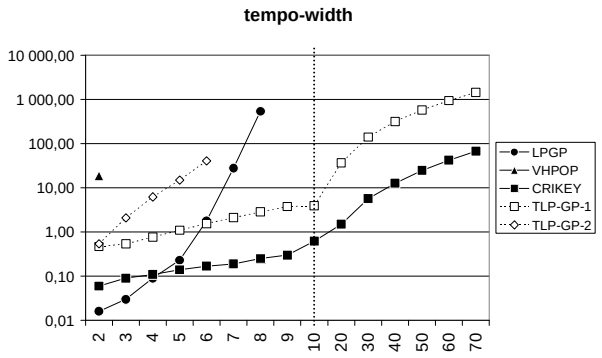
```
Searching floating plan using TLP-GP algorithm at level 3 ...
Floating plan found at level 3.
Searching static plan ...

0.000000: (A) [5.000000]
1.015625: (B) [4.000000]
1.015625: (C) [1.000000]
```

Nous allons maintenant comparer les performances de TLP-GP avec celles des trois planificateurs temporellement expressifs de référence CRIKEY, VHPOP et LPGP.

3.7.3. Comparaison avec les planificateurs temporellement expressifs

a) Problèmes temporels codés en PDDL2.1



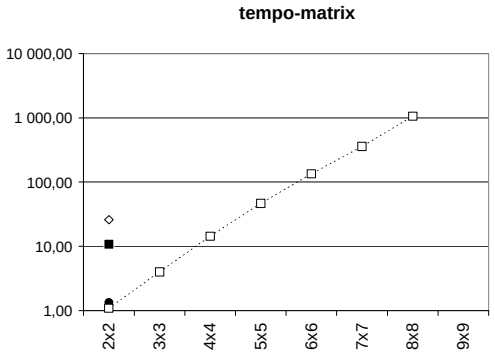


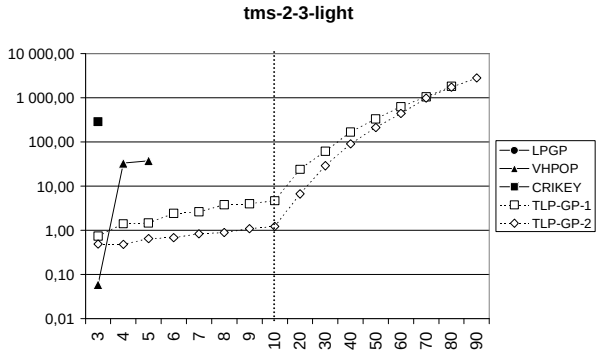
Problème		TLP-GP-1			TLP-GP-2	LPGP			VHPOP2.2			CRIKEY3		
		Act.	Make span	Temps CPU (s)	Temps CPU (s)	Act.	Make span	Temps CPU (s)	Act.	Make span	Temps CPU (s)	Act.	Make span	Temps CPU (s)
tempo-depth	2	6	10^+	0,46	0,46	18	10	1,00	6	10,04	0,16	12	30,01	0,03
	3	9	15^+	0,66	1,17	-	-	-	9	15,06	0,28	18	45,02	0,06
	4	12	20^+	0,96	2,91	-	-	-	12	20,08	0,33	24	60,02	0,08
	5	15	25^+	1,26	6,44	-	-	-	15	25,10	0,41	30	75,03	0,11
	6	18	30^+	1,58	97,53	-	-	-	18	30,12	0,49	36	90,04	0,14
	7	21	35^+	1,92	2 882,42	-	-	-	21	35,14	0,59	42	105,04	0,16
	8	24	40^+	2,36	-	-	-	-	24	40,16	0,70	48	120,05	0,16
	9	27	45^+	2,85	-	-	-	-	27	45,18	0,82	54	135,05	0,17
	10	30	50^+	3,28	-	-	-	-	30	50,20	1,39	60	150,06	0,21
	20	60	100^+	13,36	-	-	-	-	60	100,40	5,20	120	300,12	0,67
	30	90	150^+	28,89	-	-	-	-	90	150,60	21,29	180	450,18	1,56
	40	120	200^+	51,06	-	-	-	-	120	200,80	31,89	240	600,24	2,87
	50	150	250^+	75,45	-	-	-	-	150	251,00	58,05	300	750,30	5,93
	100	300	500^+	343,00	-	-	-	-	300	502,00	390,63	600	1 500,60	31,96
	150	450	750^+	1 855,53	-	-	-	-	450	753,00	1 336,43	900	2 250,90	90,94
	200	600	1000^+	4 658,95	-	-	-	-	*	*	*	1 200	2 001,20	198,97
tempo-width	2	6	5^+	0,47	0,54	18	8,002	0,02	6	10,04	18,83	9	15,008	0,06
	3	9	5^+	0,54	2,09	27	8,002	0,03	*	*	*	12	15,010	0,09
	4	12	5^+	0,77	6,23	36	8,002	0,09	*	*	*	15	15,012	0,11
	5	15	5^+	1,10	14,93	45	8,002	0,23	*	*	*	18	15,014	0,14
	6	18	5^+	1,55	41,06	54	8,002	1,80	*	*	*	21	15,016	0,17
	7	21	5^+	2,11	*	63	8,002	27,93	*	*	*	24	15,018	0,19
	8	24	5^+	2,85	*	72	8,002	538,70	*	*	*	27	15,020	0,25
	9	27	5^+	3,77	*	-	-	-	*	*	*	30	15,022	0,30
	10	30	5^+	3,92	*	-	-	-	*	*	*	33	15,024	0,62
	20	60	5^+	36,89	*	-	-	-	*	*	*	63	15,044	1,51
	30	90	5^+	140,77	*	-	-	-	*	*	*	93	15,064	5,70
	40	120	5^+	315,74	*	-	-	-	*	*	*	123	15,084	12,68
	50	150	5^+	576,45	*	-	-	-	*	*	*	153	15,104	24,82
	60	180	5^+	945,25	*	-	-	-	*	*	*	183	15,124	42,19
	70	210	5^+	1 447,07	*	-	-	-	*	*	*	213	15,144	67,69

Sur le domaine "tempo-depth", LPGP explose dès le problème tempo-depth-3 qu'il résout en plus de trois heures et VHPOP, s'il est meilleur que LPGP sur la plupart des problèmes, ne dispose pas de suffisamment de mémoire pour résoudre le problème tempo-depth-200. TLP-GP résout tous ces problèmes avec des temps de recherche plus importants que VHPOP mais qui restent du même ordre de grandeur. Il donne une solution flottante pour les 40 premiers problèmes en moins d'une minute et résout encore le problème tempo-depth-150 en moins d'une heure avec une base de 1.197 contraintes simples et 3.439 contraintes disjonctives. CRIKEY est le plus performant et résout l'ensemble des problèmes en moins de 200 secondes.

Cependant, le nombre d'actions des plans-solutions est multiplié par 2 et leurs temps d'exécution par 3 par rapport à ceux retournés par TLP-GP, et ils ne peuvent pas être compactés.

Sur le domaine "tempo-width", LPGP résout avec succès les problèmes jusqu'à tempo-width-8. VHPOP ne résout que le premier problème et ne dispose pas de suffisamment de mémoire pour résoudre les suivants. TLP-GP résout tous ces problèmes. Il donne une solution flottante pour les 10 premiers problèmes en moins de 5 s., pour les 50 premiers en moins de 10 minutes, et résout encore le problème tempo-width-70 en moins d'une demi-heure alors que la base de contraintes pour ce problème atteint 350 contraintes simples et 34.857 contraintes disjonctives. CRIKEY est encore le meilleur en terme de performances ; il résout le dernier problème en à peine plus d'une minute. Cependant, si le nombre d'actions des plans-solutions est sensiblement le même que pour TLP-GP, leurs temps d'exécution est à nouveau multiplié par 3.





Problème		TLP-GP-1			TLP-GP-2	LPGP			VHPOP2.2			CRIKEY3		
		Act.	Make span	Temps CPU (s)	Temps CPU (s)	Act.	Make span	Temps CPU (s)	Act.	Make span	Temps CPU (s)	Act.	Make span	Temps CPU (s)
tempo-matrix	2x2	12	10^+	1,08	25,95	36	24,002	1,33	*	*	*	12	10,009	10,72
	3x3	27	15^+	3,99	*	-	-	-	*	*	*	-	-	-
	4x4	48	20^+	14,32	*	-	-	-	*	*	*	-	-	-
	5x5	75	25^+	46,57	*	-	-	-	*	*	*	-	-	-
	6x6	108	30^+	134,19	*	-	-	-	*	*	*	-	-	-
	7x7	147	35^+	358,47	*	-	-	-	*	*	*	-	-	-
	8x8	192	40^+	1 058,75	*	-	-	-	*	*	*	-	-	-
	9x9	*	*	*	*	-	-	-	*	*	*	-	-	-
tms-2-3-light	3	9	20	0,74	0,49	-	-	-	8	14,04	0,06	10	25,003	289,36
	4	17	20	1,41	0,48	-	-	-	13	14,04	33,08	*	*	*
	5	17	20	1,47	0,65	-	-	-	14	14,04	37,67	*	*	*
	6	24	20	2,41	0,68	-	-	-	*	*	*	*	*	*
	7	25	20	2,63	0,84	-	-	-	*	*	*	*	*	*
	8	31	20	3,81	0,89	-	-	-	*	*	*	*	*	*
	9	31	20	4,02	1,09	-	-	-	*	*	*	*	*	*
	10	32	20	4,68	1,22	-	-	-	*	*	*	*	*	*
	20	73	20	24,07	6,68	-	-	-	*	*	*	*	*	*
	30	103	20	61,67	28,96	-	-	-	*	*	*	*	*	*
	40	143	20	167,52	90,82	-	-	-	*	*	*	*	*	*
	50	175	20	333,74	213,58	-	-	-	*	*	*	*	*	*
	60	215	20	629,22	440,35	-	-	-	*	*	*	*	*	*
	70	245	20	1 047,03	980,95	-	-	-	*	*	*	*	*	*
	80	277	20	1 820,51	1 729,95	-	-	-	*	*	*	*	*	*
	90	-	-	-	2 800,63	-	-	-	*	*	*	*	*	*

Sur le domaine "tempo-matrix", LPGP n'arrive à résoudre que le problème tempo-matrix-2x2 en moins d'une heure et VHPOP ne résout aucun de ces problèmes par manque de mémoire. TLP-GP résout encore le problème tempo-matrix-8x8 en moins de 20 minutes avec une base de 488 contraintes simples et 6.256 contraintes disjonctives.

Sur le domaine "tms-2-3", LPGP ne résout aucun des problèmes. VHPOP résout les trois premiers problèmes en moins d'un minute, mais échoue sur le problème tms-2-3-06 par manque de mémoire. TLP-GP résout 29 problèmes en moins d'une minute. Le problème tms-2-3-80 est le dernier problème résolu par TLP-GP ; il nécessite 30 mn pour être résolu avec une base de 474 contraintes simples et 6.873 contraintes disjonctives. Pour ce dernier problème, le plan-solution flottant contient 277 actions.

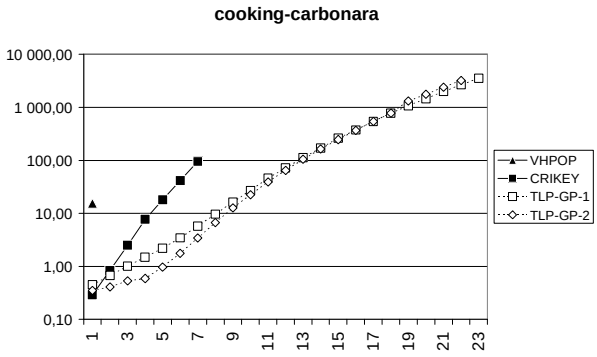
Ces tests montrent sans ambiguïté que, pour des problèmes temporellement expressifs, notre approche est bien plus efficace que celle de LPGP ou VHPOP. Par ailleurs, lorsque la complexité des domaines augmente, elle reste efficace là où CRIKEY n'arrive plus à résoudre que quelques problèmes.

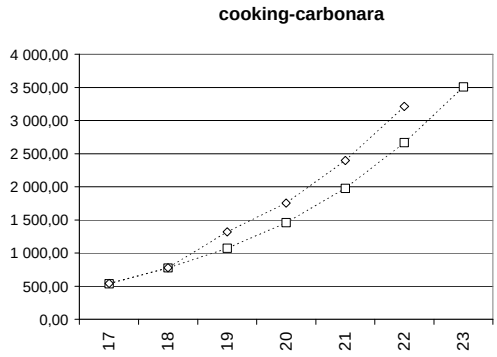
Nous allons maintenant présenter les résultats expérimentaux que nous avons obtenus sur les problèmes utilisant l'expressivité du langage de représentation de TLP-GP.

b) Problèmes temporels codés avec les extensions de [Smith, 2003]

Pour le domaine suivant, VHPOP et CRIKEY ne prennent pas en compte les extensions d'expressivité de [Smith, 2003]. Pour tester ces planificateurs, nous utiliserons donc la version traduite en PDDL2.1. Nous n'avons pas testé les problèmes avec LPGP en raison des trop faibles performances qu'il montre sur les problèmes précédents. En effet, il serait nécessaire, en plus de la décomposition en sous-actions de la traduction en PDDL2.1, de découper chacune de ces sous-actions en trois nouvelles actions (start, invariant et end). Cette décomposition donnerait par exemple un domaine "cooking" avec 54 opérateurs, qui, au vu des résultats précédents, n'a aucune chance d'être résolu.

Sur le domaine "cooking", TLP-GP est capable de planifier en moins d'une minute la production et la consommation de 11 plats de pâtes à la carbonara. Il peut encore résoudre les problèmes jusqu'à 23 plats de pâtes en moins d'une heure, mais échoue sur le problème cooking-carbonara-26 par manque de mémoire. Pour la version du domaine "cooking" traduite en PDDL2.1, VHPOP et CRIKEY ne parviennent à résoudre aucun des problèmes par manque de mémoire. Pour permettre à ces planificateurs de résoudre ces problèmes, nous avons donc relaxé légèrement le domaine en augmentant la durée des opérateurs décomposés en sous-actions, ce qui donne une certaine liberté de mouvement aux sous-actions en autorisant une marge d'erreur temporelle dans les plans-solutions. Pour des durées augmentées de 0,01, VHPOP ne résout aucun problème et CRIKEY échoue à partir du 4ème. Nous avons donc augmenté les durées de 0,1, ce qui donne encore une marge d'erreur temporelle relativement faible par rapport au problème original. Notons cependant que, même si ce n'est pas le cas pour le domaine "cooking", sur d'autres domaines, cette relaxation des contraintes temporelles pourrait générer des plans-solutions au problème relaxé qui ne soient pas des solutions du problème original. Malgré cette relaxation, VHPOP ne permet de résoudre que le premier problème. CRIKEY est moins performant que TLP-GP et ne résout pas le problème cooking-carbonara-pdd2.1-7 équivalent à cooking-carbonara-7. Si l'on omet les sous-actions de la traduction en PDDL2.1, les plans retournés par tous ces planificateurs sont les mêmes. Le temps d'exécution est toujours de 17 (ou légèrement supérieur en raison de la marge d'erreur pour VHPOP ou CRIKEY).





Problème		TLP-GP-1				TLP-GP-2				VHPOP2.2		CRIKEY3	
		STC	DTC	Act.	Temps CPU (s)	Var.R.	Var.P.	Clauses	Temps CPU (s)	Act.	Temps CPU (s)	Act.	Temps CPU (s)
cooking-carbonara	01	11	22	8	0,45	23	58	291	0,35	18	15,55	18	0,29
	02	20	44	13	0,67	32	90	459	0,41	*	*	27	0,84
	03	29	68	18	1,01	41	122	627	0,53	*	*	36	2,50
	04	38	94	23	1,49	50	154	795	0,59	*	*	45	7,68
	05	47	122	28	2,20	59	186	963	0,97	*	*	54	17,92
	06	56	152	33	3,44	68	218	1 131	1,77	*	*	63	41,49
	07	65	184	38	5,71	77	250	1 299	3,41	*	*	72	94,79
	08	74	218	43	9,64	86	282	1 467	6,70	*	*	*	*
	09	83	254	48	16,30	95	314	1 635	12,65	*	*	*	*
	10	92	292	53	26,82	104	346	1 803	22,38	*	*	*	*
	11	101	332	58	46,12	113	378	1 971	39,17	*	*	*	*
	12	110	374	63	71,56	122	410	2 139	64,89	*	*	*	*
	13	119	418	68	112,19	131	442	2 307	104,45	*	*	*	*
	14	128	464	73	170,69	140	474	2 475	163,26	*	*	*	*
	15	137	512	78	260,28	149	506	2 643	246,13	*	*	*	*
	16	146	562	83	373,66	158	538	2 811	365,76	*	*	*	*
	17	155	614	88	537,41	167	570	2 979	541,49	*	*	*	*
	18	164	668	93	775,79	176	602	3 147	777,81	*	*	*	*
	19	173	724	98	1 071,92	185	634	3 315	1 319,58	*	*	*	*
	20	182	782	103	1 457,40	194	666	3 483	1 757,41	*	*	*	*
	21	191	842	108	1 977,49	203	698	3 651	2 395,87	*	*	*	*
	22	200	904	113	2 664,92	212	730	3 819	3 211,08	*	*	*	*
	23	209	968	118	3 507,02	-	-	-	-	*	*	*	*

Ces tests montrent sans ambiguïté que, pour des problèmes temporellement expressifs utilisant les extensions d'expressivité de [Smith, 2003], notre approche est bien plus efficace que celles des planificateurs de référence.

Si l'on compare nos deux approches, TLP-GP-2 est meilleur jusqu'au 17ème problème, mais à partir du 18ème, TLP-GP-1 obtient de meilleurs résultats et creuse rapidement l'écart. Pour la suite, nous n'utiliserons que TLP-GP-1 car il montre globalement de meilleurs résultats que TLP-GP-2.

c) Problèmes temporels codés en utilisant les modalités temporelles

Sur le domaine "temporal-machine-shop-2-3", l'assemblage et la cuisson de 10 céramiques est résolue par TLP-GP-1 en moins d'une minute. Nous obtenons encore un plan-solution flottant de 324 actions pour 80 céramiques en moins d'une heure.

Problème		TLP-GP-1				
		STC	DTC	Act.	Make span	Temps CPU (s)
temporal-machine-shop-2-3	03	15	38	12	42^+	2,16
	04	26	78	20	42^+	3,63
	05	28	87	21	42^+	4,33
	06	39	137	28	42^+	7,50
	07	41	147	29	42^+	7,00
	08	52	209	36	42^+	11,06
	09	54	224	37	42^+	12,38
	10	65	340	43	42^+	20,80
	20	130	918	84	42^+	71,87
	30	195	1 983	124	42^+	195,94
	40	260	3 163	164	42^+	397,10
	50	325	4 869	204	42^+	745,81
	60	390	6 654	244	42^+	1 263,49
	70	455	9 087	284	42^+	2 060,19
	80	520	11 841	324	42^+	3 429,56

A ce jour, aucun autre planificateur ne prend en compte les modalités que nous avons introduites et il est impossible de comparer TLP-GP avec d'autres systèmes sur ce point. Nous souhaitons cependant proposer leur intégration à une nouvelle catégorie de planificateurs "temporellement expressifs" pour les futures compétitions IPC afin d'encourager le développement de systèmes encore plus expressifs permettant d'envisager la résolution efficace de problèmes réels.

V. Conclusion et perspectives

1. Ce qui a été fait…

Dans cette thèse, nous avons proposé plusieurs améliorations à l'état de l'art de la planification SAT dans le cadre classique et à la planification dans le cadre temporel. Dans les deux cas, nous avons travaillé en utilisant des solveurs dédiés, en justifiant ce choix par le fait que tous les gains en efficacité de ces solveurs permettent automatiquement des gains de performance pour les planificateurs qui les utilisent.

En ce qui concerne l'approche SAT de la planification, les progrès dans les codages, linéaires à l'origine, puis parallèles et enfin utilisant les relations d'autorisation forte et faible participent également à l'amélioration de l'efficacité en permettant d'obtenir des bases de clauses plus compactes et pour lesquelles il est plus facile de trouver un modèle.

Dans ce contexte, nous avons réalisé un travail qui contribue à l'amélioration des perfomances de cette approche et porte sur plusieurs aspects. Nous avons d'abord proposé, en leur intégrant la relation d'autorisation, des codages plus performants dans les espaces d'états, dans les espaces de plans, et pour le graphe de planification. Cette intégration de la relation d'autorisation permet de diminuer de manière significative le nombre de clauses, de variables et de niveaux nécessaires pour représenter les plans-solutions. L'implémentation du planificateur TSP qui intègre un traducteur spécifique nous a ensuite permis de mesurer objectivement les performances de ces différents codages. Dans la large majorité des exemples testés, nous avons ainsi obtenu des diminutions importantes des temps de résolution avec les codages qui utilisent la relation d'autorisation. Les codages dans les espaces d'états (LCS) et dans les espaces de plans (LCP) que nous avons développé s'avèrent ainsi être les plus performants (aussi bien en nombre de clauses et de variables que par leurs performances pour la résolution). Nous avons enfin étudié plusieurs codages récents et proposé un nouveau codage compact qui permet de bénéficier des avantages de la relation d'autorisation et d'un découpage des opérateurs.

La prise en compte d'aspects temporels étant indispensable à la modélisation de problèmes plus proches du monde réel, nous nous sommes ensuite intéressés au cadre de la planification temporelle. Nous avons d'abord dressé un état de l'art des techniques utilisées par les planificateurs temporels actuels et présenté les principaux langages de représentation de problèmes temporels et leur expressivité. Nous nous sommes alors intéressés à la résolution de problèmes qui nécessitent impérativement la concurrence des actions pour leur résolution (problèmes temporellement expressifs). Après avoir remarqué que très peu de planificateurs actuels permettaient de résoudre efficacement ce type de problèmes, nous avons développé deux algorithmes spécifiques que nous avons ensuite implémentés dans le planificateur TLP-GP. Dans ces deux algorithmes, comme dans le cadre de la planification SAT, un solveur est utilisé pour rechercher une solution (DTP ou SMT). Notre planificateur permet de résoudre

des problèmes temporellements expressifs représentés dans un langage dont l'expressivité est supérieure à celle de PDDL2.1 sans que cette extension se fasse au détriment de la complétude de l'algorithme.

Nous avons ensuite étendu l'expressivité de PDDL2.1 et intégré ces extensions dans TLP-GP. L'expressivité de notre langage de représentation permet de prendre en compte des préconditions qui peuvent être requises et des effets qui peuvent apparaître sur n'importe quel intervalle temporel relatif à un instant caractéristique d'une action (par exemple son instant de début ou de fin). TLP-GP peut aussi prendre en compte, de manière simple, des événements exogènes, des buts temporellement étendus, des fenêtres d'activation… Il est également capable d'interpréter plusieurs modalités pour la sémantique des intervalles temporels associés aux effets et préconditions des actions. Ces modalités étendent significativement l'expressivité de son langage de représentation par une certaine prise en compte de l'incertitude, du choix, ou des transitions continues.

Même si TLP-GP s'est avéré bien moins performant que les autres planificateurs temporels sur les benchmarks temporellement simples de la dernière compétition IPC'2008 à laquelle nous avons participé, il est le seul capable de résoudre des problèmes temporellement expressifs. Pour mener notre étude expérimentale, nous avons mis au point un ensemble de nouveaux benchmarks temporellement expressifs. Sur ces benchmarks, TLP-GP se montre bien plus performant que trois planificateurs de référence (LPGP, VHPOP et CRIKEY3) capables de traiter le même type de problèmes. Ces résultats permettent d'envisager la représentation et la résolution de problèmes plus proches de problèmes réels. La production d'un plan-solution flottant plutôt que d'une solution fixe autorise également une plus grande flexibilité lors de l'exécution du plan.

2. … et ce qui reste à faire.

De nombreux points restent encore à traiter. Pour continuer d'améliorer la planification SAT, plusieurs voies méritent encore d'être explorées :

- Perfectionner les codages. Nous avons par exemple proposé un codage compact basé à la fois sur la relation d'autorisation et sur un découpage des opérateurs qui permet d'obtenir une base de clauses réduite et qui reste à implémenter et à tester.
- Fournir au solveur SAT une heuristique permettant de diriger l'ordre d'instanciation des variables propositionnelles. Celle-ci pourrait prendre en compte des propriétés particulières des problèmes comme la hiérarchisation du domaine ou la symétrie des objets du problème.
- Implémenter un solveur dédié travaillant directement à partir du graphe de planification pour tirer parti de la structure du problème, comme par exemple DPPLAN [Baioletti, Marcugini, et Milani 2000] et LCDPP [Vidal 2001], [Vidal 2002].

Ces possibilités permettent de penser que les méthodes de planification SAT, qui sont assez récentes peuvent encore progresser.

De nombreux points restent également à explorer dans le cadre de la planification temporelle :

- Tester, dans TLP-GP-1, des heuristiques plus informatives pour choisir les buts à asserter et les actions à utiliser.
- Adapter un solveur DTP pour rajouter incrémentalement les nouvelles contraintes à la base de contraintes initiale, afin de vérifier plus rapidement la satisfiabilité (au lieu de relancer complètement le travail du solveur à chaque appel).
- Rechercher des heuristiques plus efficaces, basées sur l'estimation du temps d'exécution, l'analyse des contraintes de la base de contraintes temporelles… pour extraire plus efficacement une solution dans TLP-GP-1.
- Travailler sur la production de plans optimaux en temps d'exécution, au moins pour un nombre de niveaux k donné.

TABLE DES MATIERES

I. Introduction ... 3
1. Introduction générale au domaine ... 3
2. Les différents cadres de travail concernés ... 4
2.1. Le cadre classique ... 4
2.2. Le cadre temporel ... 6
2.3. La planification par compilation ... 6
3. Présentation de la thèse ... 7
II. Algorithmes de planification dans le cadre classique ... 9
1. Définitions ... 9
2. Le langage PDDL ... 11
3. Recherche dans les espaces d'états ... 13
3.1. Définitions ... 13
3.2. Algorithmique ... 14
4. Recherche dans les espaces de plans partiels ... 15
4.1. Définitions ... 15
4.2. Algorithmique ... 18
5. Graphplan ... 19
5.1. Définitions ... 19
5.2. Algorithmique ... 21
6. Conclusion ... 23
III. Planification SAT : le système TSP ... 25
1. Introduction ... 25
2. Etat de l'art de la planification SAT ... 27
2.1. Définitions préliminaires ... 27
2.2. Historique ... 27
2.3. Nomenclature des différents codages ... 33
2.4. Codages MK99 et V01 dans les espaces d'états ... 33
2.4.1. Codage dans les espaces d'états avec frame-axiomes explicatifs ... 33
2.4.2. Codage dans les espaces d'états avec no-ops ... 35
2.5. Codages MK99 et V01 dans les espaces de plans ... 35
2.5.1. Partie commune des codages dans les espaces de plans ... 36
2.5.2. Codage par liens causaux, protection d'intervalles et ordre partiel ... 38
2.5.3. Codage par liens causaux, protection d'intervalles et étapes contiguës ... 39
2.5.4. Codage du "white knight" ... 40
2.6. Codages KS99 et V01 du graphe de planification ... 41
2.6.1. Codage KS99 du graphe de planification ... 42
2.6.2. Codage V01 du graphe de planification ... 42
3. Introduction de la relation d'autorisation dans les codages ... 43
3.1. Définition de la relation d'autorisation ... 43
3.1.1. Introduction ... 43
3.1.2. Définitions préliminaires ... 44
3.2. Intégration de la relation d'autorisation dans les codages ... 46
3.2.1. Nomenclature des différents codages ... 46
3.2.2. Codages dans les espaces d'états et relation d'autorisation ... 46
3.2.3. Codages dans les espaces de plans et relation d'autorisation ... 48
3.2.4. Codage du graphe de planification et relation d'autorisation ... 49
4. Automatisation de la traduction : le traducteur TSP ... 49
4.1. Principe de fonctionnement général de TSP ... 49

4.2. Syntaxe et sémantique de TSPL 50
4.2.1. Structure d'une spécification TSPL 50
4.2.2. Syntaxe, sémantique et types de TSPL 51
4.2.3. Données prédéfinies par TSP 52
4.3. Fonctionnement interne du traducteur TSP 53
4.4. Codage des traductions en TSPL 55
4.5. Etude de la complexité des codages 55
5. Etude comparative 56
5.1. Les benchmarks utilisés 57
5.2. Etude comparative des traductions MK99 et V01 58
5.2.1. Tests des codages dans les espaces d'états 58
5.2.2. Tests des codages dans les espaces de plans 59
5.3. Etude comparative des traductions V01, LCS et LCP (introduction de l'autorisation) 59
5.3.1. Etude par recherche incrémentale du plan solution 59
5.3.2. Etude des performances sur le dernier niveau 61
5.4. Etude comparative des traductions KS99 et V01 61
6. Etude et amélioration de codages récents 63
6.1. Nomenclature des différents codages 63
6.2. Codages avec indépendance et sémantique ∀-Step 64
6.3. Codages avec autorisation et sémantiques ∃-Step / R∃-Step 66
6.4. Un codage compact utilisant la relation d'autorisation faible 68
7. Conclusion 70
IV. Planification temporelle : le système TLP-GP 71
1. Introduction 71
2. Etat de l'art de la planification temporelle 72
2.1. Représentation de problèmes de planification temporels 72
2.1.1. Les logiques temporelles pour la planification 73
2.1.2. PDDL 2.1 : une extension pour la prise en compte de domaines temporels 76
2.1.3. Expressivité temporelle 79
2.2. Les premiers planificateurs temporels 83
2.3. Algorithmique des planificateurs temporels actuels 84
2.3.1. Recherche dans les espaces d'états étendus 84
2.3.2. Recherche dans les espaces de plans 87
2.3.3. Extensions temporelles de GRAPHPLAN 91
2.4. Taxinomie des principaux planificateurs temporels 96
2.4.1. Planificateurs temporellement simples 96
2.4.2. Planificateurs temporellement expressifs 98
2.4.3. Synthèse 99
3. TLP-GP : un planificateur pour résoudre des problèmes temporellement-expressifs 101
3.1. Introduction 101
3.2. Extension de l'expressivité de PDDL2.1 aux intervalles temporels 101
3.2.1. Prise en compte des extensions de [Smith, 2003] 101
3.2.2. Exemples d'utilisation de cette expressivité 103
3.3. Algorithmique de TLP-GP 103
3.3.1. Expansion du graphe de planification temporel 103
3.3.2. Extraction d'un plan-solution flottant en recherche arrière 105
3.3.3. Extraction d'un plan-solution flottant par codage du graphe 110
3.4. Présentation de l'interface graphique 115
3.5. Extension supplémentaire de l’expressivité 116

3.5.1. Définition d'instants caractéristiques pour les actions 117
3.5.2. Définition de modalités pour les intervalles temporels 117
3.5.3. Exemples d'actions utilisant les instants caractéristiques ou les modalités 122
3.6. Benchmarks temporellement expressifs 123
3.6.1. Les benchmarks des IPC 123
3.6.2. Elaboration de benchmarks temporellement expressifs 124
3.7. Résultats expérimentaux 127
3.7.1. Introduction 127
3.7.2. Comparaison avec les planificateurs de la compétition IPC'2008 127
3.7.3. Comparaison avec les planificateurs temporellement expressifs 130
V. Conclusion et perspectives 135
1. Ce qui a été fait…..... 135
2. … et ce qui reste à faire. 136

TABLE DES DEFINITIONS

Définition 1 (opérateur) 9
Définition 2 (état, fluent) 9
Définition 3 (action) 9
Définition 4 (problème de planification) 10
Définition 5 (planificateur sain) 10
Définition 6 (planificateur complet) 10
Définition 7 (planificateur systématique) 10
Définition 8 (planificateur optimal) 10
Définition 9 (application d'une action à un état) 13
Définition 10 (régression d'un état par une action) 13
Définition 11 (plan séquentiel, espaces d'états) 13
Définition 12 (application d'une séquence d'actions à un état) 14
Définition 13 (plan-solution séquentiel, correct, espaces d'états) 14
Définition 14 (plan-solution séquentiel minimal en nombre d'actions) 14
Définition 15 (plan partiel) 15
Définition 16 (parties d'un plan partiel) 16
Définition 17 (demandeur) 16
Définition 18 (établisseur) 16
Définition 19 (lien causal) 16
Définition 20 (casseur, menace) 16
Définition 21 (menace positive, menace négative) 17
Définition 22 (chevalier blanc) 17
Définition 23 (choix d'un établisseur) 17
Définition 24 (promotion d'un casseur) 17
Définition 25 (rétrogradation, démotion d'un casseur) 17
Définition 26 (choix d'un white knight) 17
Définition 27 (séparation casseur / demandeur) 17
Définition 28 (indépendance entre deux actions) 20
Définition 29 (ensemble d'actions indépendant) 20
Définition 30 (autorisation forte entre deux actions) 44
Définition 31 (séquence fortement autorisée) 44
Définition 32 (autorisation faible entre deux actions) 45
Définition 33 (séquence faiblement autorisée) 45
Définition 34 (opérateur avec effets conditionnels) 64
Définition 35 (plans ∀-Step) 64
Définition 36 (plans ∃-Step) 66
Définition 37 (graphe de désactivation) 67
Définition 38 (plan concurrent / séquentiel) 79
Définition 39 (problème temporellement expressif) 79
Définition 40 (problème temporellement simple) 79
Définition 41 (domaine temporellement expressif) 80
Définition 42 (domaine temporellement simple) 80
Définition 43 (domaine temporellement simple mixte) 80
Définition 44 (condition-avant / condition-après) 81
Définition 45 (trou temporel / instant critique) 81
Définition 46 (planificateur temporellement simple) 82
Définition 47 (planificateur temporellement expressif) 82
Définition 48 (état temporel) 85

Définition 49 (état temporel flottant) 86
Définition 50 (STP/STN) 88
Définition 51 (DTP) 88
Définition 52 (plan-solution flottant) 109
Définition 53 (action de PDDL temporel étendu) : 117
Définition 54 (modalités atomiques *supported*/*forbidden*) : 118
Définition 55 (forme temporelle) : 118
Définition 56 (effet-cohérence) 118
Définition 57 (précondition-cohérence) 118

BIBLIOGRAPHIE

[Allen, 1984] J.F. Allen, "Towards a general theory of action and time", Artificial Intelligence 23, pp. 123-154, 1984.

[Allen, 1991] J.F. Allen, "Temporal reasoning and planning", In J.F. Allen, H. Kautz, R. Pelavin, and J. Tenenberg, "Reasoning about Plans", Morgan Kaufmann, 1991.

[Allen, Ferguson, 1994] J.F. Allen, G. Ferguson, "Arguing about Plans: Plan Representation and Reasoning for Mixed-initiative Planning", AIPS 1994, pp. 43-48.

[Allen, Koomen, 1983] J.F. Allen, J.A.G.M. Koomen, "Planning Using a Temporal World Model", IJCAI 1983, pp. 741-747.

[Bacchus, Kabanza, 1995] F. Bacchus, F. Kabanza, "Using temporal logic to control search in a forward-chaining planner", in: M. Ghallab, A. Milani (Eds.), New directions in AI Planning, IOS Press, Amsterdam, Netherlands, 1996, pp. 141-153.

[Bacchus, Kabanza, 2000] F. Bacchus, F. Kabanza, "Using temporal logics to express search control knowledge for planning", Artificial Intelligence 116(1-2), 123-191 (2000).

[Baioletti, Marcugini, Milani, 1998a] M. Baioletti, S. Marcugini, A. Milani, « An extension of SATPLAN for Planning with Constraints », 8th International Conference on Artificial Intelligence: Methodology, Systems, and Applications, AIMSA'98, pp. 39-49, 1998.

[Baioletti, Marcugini, Milani, 1998b] M. Baioletti, S. Marcugini, A. Milani, « C-SATPlan: a SATPlan-based tool for planning with constraints », Workshop Planning As Combinatorial Search, AIPS'98, 1998.

[Baioletti, Marcugini, Milani, 2000] M. Baioletti, S. Marcugini, A. Milani, « DPPLAN: An Algorithm for Fast Solutions Extraction from a Planning Graph », 5th International Conference on Artificial Intelligence Planning and Scheduling, AIPS'00, pp. 13-21, 2000.

[Bayardo, Schrag, 1997] R.J.J. Bayardo, R.C. Schrag, "Using CSP look-back techniques to solve real-world SAT instances", in: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), Madison, WI, USA, 1998, pp. 203-208.

[Blum, Furst, 1995] A.L. Blum, M.L. Furst, "Fast planning through planning-graphs analysis", in: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montréal, Québec, Canada, 1995, pp. 1636-1642.

[Blum, Furst, 1997] A.L. Blum, M.L. Furst, "Fast planning through planning-graphs analysis", Artificial Intelligence 90 (1997) 281- 300.

[Bonet, Geffner, 1998] B. Bonet, H. Geffner, "HSP: Heuristic search planner", Planning Competition of the 4th International Conference on Artificial Intelligence Planning and Scheduling (AIPS− 98), Pittsburgh, PA, USA, 1998.

[Bonet, Loerincs, Geffner, 1997] B. Bonet, G. Loerincs, H. Geffner, « A robust and fast action selection mechanism for planning », Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, pp. 714− 719, Providence, Rhode Island, USA, 1997.

[Borning et al., 1997] A. Borning, K. Marriot, P. Stuckey, Y. Xiao, "Solving linear arithmetic constraints for user interface applications", in: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology (UIST- 97), Banff, Alberta, Canada, 1997, pp. 87- 96.

[Bylander, 1994] T. Bylander, "The Computational Complexity of Propositional STRIPS Planning", Artificial Intelligence 69(1-2), 165-204, (1994).

[Castellini, Giunchiglia, Tacchclla, 2001] C. Castellini, E. Giunchiglia, A. Tacchclla, « Improvements to SAT-based conformant planning », European Conference on Planning ECP'01, pp. 241-252, 2001.

[Castellini, Giunchiglia, Tacchclla, 2003] C. Castellini, E. Giunchiglia, A. Tacchclla, « SAT -based planning in complex domains: Concurrency, constraints and nondeterminism », Artificial Intelligence 147, pp. 85-118, 2003.

[Cayrol, Régnier, Vidal, 2000] M. Cayrol, P. Régnier, V. Vidal, « New results about LCGP, a least commited GRAPHPLAN », International Conference on Artificial Intelligence Planning and Scheduling Systems, AIPS'00, 2000.

[Cayrol, Régnier, Vidal, 2001] M. Cayrol, P. Régnier, V. Vidal, « Least commitment in GRAPHPLAN », Artificial Intelligence, 130, pp. 85-118, 2001.

[Chapman, 1987] D. Chapman, "Planning for conjunctive goals", Artificial Intelligence 32 (1987), 333- 377.

[Chen, Wah, Hsu, 2006] Y.X.Chen, B.W.Wah, C.W.Hsu, "Temporal planning using subgoal partitioning and resolution in SGPlan", Journal of AI Research, 2006.

[Chen, Zhao, Zhang, 2007] Y. Chen, X. Zhao, W. Zhang, « Long Distance Mutual Exclusion for Propositional Planning », International Joint Conference on Artificial Intelligence, IJCAI'07, 2007.

[Coles, Fox, Long, Smith, 2008] A. Coles, M. Fox, D. Long, A. Smith, "Planning with Problems Requiring Temporal Coordination", AAAI 2008, pp. 892-897.

[Crawford, Auton, 1996] J. Crawford, L. Auton, "Experimental results on the crossover point in satisfiability problems", Artificial Intelligence 81 (1996), 31- 58.

[Cushing et al., 2007.a] W. Cushing, S. Kambhampati, Mausam, D.S. Weld, "When is Temporal Planning Really Temporal?", IJCAI 2007, 1852-1859.

[Cushing et al., 2007.b] W. Cushing, D.S. Weld, S. Kambhampati, Mausam, K. Talamadupula, "Evaluating Temporal Planning Domains", ICAPS 2007, 105-112.

[Davidson, 1967] D. Davidson, "The Logical Form of Action Sentences", in N. Rescher (ed.), The Logic of Decision and Action.

[Davis, Logemann, Loveland, 1962] M. Davis, G. Logemann, D. Loveland, "A computing procedure for quantification theory", Communications of the ACM 5 (1962) 394- 397.

[Davis, Putnam, 1960] M. Davis, H. Putnam, "A computing procedure for quantification theory", Journal of the ACM 7 (1960) 201- 215.

[Dean, Firby, Miller, 1988] T.Dean, J.Firby, D.Miller, "Hierarchical Planning involving deadlines, travel time and ressources", Computational Intelligence 6(1), 381-398, 1988.

[Dean, McDermott, 1987] T. Dean, D.V. McDermott, "Temporal Data Base Management", Artificial Intelligence 32(1), pp. 1-55, 1987.

[Dechter, Meiri, Pearl, 1991] R. Dechter, I. Meiri, J. Pearl: "Temporal Constraint Networks", Artificial Intelligence 49(1-3), 61-95, (1991).

[Dimopoulos, Nebel et Koehler 1997] Y. Dimopoulos, B. Nebel, J. Koehler, "Encoding planning problems in nonmonotonic logic programs", in: Proceedings of the 4th European Conference on Planning (ECP-97), Toulouse, France, 1997, pp. 169-181.

[Dinh, Smith, 2003] T.B.Dinh, B.M.Smith, "CPPlanner: A Temporal Planning System using Critical Paths", APES, 2003.

[Do, Kambhampati 2001] M.B. Do, S. Kambhampati, "Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP", Artificial Intelligence 132(2), 151-182, (2001).

[Do, Srivastava, Kambhampati, 2000a] M. B. Do, B. Srivastava, S. Kambhampati, « Investigating the effect of relevance and reachability constraints in SAT Encoding of Planning », Fifth International Conference on Artificial Intelligence Planning and Scheduling, AIPS'00, 2000.

[Do, Srivastava, Kambhampati, 2000b] M. B. Do, B. Srivastava, S. Kambhampati, « On the Use of LP Relaxations in Solving SAT encodings of Planning Problems », Workshop on the Integration of AI and OR Techniques for Combinatorial Optimization, AAAI'00, 2000.

[Edelkamp, Helmert, 2001] S.Edelkamp, M.Helmert, "The Model Checking Integrated Planning System", AI Magazine, 22(3), 2001.

[Ernst, Millstein, Weld, 1997] M. Ernst, T. Millstein, D. Weld, « Automatic SAT-Compilation of Planning Problems », Proceedings of the 15th International Joint Conference on Artificial Intelligence, IJCAI'97, 1997

[Ferraris, Giunchiglia, 2000] P. Ferraris, E. Giunchiglia, « Planning as satisfiability in non deterministic domains », AAAI / IAAI'00, 2000.

[Fikes et Nilsson 1971] R.E. Fikes, N.J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving", Artificial Intelligence 2 (1971) 189-208.

[Fox, Long, 2003] M.Fox, D.Long, "PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains", Journal of AI Research, 20, 2003.

[Garrido, Fox, Long, 2002] A.Garrido, M.Fox; D.Long, "TPSYS: Temporal Planning with PDDL2.1", ECAI, 2002.

[Gerevini, Saetti, Serina, 2006] A.Gerevini, A.Saetti, I.Serina, "An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events", JAIR, 25, 2006.

[Ghallab, Allaoui, 1989] M.Ghallab, A.M.Alaoui, "Managing Efficiently Temporal Relations Through Indexed Spanning Trees", IJCAI, 1989.

[Ghallab, Laruelle, 1994] M.Ghallab , H.Laruelle, "Representation and Control in Ixtet, a Temporal Planner", AIPS, 1994.

[Giunchiglia, 2000] E. Giunchiglia, « Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism », Seventh International Conference on Principles of Knowledge Representation and Reasoning, KR'00, 2000.

[Giunchiglia, Massarotto, Sebastiani, 1998] E. Giunchiglia, A. Massarotto, R. Sebastiani, « Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability », Tenth Conference on Innovative Applications of Artificial Intelligence, IAAI'98, 1998.

[Hasley, Long, Fox, 2004] K.Halsey, D.Long, and M..Fox,"CRIKEY - A Planner Looking at the Integration of Scheduling and Planning", Proc. of the “Integration Scheduling Into Planning” Workshop, ICAPS'03, pp. 46-52, 2004.

[Hoffmann, Kautz, Gomes, Selamn, 2007] J. Hoffmann, H. Kautz, C. Gomes, B. Selman, « SAT Encodings of State-Space Reachability Problems in Numeric Domains », Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI’07, 2007.

[Hu, 2007] Y.Hu, “Temporally-Expressive Planning as Constraint Satisfaction Problems”, ICAPS'07, 2007.

[Huang, Selman, Kautz, 1999] Y.C. Huang, B. Selman, H. Kautz, « Control Knowledge in Planning : Benefits and Tradeoffs », Eleventh Conference on Innovative Applications of Artificial Intelligence, IAAI’99, 1999.

[Iwen, Mali, 2002] M. Iwen, A. D. Mali, « DSatz: A Directional SAT Solver for Planning », Proceedings of IEEE International Conference on Tools with Artificial Intelligence, pages 199-208, ICTAI’02, 2002.

[Kambhampati, 1997] S. Kambhampati, « Refinement Planning as a Unifying Framework for Plan Synthesis », AI Magazine, 18, pp. 67-97, 1997.

[Kambhampati, 2000] S. Kambhampati. "Planning graph as a (dynamic) CSP : Exploiting EBL, DDB and other CSP techniques in Graphplan", JAIR, vol. 12, pp. 1–34, 2000.

[Kautz, 2004] H. Kautz, « SATPLAN’04: Planning as Satisfiability », IPC, 2004.

[Kautz, 2006] H. Kautz, « Deconstructing Planning as Satisfiability », Proceedings of the Twenty-first National Conference on Artificial Intelligence, AAAI’06, 2006.

[Kautz, McAllester, Selman, 1996] H. Kautz, D. McAllester, B. Selman, « Encoding Plans in Propositional Logic », 5th International Conference on Principles of Knowledge Representation and Reasoning, pages 374-384, KR’96, 1996.

[Kautz, Selman, 1992] H. Kautz, B. Selman, « Planning as Satisfiability », Proceedings of the 10th European Conference on Artificial Intelligence, ECAI’92, pages 359-363, 1992.

[Kautz, Selman, 1996] H. Kautz, B. Selman, « Pushing the Envelope: Planning, Propositional Logic and Stochastic Search », Proceedings of the 13th National Conference on Artificial Intelligence, pages 1194-1201, AAAI’96, 1996.

[Kautz, Selman, 1998a] H. Kautz, B. Selman, « BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving », Workshop on Planning as Combinatorial Search, AIPS’98, 1998.

[Kautz, Selman, 1998b] H. Kautz, B. Selman, « The Role of Domain-Specific Knowledge in the Planning as Satisfiability Framework », Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, AIPS’98, 1998.

[Kautz, Selman, 1999] H. Kautz, B. Selman, « Unifying SAT-based and Graph-based Planning », Proceedings of the 16th International Joint Conference on Artificial Intelligence, pages 318-325, IJCAI’99, 1999.

[Kautz, Selman, Hoffmann, 2006] H. Kautz, B. Selman, J. Hoffmann « SatPlan: Planning as Satisfiability, Abstracts of the 5th International Planning Competition, 2006.

[Kripke, 1959] S. Kripke, "A Completeness Theorem in Modal Logic", Journal of Symbolic Logic 24(1), pp. 1-14, (1959).

[Kripke, 1963] S. Kripke, "Semantical Considerations for Modal Logics", Acta Philosophica Fennica 16, pp. 83-94, 1963.

[Kumar, 2005] T.K. Satish Kumar, "On the tractability of restricted disjunctive temporal problems", In Proc. of ICAPS'05, pp 110–119, Monterey, CA, 2005.

[Kvarnström, Doherty, Haslum, 2000] J. Kvarnström, P. Doherty, P. Haslum, "Extending TALplanner with concurrency and resources", in: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI- 2000), Berlin, Germany, 2000, pp. 501- 505.

[Liu, Jiang, 2008] Y. Liu, Y. Jiang, "Topology-based Variable Ordering Strategy for Solving Disjunctive Temporal Problems", TIME 2008, pp. 129-136.

[Long, Fox, 2003] D.Long, M.Fox, "Exploiting a graphplan framework in temporal planning", ICAPS, 2003.

[Mali, 1999a] A. D. Mali, « Plan Merging and Plan Reuse As Satisfiability », Proceedings of the 5th European Conference on Planning, pages 84-96, ECP'99, 1999.

[Mali, 1999b] A. D. Mali, « Hierarchical Task Network Planning as Satisfiability », Proceedings of the 5th European Conference on Planning, pages 122-134, ECP'99, 1999.

[Mali, 2000a] A. D. Mali, « Enhancing HTN Planning As Satisfiability », Proceedings of the International Conference on Artificial Intelligence and Soft Computing, pages 325-333, ASC'00, 2000.

[Mali, 2000b] A. D. Mali, « On the Hybrid Propositional Plan », Proceedings of the International Conference on Artificial Intelligence and Soft computing, pages 334-342, ASC'00, 2000.

[Mali, Kambhampati, 1998a] A. Mali, S. Kambhampati, « Refinement based planning as satisfiability », AIPS Workshop on Planning as Combinatorial Search, Fourth International Conference on Artificial Intelligence Planning Systems, AIPS'98, 1998.

[Mali, Kambhampati, 1998b] A. Mali, S. Kambhampati, « Frugal propositional encodings for planning », AIPS Workshop on Planning as Combinatorial Search, Fourth International Conference on Artificial Intelligence Planning Systems, AIPS'98, 1998.

[Mali, Kambhampati, 1998c] A. Mali, S. Kambhampati, « Encoding HTN planning in propositional logic », Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, AIPS'98, 1998.

[Mali, Kambhampati, 1999] A. Mali, S. Kambhampati, « On the Utility of Plan-Space (Causal) Encodings », Proceedings of the 16th National Conference on Artificial Intelligence, pages 557-563, AAAI'99, 1999.

[Maris, Régnier, 2008.a] F. Maris, P. Régnier, "TLP-GP: Solving Temporally-Expressive Planning Problems", TIME 2008, 137-144.

[Maris, Régnier, 2008.b] F. Maris, P. Régnier, "TLP-GP: New Results on Temporally-Expressive Planning Benchmarks", ICTAI (1) 2008, 507-514.

[Maris, Régnier, Rodriguez, Vidal, 2004] F. Maris, P. Régnier, X. Rodriguez, V. Vidal, « Planification BLACKBOX : Amélioration des codages », Rapport IRIT, n° 2004-13-R, septembre 2004.

[Maris, Régnier, Vidal, 2002] F. Maris, P. Régnier, V. Vidal, « Planification SAT : amélioration des codages, automatisation de la traduction et étude comparative », Rapport IRIT, n° 2002-39-R, novembre 2002.

[Maris, Régnier, Vidal, 2004] F. Maris, P. Régnier, V. Vidal, « Planification SAT : Amélioration des codages et traduction automatique », Congrès Reconnaissance des formes et IA, RFIA'04, 2004.

[Mattmüller, Rintanen, 2007] R. Mattmüller, J. Rintanen, « Planning for temporally extended goals as propositional satisfiability », Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07, pages 1966-1971, AAAI Press, 2007.

[McCarthy, Hayes 1969] J. McCarthy, P.J. Hayes, "Some philosophical problems from the standpoint of AI", Machine Intelligence 4 (1969).

[McDermott, 1982] D.V. McDermott, "A Temporal Logic for Reasoning About Processes and Plans", Cognitive Science 6, pp. 101-155, 1982.

[McDermott, 1998] D.V. McDermott, "PDDL, The Planning Domain Definition Language", Technical Report, Yale University, (http://cs-www.cs.yale.edu/homes/dvm/), 1998.

[Montanari, 1974] U. Montanari, "Networks of constraints: Fundamental properties and applications to picture processing", Information Science 7, 95-132, (1974).

[Pearl, 1985] J. Pearl, « Heuristics : Intelligent Search Strategies for Computer Problem Solving », Addison-Wesley Editeur, 1985.

[Pednault 1989] E.P.D. Pednault, "ADL: Exploring the middle ground between STRIPS and the situation calculus", in: Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR-89), Toronto, Canada, 1989, pp. 324-332.

[Penberthy, Weld, 1992] J.S. Penberthy, D.S. Weld, "UCPOP: A sound, complete, partial order planner for ADL", in: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92), Cambridge, MA, USA, 1992, pp. 103-114.

[Penberty, Weld, 1994] J.S. Penberthy, D.S. Weld, "Temporal Planning with Continuous Change", AAAI 1994, pp. 1010-1015.

[Prior, 1957] A.N. Prior, "Time and Modality", Oxford, Clarendon Press, 1957.

[Régnier, Vidal, 2004] P. Régnier, V. Vidal, « Algorithmique de la planification en IA », éditions Cépaduès, 2004.

[Reichgelt, Shadbolt, 1990] H. Reichgelt, N. Shadbolt, "A Specification Tool for Planning Systems", ECAI 1990, 541-546.

[Rintanen 1998] J. Rintanen, "A planning algorithm not based on directionnal search", in: Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98), Trento, Italy, 1998, pp. 617-624.

[Rintanen, 2003] J. Rintanen, « Symmetry reduction for SAT representations of transition systems », Proceedings of the 13th International Conference on Automated Planning and Scheduling, pages 32-40, ICAPS'03, AAAI Press, 2003.

[Rintanen, 2004] J. Rintanen, « Evaluation strategies for planning as satisfiability », Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'04, pages 682-687, IOS Press, 2004.

[Rintanen, 2007.a] J.Rintanen, "Complexity of Concurrent Temporal Planning", ICAPS, 2007.

[Rintanen, 2007.b] J. Rintanen, « Asymptotically optimal encodings of conformant planning in QBF », Proceedings of the 22nd AAAI Conference on Artificial Intelligence, AAAI'07, 2007.

[Rintanen, Heljanko, Niemelä, 2004] J. Rintanen, K. Heljanko, I. Niemelä, « Parallel encodings of classical planning as satisfiability », Proceedings of the 9th European Conference on Logics in Artificial Intelligence, JELIA'04, Lecture Notes in Computer Science 3229, pages 307-319, Springer-Verlag, 2004.

[Rintanen, Heljanko, Niemelä, 2006] J. Rintanen, K. Heljanko, I. Niemelä, « Planning as satisfiability: parallel plans and algorithms for plan search », Artificial Intelligence, 170(12-13), pages 1031-1080, 2006.

[Robinson, Gretton, Pham, Sattar, 2008] N. Robinson, C. Gretton, D.N. Pham, A. Sattar, "A Compact and Efficient SAT Encoding for Planning", ICAPS 2008, 296-303.

[Rutten, Hertzberg, 1993] E. Rutten, J. Hertzberg, "Temporal Planner = Nonlinear Planner + Time Map Manager", AI Communications 6(1), pp. 18-26 (1993).

[Schwartz, Pollack, 2004] P. Schwartz, M. E. Pollack, "Planning with Disjunctive Temporal Constraints," ICAPS'04 Workshop on Integrating Planning into Scheduling, 2004.

[Shin, Davis, 2004], J. Shin, Ernest Davis, « Continuous Time in a SAT-Based Planner », Proceedings of the the 19th National Conference on Artificial Intelligence, pages 531-536, AAAI'04, 2004.

[Shin, Davis, 2005], J. Shin, Ernest Davis, « Processes and Continuous Change in a SAT-based Planner », Artificial Intelligence, vol. 166, pages 194-253, 2005.

[Smith, 2003] D.E. Smith, "The Case for Durative Actions: A Commentary on PDDL2.1", Journal of Artificial Intelligence Research (JAIR) 20, 149-154, (2003).

[Smith, Weld, 1999] D.E. Smith, D.S. Weld, "Temporal Planning with Mutual Exclusion Reasoning", IJCAI 1999, 326-337.

[Stergiou, Koubarakis, 2000] K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. Artificial Intelligence, 120(1):81–117, 2000.

[Tsamardinos, Pollack, 2003] I. Tsamardinos and M. E. Pollack, "Efficient solution techniques for disjunctive temporal reasoning problems". Artificial Intelligence, 151(1–2):43–89, 2003.

[Vere, 1983] S.Vere, « Planning in Time: Windows and Durations for Activities and Goals », IEEE Trans. on Pattern Analysis and Machine Intelligence, 5, pp. 246-267, 1983.

[Vidal, 2001] V. Vidal, « Recherche dans les graphes de planification, satisfiabilité et stratégies de mondre engagement », Thèse de Doctorat de l'Université Paul Sabatier, IRIT, UPS, Toulouse, 2001.

[Vidal, 2002] V. Vidal, « Le moindre engagement dans une approche de la planification basée sur la procédure de Davis et Putnam », Journées Nationales sur la résolution de problèmes NP-Complets, JNPC'02, pp. 239-25, 2002.

[Vidal, Geffner, 2006] V.Vidal, H.Geffner, "Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming", AI, 170 (3), 2006.

[Wehrle, Rintanen, 2007] M. Wehrle, J. Rintanen, "Planning as Satisfiability with Relaxed ∃-Step Plans", Australian Conference on Artificial Intelligence 2007, 244-253.

[Wolfman, Weld, 1999] S. Wolfman, D. Weld, « The LPSAT Engine and its Application to Resource Planning », Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI'99, 1999.

[Wolfman, Weld, 2000] S. A. Wolfman, D. Weld, « Combining Linear Programming and Satisfiability Solving for Resource Planning », Knowledge Engineering Review, 15:1, 2000.

[Xing, Chen, Zhang, 2006] Z. Xing, Y. Chen, W. Zhang, « MaxPlan: Optimal Planning by Decomposed Satisfiability and Backward Reduction », Fifth IPC, International Conference on Automated Planning and Scheduling, ICAPS'06, pp. 53-56, 2006.

[Younes, Simmons, 2003] H.L.S.Younes, R.G.Simmons, "VHPOP: Versatile Heuristic Partial Order Planner", Journal of AI Research, 20, 2003.

[Zhang 1997] H. Zhang, "SATO: An efficient propositional prover", in: Proceedings of the 14th International Conference on Automated Deduction (ICAD-97), 1997, pp. 272-275.

[Zhao, Chen, Zhang, 2006] X. Zhao, Y. Chen, W. Zhang, « Optimal Planning by Maximum Satisfiability and Accumulative Learning », International Conference on Automated Planning and Scheduling, ICAPS'06, pp. 442-447, 2006.

Annexe 1 : Codages SAT

1. $$\left(\bigwedge_{f\in I} f(0)\right)\wedge\left(\bigwedge_{f\in(F-I)} \neg f(0)\right)\wedge\left(\bigwedge_{f\in G} f(k)\right)$$

2. $$\bigwedge_{i\in[1,k]}\bigwedge_{a\in O}\left(a(i)\Rightarrow\left(\bigwedge_{f\in \mathrm{Pre}c(a)} f(i-1)\right)\wedge\left(\bigwedge_{f\in Add(a)} f(i)\right)\wedge\left(\bigwedge_{f\in Del(a)} \neg f(i)\right)\right)$$

3. $$\bigwedge_{i\in[1,k]}\bigwedge_{f\in((I\cup F_a)\cap F_d)}\left(\left(f(i-1)\wedge\neg f(i)\right)\Rightarrow\bigvee_{a\in O\,/\,f\in Del(a)} a(i)\right)$$

$$\bigwedge_{i\in[1,k]}\bigwedge_{f\in(((F-I)\cup F_d)\cap F_a)}\left(\left(\neg f(i-1)\wedge f(i)\right)\Rightarrow\bigvee_{a\in O\,/\,f\in Add(a)} a(i)\right)$$

4. $$\bigwedge_{i\in[1,k]}\ \bigwedge_{\substack{(a_m,a_n)\in O^2\,/\,m<n\\ \wedge\left(\left(\mathrm{Pre}c(a_m)\cap Del(a_n)\neq\varnothing\right)\vee\left(Del(a_m)\cap \mathrm{Pre}c(a_n)\neq\varnothing\right)\right)\\ \wedge\left(\left(Add(a_m)\cap Del(a_n)=\varnothing\right)\vee\left(Del(a_m)\cap Add(a_n)=\varnothing\right)\right)}}\left(\neg a_m(i)\vee\neg a_n(i)\right)$$

Codage 1 : V01(1) – Espaces d'états avec frame-axiomes explicatifs

1. $$\left(\bigwedge_{f\in I} f(0)\right)\wedge\left(\bigwedge_{f\in(F-I)} \neg f(0)\right)\wedge\left(\bigwedge_{f\in G} f(k)\right)$$

2. $$\bigwedge_{i\in[1,k]}\bigwedge_{a\in O}\left(a(i)\Rightarrow\left(\bigwedge_{f\in \mathrm{Pre}c(a)} f(i-1)\right)\wedge\left(\bigwedge_{f\in Add(a)} f(i)\right)\wedge\left(\bigwedge_{f\in Del(a)} \neg f(i)\right)\right)$$

$$\bigwedge_{i\in[1,k]}\bigwedge_{f\in(I\cup F_a)}\left(N(f,i)\Rightarrow f(i-1)\wedge f(i)\right)$$

3. $$\bigwedge_{i\in[1,k]}\bigwedge_{f\in(I\cup F_a)}\left(f(i)\Rightarrow N(f,i)\vee\left(\bigvee_{a\in O\,/\,f\in Del(a)} a(i)\right)\right)$$

4. $$\bigwedge_{i\in[1,k]}\ \bigwedge_{\substack{(a_m,a_n)\in O^2\,/\,m<n\\ \wedge\left(\left(\mathrm{Pre}c(a_m)\cap Del(a_n)\neq\varnothing\right)\vee\left(Del(a_m)\cap \mathrm{Pre}c(a_n)\neq\varnothing\right)\right)\\ \wedge\left(\left(Add(a_m)\cap Del(a_n)=\varnothing\right)\vee\left(Del(a_m)\cap Add(a_n)=\varnothing\right)\right)}}\left(\neg a_m(i)\vee\neg a_n(i)\right)$$

Codage 2 : V01(2) – Espaces d'états avec No-ops

1. $\bigwedge_{i\in[1,k]}\left(\bigvee_{a\in O}(p_i=a)\vee(p_i=\phi)\right)$

2. $\bigwedge_{i\in[1,k]}\bigwedge_{a_1\in O}\bigwedge_{a_2\in O/a_1\neq a_2}\left(\neg(p_i=a_1)\vee\neg(p_i=a_2)\right)$

$\bigwedge_{i\in[1,k]}\bigwedge_{a\in O}\left(\neg(p_i=a)\vee\neg(p_i=\phi)\right)$

3. $\left(\bigwedge_{f\in I}Adds(Init,f)\right)\wedge\left(\bigwedge_{f\in(F-I)}\neg Adds(Init,f)\right)\wedge\left(\bigwedge_{f\in G}Needs(Goal,f)\right)$

4. $\bigwedge_{i\in[1,k]}\bigwedge_{a\in O}\left((p_i=a)\Rightarrow\left(\bigwedge_{f\in Add(a)}Adds(p_i,f)\right)\wedge\left(\bigwedge_{f\in Prec(a)}Needs(p_i,f)\right)\wedge\left(\bigwedge_{f\in Del(a)}Dels(p_i,f)\right)\right)$

5. $\bigwedge_{i\in[1,k]}\bigwedge_{f\in F}\left(Adds(p_i,f)\Rightarrow\bigvee_{a\in O/f\in Add(a)}(p_i=a)\right)$

$\bigwedge_{i\in[1,k]}\bigwedge_{f\in F}\left(Dels(p_i,f)\Rightarrow\bigvee_{a\in O/f\in Del(a)}(p_i=a)\right)$

$\bigwedge_{i\in[1,k]}\bigwedge_{f\in F}\left(Needs(p_i,f)\Rightarrow\bigvee_{a\in O/f\in Prec(a)}(p_i=a)\right)$

Codage 3 : MK99 – Espaces de plans, Partie commune

1. $\bigwedge_{i\in[1,k]}\left(\bigvee_{a\in O}(a\in p_i)\right)$

2. $\bigwedge_{i\in[1,k]}\bigwedge_{\substack{(a_m,a_n)\in O^2/m<n\wedge\\((Add(a_m)\cup Prec(a_m))\cap Del(a_n)\neq\varnothing\\\vee(Add(a_n)\cup Prec(a_n))\cap Del(a_m)\neq\varnothing)}}\left(\neg(a_m\in p_i)\vee\neg(a_n\in p_i)\right)$

3. $\left(\bigwedge_{f\in I}Adds(Init,f)\right)\wedge\left(\bigwedge_{f\in(F-I)}\neg Adds(Init,f)\right)\wedge\left(\bigwedge_{f\in G}Needs(Goal,f)\right)$

4. $\bigwedge_{i\in[1,k]}\bigwedge_{f\in F_a}\left(Adds(p_i,f)\Leftrightarrow\bigvee_{a\in O/f\in Add(a)}(a\in p_i)\right)$

$\bigwedge_{i\in[1,k]}\bigwedge_{f\in F_d}\left(Dels(p_i,f)\Leftrightarrow\bigvee_{a\in O/f\in Del(a)}(a\in p_i)\right)$

$\bigwedge_{i\in[1,k]}\bigwedge_{f\in F_p}\left(Needs(p_i,f)\Leftrightarrow\bigvee_{a\in O/f\in Prec(a)}(a\in p_i)\right)$

Codage 4 : V01 – Espaces de plans, Partie commune

1. $$\bigwedge_{i\in[1,k]}\ \bigwedge_{f\in F_p}\left(Needs(p_i,f)\Rightarrow\left(\begin{array}{l}\left[f\in F_a\mapsto\bigvee_{j\in[1,k]/\,j\neq i}Link(p_j,f,p_i)\vdots\bot\right]\\ \vee\left[f\in I\mapsto Link(Init,f,p_i)\vdots\bot\right]\end{array}\right)\right)$$

$$\bigwedge_{f\in G}\left(Needs(Goal,f)\Rightarrow\left(\begin{array}{l}\left[f\in F_a\mapsto\bigvee_{i\in[1,k]}Link(p_i,f,Goal)\vdots\bot\right]\\ \vee\left[f\in I\mapsto Link(Init,f,Goal)\vdots\bot\right]\end{array}\right)\right)$$

2. $$\bigwedge_{i\in[1,k]}\ \bigwedge_{j\in[1,k]/\,j\neq i}\ \bigwedge_{f\in(F_p\cap F_a)}\left(Link(p_i,f,p_j)\Rightarrow Adds(p_i,f)\wedge Needs(p_j,f)\wedge \Pr ed(p_i,p_j)\right)$$

$$\bigwedge_{i\in[1,k]}\ \bigwedge_{f\in(G\cap F_a)}\left(Link(p_i,f,Goal)\Rightarrow Adds(p_i,f)\right)$$

$$\bigwedge_{i\in[1,k]}\ \bigwedge_{f\in(F_p\cap I)}\left(Link(Init,f,p_i)\Rightarrow Needs(p_i,f)\right)$$

3. $$\bigwedge_{i\in[1,k]}\ \bigwedge_{j\in[1,k]/\,j\neq i}\ \bigwedge_{q\in[1,k]/\,q\neq i\wedge q\neq j}\ \bigwedge_{f\in(F_p\cap F_a\cap F_d)}\left(\begin{array}{l}Link(p_i,f,p_j)\wedge Dels(p_q,f)\\ \Rightarrow \Pr ed(p_q,p_i)\vee \Pr ed(p_j,p_q)\end{array}\right)$$

$$\bigwedge_{i\in[1,k]}\ \bigwedge_{j\in[1,k]/\,j\neq i}\ \bigwedge_{f\in(G\cap F_a\cap F_d)}\left(Link(p_i,f,Goal)\wedge Dels(p_j,f)\Rightarrow \Pr ed(p_j,p_i)\right)$$

$$\bigwedge_{i\in[1,k]}\ \bigwedge_{j\in[1,k]/\,j\neq i}\ \bigwedge_{f\in(F_p\cap I\cap F_d)}\left(Link(Init,f,p_i)\wedge Dels(p_j,f)\Rightarrow \Pr ed(p_i,p_j)\right)$$

$$\bigwedge_{i\in[1,k]}\ \bigwedge_{f\in(G\cap I\cap F_d)}\left(Link(Init,f,Goal)\wedge Dels(p_i,f)\Rightarrow\bot\right)$$

4. $$\bigwedge_{i\in[1,k]}\ \bigwedge_{j\in[1,k]/\,j\neq i}\ \bigwedge_{s\in[1,k]/\,s\neq i\wedge s\neq j}\left(\Pr ed(p_i,p_j)\wedge \Pr ed(p_j,p_s)\Rightarrow \Pr ed(p_i,p_s)\right)$$

$$\bigwedge_{i\in[1,k]}\ \bigwedge_{j\in[i+1,k]}\left(\neg \Pr ed(p_i,p_j)\vee\neg \Pr ed(p_j,p_i)\right)$$

Codage 5 : V01(3) – Espaces de plans par liens causaux, protection d'intervalles et ordre partiel

$$1.\quad \bigwedge_{i\in[1,k]} \bigwedge_{f\in F_p} \left(Needs(p_i, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{j\in[1,i-1]} Link(p_j, f, p_i) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Link(Init, f, p_i) \vdots \bot \right] \end{array} \right) \right)$$

$$\bigwedge_{f\in G} \left(Needs(Goal, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{i\in[1,k]} Link(p_i, f, Goal) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Link(Init, f, Goal) \vdots \bot \right] \end{array} \right) \right)$$

$$2.\quad \bigwedge_{i\in[1,k-1]} \bigwedge_{j\in[i+1,k]/\, j\neq i} \bigwedge_{f\in(F_p\cap F_a)} \left(Link(p_i, f, p_j) \Rightarrow \left(\begin{array}{l} Adds(p_i, f) \wedge Needs(p_j, f) \\ \wedge \left[f \in F_d \mapsto \bigwedge_{q\in[i+1, j-1]} \neg Dels(p_q, f) \vdots \mathrm{T} \right] \end{array} \right) \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(G\cap F_a)} \left(Link(p_i, f, Goal) \Rightarrow Adds(p_i, f) \wedge \left[f \in F_d \mapsto \bigwedge_{j\in[i+1,k]} \neg Dels(p_j, f) \vdots \mathrm{T} \right] \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(F_p\cap I)} \left(Link(Init, f, p_i) \Rightarrow Needs(p_i, f) \wedge \left[f \in F_d \mapsto \bigwedge_{j\in[1,i-1]} \neg Dels(p_j, f) \vdots \mathrm{T} \right] \right)$$

$$\bigwedge_{f\in(I\cap G)} \left(Link(Init, f, Goal) \Rightarrow \left[f \in F_d \mapsto \bigwedge_{i\in[1,k]} \neg Dels(p_i, f) \vdots \mathrm{T} \right] \right)$$

Codage 6 : V01(4) – Espaces de plans par liens causaux, protection d'intervalles et étapes contiguës

$$1.\quad \bigwedge_{i\in[1,k]} \bigwedge_{f\in F_p} \left(Needs(p_i, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{j\in[1,i-1]} Adds(p_j, f) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Adds(Init, f) \vdots \bot \right] \end{array} \right) \right)$$

$$\bigwedge_{f\in G} \left(Needs(Goal, f) \Rightarrow \left(\begin{array}{l} \left[f \in F_a \mapsto \bigvee_{i\in[1,k]} Adds(p_i, f) \vdots \bot \right] \\ \vee \left[f \in I \mapsto Adds(Init, f) \vdots \bot \right] \end{array} \right) \right)$$

$$2\quad \bigwedge_{i\in[2,k]} \bigwedge_{j\in[1,i-1]} \bigwedge_{f\in(F_p\cap F_d)} \left(Needs(p_i, f) \wedge Dels(p_j, f) \Rightarrow \left[f \in F_a \mapsto \bigvee_{q\in[j+1,i-1]} Adds(p_q, f) \vdots \bot \right] \right)$$

$$\bigwedge_{i\in[1,k]} \bigwedge_{f\in(G\cap F_d)} \left(Needs(Goal, f) \wedge Dels(p_i, f) \Rightarrow \left[f \in F_a \mapsto \bigvee_{j\in[i+1,k]} Adds(p_j, f) \vdots \bot \right] \right)$$

Codage 7 : V01(5) – Espaces de plans avec "White Knight"

1. $\left(\bigwedge_{n_f \in NoeudsInit(GP)} n_f \right) \wedge \left(\bigwedge_{n_f \in NoeudsBut(GP)} n_f \right)$

2. $\bigwedge_{(n_f, n_a) \in Arcs\,Prec(GP)} (n_a \Rightarrow n_f)$

3. $\bigwedge_{n_f \in (NoeudsF(GP) - NoeudsInit(GP))} \left(n_f \Rightarrow \bigvee_{(n_a, n_f) \in ArcsAdd(GP)} n_a \right)$

4. $\bigwedge_{\{n_a, n_b\} \in MutexA(GP)} (\neg n_a \vee \neg n_b)$

Codage 8 : KS99 – Codage du graphe de planification

1. $\bigwedge_{n_f \in NoeudsBut\ (GP)} \left(\bigvee_{(n_a, n_f) \in ArcsAdd\ (GP)} n_a \right)$

2. $\bigwedge_{(n_f, n_a) \in Arcs\ Prec(GP) / n_f \notin NoeudsInit\ (GP)} \left(n_a \Rightarrow \bigvee_{(n_b, n_f) \in ArcsAdd\ (GP)} n_b \right)$

3. $\bigwedge_{\{n_a, n_b\} \in MutexA\ (GP)} (\neg n_a \vee \neg n_b)$

Codage 9 : V01(6) – Codage du graphe de planification

Annexe 2 : Codage de l'autorisation faible dans les espaces d'états.

On considère le codage dans les espaces d'états avec frame-axiomes explicatifs et autorisation forte (LCS) que l'on va modifier pour prendre en compte la relation d'autorisation faible. Pour un niveau i auquel plusieurs actions peuvent être actives simultanément, on définit la séquence autorisée active comme étant la séquence faiblement autorisée des actions qui font partie du plan-solution à ce niveau. On définit un nouveau prédicat ASeq(i, a_1, a_2) qui représente le fait que les actions a_1 et a_2 font partie de la séquence autorisée active au niveau i (a_1 et a_2 sont actives au niveau i et $a_1 \leqq a_2$). La deuxième règle est modifiée et permet maintenant de rendre faux un fluent retiré par une action faisant partie de la séquence autorisée active uniquement si aucune autre action suivante dans la séquence ne rétablit ce fluent.

$$2.1 \quad \bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \left(a(i) \Rightarrow \left(\bigwedge_{f\in \mathrm{Pre}c(a)} f(i-1) \right) \wedge \left(\bigwedge_{f\in Add(a)} f(i) \right) \right)$$

$$2.2 \quad \bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \bigwedge_{f\in Del(a)} \left(a(i) \wedge \bigwedge_{b\in O/(b\neq a)\wedge(f\in Del(a)\cap Add(b))} \left(\neg ASeq(i,a,b)\right) \Rightarrow \neg f(i) \right)$$

Par rapport au codage avec autorisation forte, la quatrième règle doit également être modifiée pour prendre en compte la séquence autorisée active. Deux actions qui ne s'autorisent pas faiblement l'une ou l'autre ne peuvent être actives au même niveau. Si une action n'autorise pas une autre action, elles ne peuvent pas faire partie de la séquence autorisée active dans cet ordre. Enfin, deux actions autorisées sont actives au niveau i si et seulement si elles font partie de la séquence autorisée active.

$$4.1 \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_m,a_n)\in O^2/m<n \\ \wedge((Add(a_m)\cap Del(a_n)\neq\varnothing)\vee(Del(a_m)\cap \mathrm{Pre}c(a_n)\neq\varnothing)) \\ \wedge((Del(a_m)\cap Add(a_n)\neq\varnothing)\vee(\mathrm{Pre}c(a_m)\cap Del(a_n)\neq\varnothing))}} \left(\neg a_m(i) \vee \neg a_n(i)\right)$$

$$4.2 \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_1,a_2)\in O^2/a_1\neq a_2 \\ (Add(a_1)\cap Del(a_2)\neq\varnothing)\vee(Del(a_1)\cap \mathrm{Pre}c(a_2)\neq\varnothing)}} \left(\neg ASeq(i,a_1,a_2)\right)$$

$$4.3 \quad \bigwedge_{i\in[1,k]} \bigwedge_{\substack{(a_1,a_2)\in O^2/a_1\neq a_2 \\ \wedge((Add(a_1)\cap Del(a_2)=\varnothing)\wedge(Del(a_1)\cap \mathrm{Pre}c(a_2)=\varnothing))}} \left(a_1(i) \wedge a_2(i) \Leftrightarrow ASeq(i,a_1,a_2)\right)$$

Une simplification qui n'utilise pas le prédicat ASeq(i, a_1, a_2) peut également être envisagée. On peut en effet considérer que la règle 4.2 est inutile si l'on restreint la règle 2.2 aux seules actions b autorisées par a mais qui n'autorisent pas a. Dans ce cas, d'après la règle 4.3, ¬ASeq(i,a,b) ⇔ (¬a(i) ∨ ¬b(i)). Dans la règle 2.2, on peut donc remplacer ¬ASeq(i,a,b) par ¬b(i) en simplifiant par a(i). Le prédicat ASeq(i, a_1, a_2) n'étant plus utilisé, on peut supprimer la règle 4.3. Le nouveau codage ne conserve donc que les règles 2.1, 4.1 et la règle 2.2 ainsi modifiée :

$$2.2 \quad \bigwedge_{i\in[1,k]} \bigwedge_{a\in O} \bigwedge_{f\in Del(a)} \left(\left(a(i) \wedge \bigwedge_{\substack{b\in O/(b\neq a)\wedge(f\in Add(b)) \\ \wedge(Add(a)\cap Del(b)=\varnothing)\wedge(Del(a)\cap \mathrm{Pre}c(b)=\varnothing)}} \neg b(i) \right) \Rightarrow \neg f(i) \right)$$

Annexe 3 : Résultats des tests (planification SAT)

Etude expérimentale des codages dans les espaces d'états

Problèmes	Var.	Clauses		Temps CPU (s)		Niv.
		MK99(1)	V01(1)	MK99(1)	V01(1)	
Bworld-3c	132	1 046	842	0,38	0,33	4
Bworld-4c	360	5 803	4 213	3,77	3,15	5
Bworld-5c	810	22 834	15 934	38,10	32,69	6
Bworld-6c	1 374	60 491	41 681	228,71	207,30	6
Gripper-1	62	267	249	0,12	0,12	3
Gripper-2	102	536	476	0,20	0,19	3
Gripper-3	310	1 965	1 671	1,58	1,62	7
Gripper-4	398	2 810	2 306	2,72	2,79	7
Gripper-5	750	5 903	4 693	88,97	107,25	11
Gripper-6	886	7 580	5 864	218,13	164,12	11
Ferry-2	142	655	599	0,38	0,38	7
Ferry-3	298	1 599	1 401	1,63	1,58	11
Ferry-4	510	3 079	2 599	18,78	25,06	15
Ferry-5	778	5 191	4 241	1 999,37	619,93	19
Log-2323	855	4 008	3 648	13,61	19,37	6
Log-4323	1 812	10 050	8 727	113,64	170,25	9
Log-2325	2 117	10 472	9 512	147,27	217,27	10
Log-4325	2 720	15 838	13 705	410,60	669,03	9

Tableau 2 : Codages dans les espaces d'états avec frame-axiomes explicatifs

Problèmes	Var.	Clauses		Temps CPU (s)		Niv.
		MK99(2)	V01(2)	MK99(2)	V01(2)	
BWorld-3c	180	1 094	890	0,39	0,33	4
BWorld-4c	460	5 903	4 313	3,80	3,12	5
BWorld-5c	990	23 014	16 114	39,36	32,72	6
Bworld-6c	1 626	60 743	41 933	232,83	209,26	6
Gripper-1	86	291	273	0,12	0,12	3
Gripper-2	138	572	512	0,20	0,19	3
Gripper-3	422	2 077	1 783	1,67	1,77	7
Gripper-4	538	2 950	2 446	2,93	2,95	7
Gripper-5	1 014	6 167	4 957	118,31	80,90	11
Gripper-6	1 194	7 888	6 172	291,34	169,74	11
Ferry-2	205	718	662	0,41	0,41	7
Ferry-3	430	1 731	1 533	1,93	1,85	11
Ferry-4	735	3 304	2 824	25,14	26,49	15
Ferry-5	1 120	5 533	4 583	1 450,99	2 689,33	19
Log-2323	1 125	4 278	3 918	12,84	18,46	6
Log-4323	2 325	10 563	9 240	106,12	169,45	9
Log-2325	2 787	11 142	10 182	136,07	208,87	10
Log-4325	3 467	16 585	14 452	407,49	654,60	9

Tableau 3 : Codages dans les espaces d'états avec no-ops

Problème	MK99(3)				V01(3)			
	Niv.	Var.	Clauses	Temps CPU (s)	Niv.	Var.	Clauses	Temps CPU (s)
BW-3c	4	414	2 786	0,90	4	410	1 994	0,77
BW-4c	6	1 357	20 771	21,85	5	1 025	8 721	7,75
BW-5c	6	*	*	*	6	2 164	30 599	74,41
Gripper-1	3	183	805	0,21	3	180	586	0,19
Gripper-2	5	584	4 036	1,92	3	275	1 036	0,36
Gripper-3	9	1 999	20 722	-	7	1 314	8 548	-
Gripper-4	11	*	*	*	7	1 641	11 102	-
Gripper-5	15	*	*	*	11	4 176	41 966	-
Gripper-6	17	*	*	*	11	4 863	49 760	-
Ferry-2	7	739	4 871	12,31	7	732	4 402	59,51
Ferry-3	11	2 094	21 358	-	11	2 083	19 939	-
Ferry-4	15	4 489	62 693	-	15	4 474	59 528	-
Ferry-5	19	*	*	*	19	*	*	*
Log-2323	11	*	*	*	6	2 845	15 174	55,80
Log-4323	15	*	*	*	9	*	*	*
Log-2325	27	*	*	*	10	*	*	*
Log-4325	17	*	*	*	9	*	*	*

Tableau 4 : Codages dans les espaces de plans par protection d'intervalles et ordre partiel

Problème	MK99(4)				V01(4)			
	Niv.	Var.	Clauses	Temps CPU (s)	Niv.	Var.	Clauses	Temps CPU (s)
BW-3c	4	330	2 192	0,70	4	326	1 400	0,60
BW-4c	6	1 027	17 316	14,78	5	805	6 771	6,03
BW-5c	6	*	*	*	6	1 684	25 514	59,08
Gripper-1	3	153	645	0,19	3	150	426	0,17
Gripper-2	5	444	2 816	1,17	3	233	802	0,32
Gripper-3	9	1 351	10 834	23,06	7	936	3 984	4,70
Gripper-4	11	2 290	21 863	340,23	7	1 179	5 460	8,23
Gripper-5	15	*	*	*	11	2 746	15 126	1 289,52
Gripper-6	17	*	*	*	11	3 213	18 630	1 911,45
Ferry-2	7	508	2 183	1,14	7	501	1 714	1,01
Ferry-3	11	1 324	7 333	14,12	11	1 313	5 914	15,28
Ferry-4	15	2 704	18 383	588,08	15	2 689	15 218	389,93
Ferry-5	19	*	*	*	19	4 773	32 746	-
Log-2323	11	*	*	*	6	2 140	7 674	33,06
Log-4323	15	*	*	*	9	*	*	*
Log-2325	27	*	*	*	10	*	*	*
Log-4325	17	*	*	*	9	*	*	*

Tableau 5 : Codages dans les espaces de plans par protection d'intervalles et étapes contiguës

Problème	MK99(5)				V01(5)			
	Niv.	Var.	Clauses	Temps CPU (s)	Niv.	Var.	Clauses	Temps CPU (s)
BW-3c	4	234	2 020	0,62	4	230	1 228	0,53
BW-4c	6	677	16 454	12,65	5	563	6 256	5,29
BW-5c	6	*	*	*	6	1 174	24 278	52,85
Gripper-1	3	114	586	0,17	3	111	367	0,16
Gripper-2	5	289	2 481	0,95	3	176	718	0,29
Gripper-3	9	694	8 536	9,05	7	537	2 857	3,11
Gripper-4	11	1 069	16 781	76,73	7	682	4 060	5,44
Gripper-5	15	*	*	*	11	1 283	9 043	1 101,91
Gripper-6	17	*	*	*	11	1 508	11 546	1 456,52
Ferry-2	7	277	1 525	0,80	7	270	1 056	0,71
Ferry-3	11	576	4 198	5,47	11	565	2 779	4,25
Ferry-4	15	979	8 843	87,83	15	964	5 678	35,94
Ferry-5	19	*	*	*	19	1 467	10 041	-
Log-2323	11	*	*	*	6	1 398	5 880	28,76
Log-4323	27	*	*	*	9	2 841	15 111	244,50
Log-2325	17	*	*	*	10	3 462	16 542	340,20
Log-4325	15	*	*	*	9	4 219	23 457	895,22

Tableau 6 : Codages "white knight" dans les espaces de plans

Etude expérimentale des codages par recherche incrémentale

Problème	V01(1)				LCS			
	Niv.	Var.	Clauses	Temps CPU (s)	Niv.	Var.	Clauses	Temps CPU (s)
BW-3c	4	132	842	0,33	3	102	590	0,20
BW-4c	5	360	4 213	3,15	4	292	2 799	1,73
BW-5c	6	810	15 934	32,69	5	680	9 834	17,40
BW-6c	6	1 374	41 681	207,30	3	708	14 024	37,84
BW-7c	*	*	*	*	3	1 106	29 375	170,40
BW-8c	*	*	*	*	3	1 632	56 034	612,42
Gripper-1	3	62	249	0,12	2	44	171	0,08
Gripper-2	3	102	476	0,19	2	72	324	0,12
Gripper-3	7	310	1 671	1,62	4	184	967	0,49
Gripper-4	7	398	2 306	2,79	4	236	1 332	0,79
Gripper-5	11	750	4 693	107,25	6	420	2 579	3,25
Gripper-6	11	886	5 864	164,12	6	496	3 220	5,28
Gripper-7	15	1 382	9 699	-	8	752	5 199	152,84
Gripper-8	15	1 566	11 534	-	8	852	6 180	236,71
Gripper-9	19	2 206	17 073	-	10	1 180	9 019	-
Ferry-2	7	142	599	0,21	4	85	351	0,17
Ferry-3	11	298	1 401	1,58	6	168	777	0,44
Ferry-4	15	510	2 599	25,06	8	279	1 403	1,12
Ferry-5	19	778	4 241	619,93	10	418	2 253	2,67
Ferry-6	23	1 102	6 375	-	12	585	3 351	11,20
Ferry-7	27	1 482	9 049	-	14	780	4 721	51,95
Ferry-8	31	1 918	12 311	-	16	1 003	6 387	237,58
Ferry-9	35	2 410	16 209	-	18	1 254	8 373	-
Log-2323	6	855	3 648	19,37	4	585	2 196	5,98
Log-4323	9	1 812	8 727	170,25	6	1 227	5 172	49,21
Log-2325	10	2 117	9 512	217,27	7	1 502	5 903	69,76
Log-4325	9	2 720	13 705	669,03	6	1 841	7 996	187,27
Log-4326	9	3 174	16 194	1 098,09	6	2 148	9 408	310,87
Log-2525	10	3 655	19 310	1 453,49	7	2 590	10 925	448,83
Hanoi-2	3	98	601	0,20	2	69	471	0,14
Hanoi-3	7	402	3 702	3,51	5	292	2 985	1,77
Hanoi-4	15	1 404	18 058	237,18	10	944	12 928	41,91

Tableau 7 : Codages V01(1) et LCS dans les espaces d'états (étude par recherche incrémentale)

Problème	V01(5)				LCP			
	Niv.	Var.	Clauses	Temps CPU (s)	Niv.	Var.	Clauses	Temps CPU (s)
BW-3c	4	230	1 228	0,53	3	176	709	0,29
BW-4c	5	563	6 256	5,29	4	455	3 098	2,62
BW-5c	6	1 174	24 278	52,85	5	984	10 463	24,41
BW-6c	*	*	*	*	3	965	14 425	51,16
BW-7c	*	*	*	*	*	*	*	*
BW-8c	*	*	*	*	*	*	*	*
Gripper-1	3	111	367	0,16	2	77	216	0,10
Gripper-2	3	176	718	0,29	2	122	392	0,16
Gripper-3	7	537	2 857	3,11	4	315	1 210	0,78
Gripper-4	7	682	4 060	5,44	4	400	1 636	1,29
Gripper-5	11	1 283	9 043	1 101,91	6	713	3 268	6,10
Gripper-6	11	1 508	11 546	1 456,52	6	838	4 024	9,21
Gripper-7	15	2 349	20 461	-	8	1 271	6 678	131,50
Gripper-8	15	2 654	24 712	-	8	1 436	7 844	64,36
Gripper-9	19	3 735	38 647	-	10	1 989	11 728	-
Ferry-2	7	270	1 056	0,71	4	159	491	0,24
Ferry-3	11	565	2 779	4,25	6	315	1 128	0,77
Ferry-4	15	964	5 678	35,94	8	523	2 107	2,38
Ferry-5	19	1 467	10 041	-	10	783	3 488	7,09
Ferry-6	23	2 074	16 156	-	12	1 095	5 331	19,04
Ferry-7	27	2 785	24 311	-	14	1 459	7 696	53,76
Ferry-8	31	3 600	34 794	-	16	1 875	10 643	382,04
Ferry-9	35	4 519	47 893	-	18	2 343	14 232	-
Log-2323	6	1 398	5 880	28,76	4	948	2 845	10,43
Log-4323	9	2 841	15 111	244,50	6	1 914	6 738	84,09
Log-2325	10	3 462	16 542	340,20	7	2 445	8 295	128,00
Log-4325	9	4 219	23 457	895,22	6	2 842	10 278	306,01
Log-4326	9	4 908	27 630	1 430,32	6	3 306	12 048	498,98
Log-2525	10	*	*	*	7	4 065	14 647	744,08
Hanoi-2	3	166	918	0,32	2	115	534	0,17
Hanoi-3	7	643	6 183	6,76	5	465	3 348	2,86
Hanoi-4	15	2 128	32 507	492,34	10	1 428	14 542	61,22

Tableau 8 : Codages V01(5) et LCP dans les espaces de plans (étude par recherche incrémentale)

Etude expérimentale des codages sur le dernier niveau

Problème	V01(1)					LCS				
	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)
BW-3c	4	132	842	0,10	0,02	3	102	590	0,07	0,02
BW-4c	5	360	4 213	1,36	0,06	4	292	2 799	0,80	0,05
BW-5c	6	810	15 934	14,23	0,19	5	680	9 834	8,11	0,12
BW-6c	6	1 374	41 681	84,84	5,17	3	708	14 024	22,39	0,18
BW-7c	*	*	*	*	*	3	1 106	29 375	99,43	0,35
BW-8c	*	*	*	*	*	3	1 632	56 034	350,00	0,84
Gripper-1	3	62	249	0,02	0,01	2	44	171	0,01	0,01
Gripper-2	3	102	476	0,06	0,01	2	72	324	0,03	0,02
Gripper-3	7	310	1 671	0,42	0,08	4	184	967	0,17	0,03
Gripper-4	7	398	2 306	0,79	0,12	4	236	1 332	0,32	0,03
Gripper-5	11	750	4 693	2,77	0,35	6	420	2 579	0,95	0,15
Gripper-6	11	886	5 864	4,28	0,86	6	496	3 220	1,46	0,43
Gripper-7	15	1 382	9 699	11,29	670,47	8	752	5 199	3,39	51,56
Gripper-8	15	1 566	11 534	*	*	8	852	6 180	4,81	6,68
Gripper-9	19	2 206	17 073	*	*	10	1 180	9 019	10,15	404,19
Gripper-10	*	*	*	*	*	10	1 304	10 404	13,30	18,48
Gripper-11	*	*	*	*	*	12	1 704	14 231	*	*
Gripper-12	*	*	*	*	*	12	1 852	16 084	28,44	43,06
Gripper-13	*	*	*	*	*	14	*	*	*	*
Gripper-14	*	*	*	*	*	14	2 496	23 412	*	*
Ferry-2	7	142	599	0,06	0,02	4	85	351	0,03	0,02
Ferry-3	11	298	1 401	0,25	0,09	6	168	777	0,09	0,02
Ferry-4	15	510	2 599	0,79	6,75	8	279	1 403	0,25	0,05
Ferry-5	19	778	4 241	1,94	208,74	10	418	2 253	0,60	0,09
Ferry-6	23	1 102	6 375	*	*	12	585	3 351	1,26	3,46
Ferry-7	27	1 482	9 049	*	*	14	780	4 721	2,37	6,37
Ferry-8	31	1 918	12 311	*	*	16	1 003	6 387	4,27	20,17
Ferry-9	35	2 410	16 209	*	*	18	1 254	8 373	7,00	103,64
Ferry-10	*	*	*	*	*	20	1 533	10 703	*	*
Log-2323	6	855	3 648	5,99	0,07	4	585	2 196	2,51	0,04
Log-4323	9	1 812	8 727	38,18	0,41	6	1 227	5 172	14,94	0,09
Log-2325	10	2 117	9 512	43,82	0,18	7	1 502	5 903	18,82	0,09
Log-4325	9	2 720	13 705	144,57	0,25	6	1 841	7 996	57,09	0,12
Log-4326	9	3 174	16 194	232,54	0,30	6	2 148	9 408	92,68	0,15
Log-2525	10	3 655	19 310	283,72	0,50	7	2 590	10 925	120,02	0,20
Hanoi-2	3	98	601	0,06	0,01	2	69	471	0,05	0,02
Hanoi-3	7	402	3 702	1,22	0,09	5	292	2 985	0,74	0,05
Hanoi-4	15	1 404	18 058	23,02	39,47	10	944	12 928	10,93	3,45

Tableau 9 : Codages V01(1) et LCS dans les espaces d'états (étude sur le dernier niveau)

Problème	V01(5)					LCP				
	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)
BW-3c	4	230	1 228	0,19	0,03	3	176	709	0,11	0,02
BW-4c	5	563	6 256	2,00	0,08	4	455	3 098	1,10	0,05
BW-5c	6	1 174	24 278	17,71	0,30	5	984	10 463	9,09	0,14
BW-6c	6	*	*	*	*	3	965	14 425	25,91	0,37
BW-7c	*	*	*	*	*	*	*	*	*	*
BW-8c	*	*	*	*	*	*	*	*	*	*
Gripper-1	3	111	367	0,04	0,02	2	77	216	0,02	0,01
Gripper-2	3	176	718	0,11	0,02	2	122	392	0,06	0,02
Gripper-3	7	537	2 857	0,80	0,05	4	315	1 210	0,29	0,02
Gripper-4	7	682	4 060	1,42	0,14	4	400	1 636	0,51	0,04
Gripper-5	11	1 283	9 043	5,71	34,84	6	713	3 268	1,51	1,13
Gripper-6	11	1 508	11 546	8,48	36,48	6	838	4 024	2,61	1,75
Gripper-7	15	2 349	20 461	*	*	8	1 271	6 678	5,31	2,94
Gripper-8	15	2 654	24 712	*	*	8	1 436	7 844	7,52	5,92
Gripper-9	19	3 735	38 647	*	*	10	1 989	11 728	15,64	275,26
Gripper-10	19	*	*	*	*	10	2 194	13 384	*	*
Gripper-11	*	*	*	*	*	12	*	*	*	*
Gripper-12	*	*	*	*	*	12	3 112	20 932	*	*
Gripper-13	*	*	*	*	*	14	*	*	*	*
Gripper-14	*	*	*	*	*	14	*	*	*	*
Ferry-2	7	270	1 056	0,17	0,03	4	159	491	0,06	0,02
Ferry-3	11	565	2 779	0,75	0,19	6	315	1 128	0,21	0,03
Ferry-4	15	964	5 678	2,77	1,58	8	523	2 107	0,56	0,10
Ferry-5	19	1 467	10 041	8,63	6,11	10	783	3 488	1,36	0,52
Ferry-6	23	2 074	16 156	*	*	12	1 095	5 331	2,95	0,85
Ferry-7	27	2 785	24 311	*	*	14	1 459	7 696	6,08	2,39
Ferry-8	31	3 600	34 794	*	*	16	1 875	10 643	12,02	4,79
Ferry-9	35	4 519	47 893	*	*	18	2 343	14 232	21,61	36,83
Ferry-10	*	*	*	*	*	20	2 863	18 523	*	*
Log-2323	6	1 398	5 880	8,72	0,10	4	948	2 845	4,19	0,05
Log-4323	9	2 841	15 111	52,35	0,36	6	1 914	6 738	24,08	0,16
Log-2325	10	3 462	16 542	67,90	0,80	7	2 445	8 295	32,25	0,18
Log-4325	9	4 219	23 457	181,56	0,59	6	2 842	10 278	90,96	0,23
Log-4326	9	4 908	27 630	295,79	0,76	6	3 306	12 048	150,01	0,45
Log-2525	10	5 760	32 650	374,02	6,29	7	4 065	14 647	189,25	0,45
Hanoi-2	3	166	918	0,13	0,02	2	115	534	0,07	0,01
Hanoi-3	7	643	6 183	1,97	0,12	5	465	3 348	0,98	0,07
Hanoi-4	15	2 128	32 507	35,58	27,78	10	1 428	14 542	11,51	4,71

Tableau 10 : Codages V01(5) et LCP dans les espaces de plans (étude sur le dernier niveau)

Problème	KS99					V01(6)				
	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)
BW-3c	4	64	223	0,16	0,07	4	36	190	0,24	0,05
BW-5c	6	285	4 172	2,18	0,14	6	193	4 072	6,77	0,15
BW-6c	6	896	46 858	143,57	0,53	6	711	46 642	650,00	0,56
Gripper-1	3	30	63	0,02	0,06	3	15	37	0,01	0,08
Gripper-2	3	46	122	0,04	0,07	3	24	82	0,10	0,06
Gripper-3	7	288	2 288	2,29	0,17	7	195	2 170	6,39	0,17
Gripper-4	7	366	3 508	4,84	0,30	7	250	3 360	16,30	0,32
Gripper-5	11	804	10 234	82,42	1,28	11	569	9 960	271,40	0,44
Gripper-6	11	946	13 770	145,10	1,96	11	672	13 450	465,32	1,99
Gripper-7	15	1 576	27 004	748,71	26,15	15	1 135	26 510	2 368,42	11,78
Gripper-8	15	1 782	33 912	1 100,30	396,10	15	1 286	33 356	3 450,00	74,20
Ferry-2	7	141	611	0,15	0,07	7	89	548	0,35	0,09
Ferry-3	11	342	2 060	3,65	0,15	11	225	1 928	10,38	0,12
Ferry-4	15	623	4 657	33,02	0,60	15	417	4 432	93,95	0,50
Ferry-5	19	984	8 702	158,37	0,40	19	665	8 360	441,20	0,40
Ferry-6	23	1 425	14 495	520,52	5,33	23	969	14 012	1 441,00	71,00
Ferry-7	27	1 946	22 336	1 405,39	514,33	27	1 329	21 688	3 787,83	140,69
Hanoi-2	3	47	152	0,05	0,06	3	26	123	0,10	0,05
Hanoi-3	7	305	3 744	2,78	0,20	7	214	3 644	9,19	0,20
Hanoi-4	15	-	-	-	-	15	-	-	-	-
Log-2323	6	225	769	0,73	0,10	6	145	672	1,86	0,14
Log-4323	9	960	8 738	148,84	0,20	9	688	8 435	401,70	0,20
Log-2325	10	1 108	10 570	234,89	0,30	10	785	10 224	652,33	0,30
Log-2525	10	1 361	17 268	457,00	0,40	10	1 001	16 881	1 244,85	0,50
Log-4326	9	1 635	22 481	815,30	0,60	9	1 199	22 001	2 357,90	0,60
Log-4325	9	1 622	23 800	797,57	0,60	9	1 201	23 344	2 362,47	0,60

Tableau 11 : Codages KS99 et V01(6) du graphe de planification avec relation d'indépendance

Problème	LCG-KS99					LCG-V01				
	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)	Niv.	Var.	Clauses	Temps Trad. (s)	Temps CPU (s)
BW-3c	3	37	82	0,03	0,05	3	18	58	0,03	0,07
BW-5c	5	163	1 051	0,26	0,08	5	100	980	0,57	0,10
BW-6c	3	165	1 780	0,30	0,10	3	105	1 690	0,63	0,10
Gripper-1	2	13	22	0,01	0,07	2	5	7	0,01	0,04
Gripper-2	2	21	43	0,01	0,05	2	9	18	0,03	0,07
Gripper-3	4	132	785	0,13	0,10	4	84	712	0,30	0,09
Gripper-4	4	168	1 204	0,30	0,10	4	108	1 112	0,78	0,10
Gripper-5	6	384	4 259	6,28	0,50	6	264	4 100	19,28	0,50
Gripper-6	6	454	5 746	10,95	0,19	6	312	5 560	35,38	0,20
Gripper-7	8	764	11 933	71,57	0,53	8	540	11 656	231,94	0,54
Gripper-8	8	864	15 016	107,42	0,59	8	612	14 704	348,73	0,51
Ferry-2	4	65	213	0,20	0,00	4	38	175	0,20	0,00
Ferry-3	6	164	859	0,26	0,00	6	104	784	0,56	0,10
Ferry-4	8	303	2 047	2,38	0,10	8	198	1 923	6,90	0,00
Ferry-5	10	482	3 931	13,22	0,20	10	320	3 746	38,15	0,20
Ferry-6	12	701	6 661	50,95	0,30	12	470	6 403	143,22	0,30
Ferry-7	14	960	10 387	146,76	0,15	14	648	10 044	411,25	0,16
Ferry-8	16	1 259	15 259	355,68	1,39	16	854	14 819	983,75	0,76
Hanoi-2	2	21	38	0,01	0,07	2	9	19	0,03	0,08
Hanoi-3	5	164	1 320	0,27	0,10	5	107	1 254	0,69	0,10
Hanoi-4	10	750	15 610	70,23	0,90	10	564	15 414	281,13	0,12
Hanoi-5	-	-	-	-	-	-	-	-	-	-
Log-2323	4	109	275	0,40	0,00	4	64	215	0,90	0,00
Log-4323	6	509	3 704	16,54	0,10	6	354	3 519	43,92	0,10
Log-2325	7	614	4 386	31,79	0,10	7	422	4 171	87,63	0,20
Log-2525	7	755	7 757	65,68	0,20	7	543	7 518	179,20	0,20
Log-4326	6	800	7 310	79,60	0,20	6	562	7 039	222,33	0,20
Log-4325	6	888	10 723	113,60	0,30	6	643	10 444	340,35	0,30

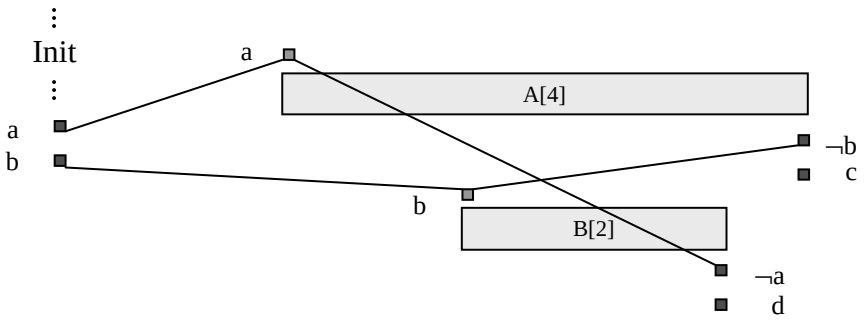
Tableau 12 : Codages LCG-KS99 et LCG-V01 du graphe de planification avec relation d'autorisation

Annexe 4 : Théorème de Cushing, Kambhampati, Mausam et Weld

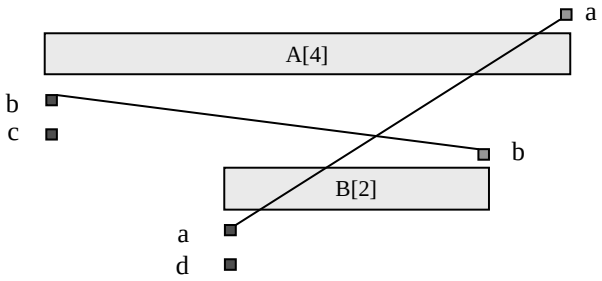
Théorème [Cushing et al., 2007.a] :
Un sous-langage de PDDL2.1/3 est temporellement simple si et seulement si il interdit les trous temporels.

Preuve :
Montrons qu'interdire les trous temporels est nécessaire pour q'un langage soit temporellement-simple.
Montrons tout d'abord que les langages L_e^s, L_s^e et $L_{s,e}$ sont tous temporellement-expressifs :

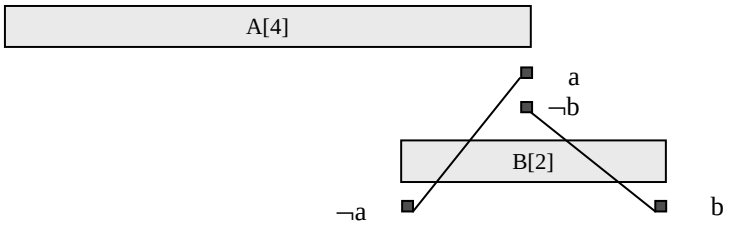
- Le problème suivant comportant deux actions A et B et dont l'état initial représenté par l'action factice instantanée Init est {a, b} montre que L_e^s est temporellement expressif puisqu'il n'existe pas de plan séquentiel menant au but {c, d} :



- Le problème suivant comportant deux actions A et B montre que L_s^e est temporellement expressif puisqu'il n'existe pas de plan séquentiel menant au but {c, d} :



- Enfin, ce dernier problème comportant deux actions A et B montre que $L_{s,e}$ est temporellement expressif puisqu'il n'existe pas de plan séquentiel menant au but {a, b} :



Les langages autorisant un trous temporel entre une condition-avant et un effet, un trou temporel entre une condition-après et un effet ou un trou temporel entre effets sont des super-langages de L^s_e, L^e_s ou $L_{s,e}$ qui sont tous, comme nous venons de le monter, temporellement expressifs. Donc, les langages temporellement simples exigent que, pour chaque action, toutes les conditions-avant soient vérifiées juste avant que tout effet soit produit, toutes les conditions-après soient vérifiées juste après que tout effet soit produit, et que tous les effets soient produits en même temps. C'est-à-dire qu'un langage temporellement-simple doit interdire les trous temporels.

Montrons maintenant que n'importe quel langage interdisant les trous temporels est temporellement-simple, en démontrant que les solutions compactées à gauche de n'importe quel problème peuvent être réorganisées en solutions séquentielles.

Soit une solution compactée à gauche à un problème dans un langage qui interdit les trous temporels. Considérons l'ordre des points critiques dans la solution compactée à gauche, ainsi que les modèles vérifiés entre ceux-ci, c'est-à-dire, M_0, c_1, M_1, c_2…, M_{n-1}, c_n, M_n, où les c_i sont les points critiques (instants où les effets sont produits) et les M_i sont les modèles. Il est trivial d'insérer un retard arbitraire entre chaque point critique, en augmentant la durée de la période sur laquelle chaque modèle est vérifié, sans altérer les modèles. Par exemple, multiplier chaque période d'exécution d'actions correspondant à un même point critique par la durée maximale d'une action permet d'obtenir une réorganisation séquentielle en préservant la séquence. Pour chaque point critique c_i, toutes les conditions-avant de cette action sont dans M_{i-1} et toutes ses conditions-après sont dans M_i, puisque le plan original est exécutable. Puisque ces modèles ne sont pas altérés dans la réorganisation séquentielle, cette réorganisation est également exécutable et ainsi une solution du problème.

Annexe 5 : définition des mutex généralisées de TGP, CPPLANNER

Définition des mutex permanentes de TGP

- **proposition-proposition :** Deux propositions p et q sont des mutex permanents (emutex) si et seulement si p est la négation de q.

- **action-proposition :** L'action A est emutex avec la proposition p si $\neg p$ est une précondition ou un effet de A, ou si p est un effet de A.

- **action-action :** L'action A est emutex avec l'action B si et seulement si au moins une des assertions suivantes est vérifiée :
  - A ou B détruit une précondition ou un effet de l'autre
  - A et B ont des préconditions qui sont emutex

Définition des mutex conditionnelles de TGP

Les propositions p et q sont cmutex lorsque p est cmutex avec toutes les actions ayant pour effet q et vice versa.

- **proposition-proposition :** Soient p et q deux propositions. Pour chaque action A_i supportant p, Φ_i est la condition sous laquelle A_i est mutex avec q (vraie si emutex, faux si non mutex, et η si A_i est cmutex avec q quand η). Pour chaque action B_i supportant q, Ψ_i est la condition sous laquelle B_i est mutex avec p. Soit $\Phi = (\wedge_i (\Phi_i \vee (A_i] > \lfloor p))) \wedge (\wedge_j (\Psi_j \vee (B_j] > \lfloor q)))$. Si Φ est satisfiable, alors les propositions p et q sont des mutex conditionnelles (cmutex) lorsque Φ est vraie.

L'action A est cmutex avec la proposition p lorsque p est cmutex avec toutes les préconditions de A ou lorsque A est cmutex avec toutes les actions ayant p pour effet.

- **action-proposition :** Soit A une action et p une proposition. Pour chaque précondition Q_i de A, soit Φ_i la condition sous laquelle p est mutex avec Q_i (vraie si emutex, …). Pour chaque action B_j qui peut supporter p, soit Ψ_j la condition sous laquelle A est mutex avec B_j. Soit $\Phi = (\vee_i \Phi_i) \vee (\wedge_j (\Psi_j \vee (B_j] > \lfloor p))) \wedge (\lfloor A < [p)$. Si Φ est satisfiable, alors l'action A et la proposition p sont cmutex lorsque Φ est vraie.

Les actions A et B sont cmutex quand A est cmutex avec toutes les préconditions de B et vice versa.

- **action-action :** Soit A et B deux actions qui ne sont pas emutex. Pour chaque précondition p_i de B, soit Φ_i la condition sous laquelle A est mutex avec p_i. Pour chaque précondition q_j de A, soit Ψ_j la condition sous laquelle B est mutex avec q_j. Soit $\Phi = (\vee_i \Phi_i) \vee (\vee_j \Psi_j)$. Si Φ est satisfiable, alors les actions A et B sont cmutex lorsque Φ est vraie.

Représentation des actions dans CPPLANNER

La représentation d'action de CPPlanner est principalement influencée par le langage PDDL+ [Fox, Long, 2001b] et la représentation du planificateur SAPA [Do, Kambhampati 2001]. Chaque action A, a une durée Dur_A, un temps de départ $Start_A$, et un temps de fin End_A (= $Start_A$ + Dur_A). La durée Dur_A peut être statiquement définie pour un domaine, ou être dynamiquement calculée à l'exécution de l'action. L'action A a un ensemble de conditions $Cond_A$, qui consiste en une conjonction des couples <$Cond_{Ai}$, d_{Ai}> (<condition, durée>). Le d_{Ai} pour un certain $Cond_{Ai}$ signifie que le $Cond_{Ai}$ doit être vrai de l'instant de départ de l'action A jusque à ($Start_A$ + d_{Ai}). De même, l'action A a un ensemble d'effets appelés Eff_A, qui se compose des tuples <Eff_{Aj}, d_{Aj}> (<effet, temps>). Le d_{Aj} pour un certain Eff_{Aj} signifie que l'Eff_{Aj} se produit à l'instant ($Start_A$ + d_{Aj}).

Par conséquent, une action A est présentée comme {Dur_A, $Cond_A$, Eff_A} dans lequel :

- Dur_A : La durée de l'action ($Dur_A > 0$) ;
- $Cond_A$: {<$Cond_{A1}$, d_{A1}>, …, <$Cond_{Ak}$, d_{Ak}>} avec $\forall i \in [1,k] : 0 \leq d_{Ai} \leq Dur_A$;
- Eff_A : {<Eff_{A1}, d_{A1}>, …, <Eff_{Ah}, d_{Ah}>} avec $\forall i \in [1,h] : 0 \leq d_{Ai} \leq Dur_A$.

Définition des mutex de CPPLANNER

- **proposition-proposition :** les propositions p et q sont mutex si (1) elles sont des négations l'une de l'autre ou (2) toutes les actions supports de q sont mutex avec p et vice versa.

- **action-proposition :** l'action A et la proposition p sont mutex si l'une des assertions suivantes est vraie :
  - (p et q sont mutex) ^ ($q \in Cond_A$) ^ (p est vrai à t avec $t \in [Start_A, Start_A + d_{Aq}]$).
  - (p et q sont mutex) ^ ($q \in Eff_A$) ^ (p est vrai à t avec $t \in [Start_A + d_{Aq}, End_A]$).

- **action-action :** les actions A, B sont mutex si l'une des assertions suivantes est vraie :
  - (p et q sont mutex) ^ ($p \in Cond_A$) ^ ($q \in Cond_B$) ^ ($Start_A + d_{Ap} \geq Start_B$)
  - (p et q sont mutex) ^ ($p \in Cond_A$) ^ ($q \in Eff_B$) ^ ($d_{Ap} \geq d_{Bq}$)
  - (p et q sont mutex) ^ ($p \in Eff_A$) ^ ($q \in Eff_B$) ^ ($End_B \geq Start_A + d_{Ap}$)

Annexe 6 : Exemple de codage SMT (TLP-GP-2)

Codage TLP-GP-2 – étape 1 (18 var. propositionnelles, 8 var. réelles, 89 clauses) :

1. $A_I \wedge A_G$

2. $B_2 \Rightarrow \mathrm{Link}(A_1,a,B_2)$
$B_3 \Rightarrow \mathrm{Link}(A_1,a,B_3) \vee \mathrm{Link}(A_2,a,B_3)$
$C_3 \Rightarrow \mathrm{Link}(B_2,c,C_3)$
$A_G \Rightarrow \mathrm{Link}(A_1,b,A_G) \vee \mathrm{Link}(A_2,b,A_G) \vee \mathrm{Link}(A_3,b,A_G)$
$A_G \Rightarrow \mathrm{Link}(B_2,d,A_G) \vee \mathrm{Link}(B_3,d,A_G)$
$A_G \Rightarrow \mathrm{Link}(C_3,e,A_G)$

3. $\mathrm{Link}(A_1,a,B_2) \Rightarrow (A_1 \wedge B_2 \wedge (\tau(A_1\rightarrow a) \leq \tau(a\rightarrow B_2)))$
$\mathrm{Link}(A_1,a,B_3) \Rightarrow (A_1 \wedge B_3 \wedge (\tau(A_1\rightarrow a) \leq \tau(a\rightarrow B_3)))$
$\mathrm{Link}(A_2,a,B_3) \Rightarrow (A_2 \wedge B_3 \wedge (\tau(A_2\rightarrow a) \leq \tau(a\rightarrow B_3)))$
$\mathrm{Link}(B_2,c,C_3) \Rightarrow (B_2 \wedge C_3 \wedge (\tau(B_2\rightarrow c) \leq \tau(c\rightarrow C_3)))$
$\mathrm{Link}(A_1,b,A_G) \Rightarrow (A_1 \wedge A_G \wedge (\tau(A_1\rightarrow b) \leq \tau(b\rightarrow A_G)))$
$\mathrm{Link}(A_2,b,A_G) \Rightarrow (A_2 \wedge A_G \wedge (\tau(A_2\rightarrow b) \leq \tau(b\rightarrow A_G)))$
$\mathrm{Link}(A_3,b,A_G) \Rightarrow (A_3 \wedge A_G \wedge (\tau(A_3\rightarrow b) \leq \tau(b\rightarrow A_G)))$
$\mathrm{Link}(B_2,d,A_G) \Rightarrow (B_2 \wedge A_G \wedge (\tau(B_2\rightarrow d) \leq \tau(d\rightarrow A_G)))$
$\mathrm{Link}(B_3,d,A_G) \Rightarrow (B_3 \wedge A_G \wedge (\tau(B_3\rightarrow d) \leq \tau(d\rightarrow A_G)))$
$\mathrm{Link}(C_3,e,A_G) \Rightarrow (C_3 \wedge A_G \wedge (\tau(C_3\rightarrow e) \leq \tau(e\rightarrow A_G)))$

4. $(\mathrm{Link}(A_1,a,B_2) \wedge A_1) \Rightarrow (((\tau(A_1\rightarrow \neg a) < \tau(A_1\rightarrow a)) \vee (\tau(a\rightarrow B_2) < \tau(A_1\rightarrow \neg a)))$
$(\mathrm{Link}(A_1,a,B_2) \wedge A_2) \Rightarrow (((\tau(A_2\rightarrow \neg a) < \tau(A_1\rightarrow a)) \vee (\tau(a\rightarrow B_2) < \tau(A_2\rightarrow \neg a)))$
$(\mathrm{Link}(A_1,a,B_2) \wedge A_3) \Rightarrow (((\tau(A_3\rightarrow \neg a) < \tau(A_1\rightarrow a)) \vee (\tau(a\rightarrow B_2) < \tau(A_3\rightarrow \neg a)))$
$(\mathrm{Link}(A_1,a,B_3) \wedge A_1) \Rightarrow (((\tau(A_1\rightarrow \neg a) < \tau(A_1\rightarrow a)) \vee (\tau(a\rightarrow B_3) < \tau(A_1\rightarrow \neg a)))$
$(\mathrm{Link}(A_1,a,B_3) \wedge A_2) \Rightarrow (((\tau(A_2\rightarrow \neg a) < \tau(A_1\rightarrow a)) \vee (\tau(a\rightarrow B_3) < \tau(A_2\rightarrow \neg a)))$
$(\mathrm{Link}(A_1,a,B_3) \wedge A_3) \Rightarrow (((\tau(A_3\rightarrow \neg a) < \tau(A_1\rightarrow a)) \vee (\tau(a\rightarrow B_3) < \tau(A_3\rightarrow \neg a)))$
$(\mathrm{Link}(A_2,a,B_3) \wedge A_1) \Rightarrow (((\tau(A_1\rightarrow \neg a) < \tau(A_2\rightarrow a)) \vee (\tau(a\rightarrow B_3) < \tau(A_1\rightarrow \neg a)))$
$(\mathrm{Link}(A_2,a,B_3) \wedge A_2) \Rightarrow (((\tau(A_2\rightarrow \neg a) < \tau(A_2\rightarrow a)) \vee (\tau(a\rightarrow B_3) < \tau(A_2\rightarrow \neg a)))$
$(\mathrm{Link}(A_2,a,B_3) \wedge A_3) \Rightarrow (((\tau(A_3\rightarrow \neg a) < \tau(A_2\rightarrow a)) \vee (\tau(a\rightarrow B_3) < \tau(A_3\rightarrow \neg a)))$
$(\mathrm{Link}(B_2,c,C_3) \wedge B_2) \Rightarrow (((\tau(B_2\rightarrow \neg c) < \tau(B_2\rightarrow c)) \vee (\tau(c\rightarrow C_3) < \tau(B_2\rightarrow \neg c)))$
$(\mathrm{Link}(B_2,c,C_3) \wedge B_3) \Rightarrow (((\tau(B_3\rightarrow \neg c) < \tau(B_2\rightarrow c)) \vee (\tau(c\rightarrow C_3) < \tau(B_3\rightarrow \neg c)))$
$(\mathrm{Link}(B_2,d,A_G) \wedge A_1) \Rightarrow (((\tau(A_1\rightarrow \neg d) < \tau(B_2\rightarrow d)) \vee (\tau(d\rightarrow A_G) < \tau(A_1\rightarrow \neg d)))$
$(\mathrm{Link}(B_2,d,A_G) \wedge A_2) \Rightarrow (((\tau(A_2\rightarrow \neg d) < \tau(B_2\rightarrow d)) \vee (\tau(d\rightarrow A_G) < \tau(A_2\rightarrow \neg d)))$
$(\mathrm{Link}(B_2,d,A_G) \wedge A_3) \Rightarrow (((\tau(A_3\rightarrow \neg d) < \tau(B_2\rightarrow d)) \vee (\tau(d\rightarrow A_G) < \tau(A_3\rightarrow \neg d)))$
$(\mathrm{Link}(B_3,d,A_G) \wedge A_1) \Rightarrow (((\tau(A_1\rightarrow \neg d) < \tau(B_3\rightarrow d)) \vee (\tau(d\rightarrow A_G) < \tau(A_1\rightarrow \neg d)))$
$(\mathrm{Link}(B_3,d,A_G) \wedge A_2) \Rightarrow (((\tau(A_2\rightarrow \neg d) < \tau(B_3\rightarrow d)) \vee (\tau(d\rightarrow A_G) < \tau(A_2\rightarrow \neg d)))$
$(\mathrm{Link}(B_3,d,A_G) \wedge A_3) \Rightarrow (((\tau(A_3\rightarrow \neg d) < \tau(B_3\rightarrow d)) \vee (\tau(d\rightarrow A_G) < \tau(A_3\rightarrow \neg d)))$

$(A_1 \wedge A_2) \Rightarrow ((\tau(A_1\rightarrow a) \neq \tau(A_2\rightarrow \neg a))$
$(A_1 \wedge A_3) \Rightarrow ((\tau(A_1\rightarrow a) \neq \tau(A_3\rightarrow \neg a))$
$(A_1 \wedge C_3) \Rightarrow ((\tau(A_1\rightarrow b) \neq \tau(C_3\rightarrow \neg b))$
$(A_2 \wedge A_1) \Rightarrow ((\tau(A_2\rightarrow a) \neq \tau(A_1\rightarrow \neg a))$
$(A_2 \wedge A_3) \Rightarrow ((\tau(A_2\rightarrow a) \neq \tau(A_3\rightarrow \neg a))$
$(A_2 \wedge C_3) \Rightarrow ((\tau(A_2\rightarrow b) \neq \tau(C_3\rightarrow \neg b))$
$(A_3 \wedge A_1) \Rightarrow ((\tau(A_3\rightarrow a) \neq \tau(A_1\rightarrow \neg a))$
$(A_3 \wedge A_2) \Rightarrow ((\tau(A_3\rightarrow a) \neq \tau(A_2\rightarrow \neg a))$
$(A_3 \wedge C_3) \Rightarrow ((\tau(A_3\rightarrow b) \neq \tau(C_3\rightarrow \neg b))$
$(B_2 \wedge B_3) \Rightarrow ((\tau(B_2\rightarrow c) \neq \tau(B_3\rightarrow \neg c))$
$(B_3 \wedge B_2) \Rightarrow ((\tau(B_3\rightarrow c) \neq \tau(B_2\rightarrow \neg c))$
$(B_2 \wedge A_1) \Rightarrow ((\tau(B_2\rightarrow d) \neq \tau(A_1\rightarrow \neg d))$
$(B_2 \wedge A_2) \Rightarrow ((\tau(B_2\rightarrow d) \neq \tau(A_2\rightarrow \neg d))$
$(B_2 \wedge A_3) \Rightarrow ((\tau(B_2\rightarrow d) \neq \tau(A_3\rightarrow \neg d))$
$(B_3 \wedge A_1) \Rightarrow ((\tau(B_3\rightarrow d) \neq \tau(A_1\rightarrow \neg d))$
$(B_3 \wedge A_2) \Rightarrow ((\tau(B_3\rightarrow d) \neq \tau(A_2\rightarrow \neg d))$
$(B_3 \wedge A_3) \Rightarrow ((\tau(B_3\rightarrow d) \neq \tau(A_3\rightarrow \neg d))$

5. Littéraux bornes inf et sup : $(\tau(A_I)\leq \tau(A_G))$
$(\tau(A_I)\leq \tau_s(A_1)) \wedge (\tau(A_I)\leq \tau_s(A_2)) \ldots (\tau(A_I)\leq \tau_s(C_3))$
$(\tau_e(A_1)\leq \tau(A_G)) \wedge (\tau_e(A_2)\leq \tau(A_G)) \ldots (\tau_e(C_3)\leq \tau(A_G))$

Codage TLP-GP-2 – étape 2 (18 var. propositionnelles, 8 var. réelles, 89 clauses) :

1. $A_I \wedge A_G$

2. $B_2 \Rightarrow \mathrm{Link}(A_1,a,B_2)$
$B_3 \Rightarrow \mathrm{Link}(A_1,a,B_3) \vee \mathrm{Link}(A_2,a,B_3)$
$C_3 \Rightarrow \mathrm{Link}(B_2,c,C_3)$
$A_G \Rightarrow \mathrm{Link}(A_1,b,A_G) \vee \mathrm{Link}(A_2,b,A_G) \vee \mathrm{Link}(A_3,b,A_G)$
$A_G \Rightarrow \mathrm{Link}(B_2,d,A_G) \vee \mathrm{Link}(B_3,d,A_G)$
$A_G \Rightarrow \mathrm{Link}(C_3,e,A_G)$

3. $\mathrm{Link}(A_1,a,B_2) \Rightarrow (A_1 \wedge B_2 \wedge (\tau_s(A_1) \leq \tau_s(B_2)))$
$\mathrm{Link}(A_1,a,B_3) \Rightarrow (A_1 \wedge B_3 \wedge (\tau_s(A_1) \leq \tau_s(B_3)))$
$\mathrm{Link}(A_2,a,B_3) \Rightarrow (A_2 \wedge B_3 \wedge (\tau_s(A_2) \leq \tau_s(B_3))))$
$\mathrm{Link}(B_2,c,C_3) \Rightarrow (B_2 \wedge C_3 \wedge (\tau_s(B_2) \leq \tau_s(C_3)))$
$\mathrm{Link}(A_1,b,A_G) \Rightarrow (A_1 \wedge A_G \wedge (\tau_s(A_1)+5 \leq \tau(A_G)))$
$\mathrm{Link}(A_2,b,A_G) \Rightarrow (A_2 \wedge A_G \wedge (\tau_s(A_2)+5 \leq \tau(A_G)))$
$\mathrm{Link}(A_3,b,A_G) \Rightarrow (A_3 \wedge A_G \wedge (\tau_s(A_3)+5 \leq \tau(A_G)))$
$\mathrm{Link}(B_2,d,A_G) \Rightarrow (B_2 \wedge A_G \wedge (\tau_s(B_2)+4 \leq \tau(A_G)))$
$\mathrm{Link}(B_3,d,A_G) \Rightarrow (B_3 \wedge A_G \wedge (\tau_s(B_3)+4 \leq \tau(A_G)))$
$\mathrm{Link}(C_3,e,A_G) \Rightarrow (C_3 \wedge A_G \wedge (\tau_s(C_3)+1 \leq \tau(A_G)))$

4. $(\mathrm{Link}(A_1,a,B_2) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(A_1)) \vee (\tau_s(B_2) < \tau_s(A_1)+5))$
$(\mathrm{Link}(A_1,a,B_2) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(A_1)) \vee (\tau_s(B_2) < \tau_s(A_2)+5))$
$(\mathrm{Link}(A_1,a,B_2) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(A_1)) \vee (\tau_s(B_2) < \tau_s(A_3)+5))$
$(\mathrm{Link}(A_1,a,B_3) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(A_1)) \vee (\tau_s(B_3) < \tau_s(A_1)+5))$
$(\mathrm{Link}(A_1,a,B_3) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(A_1)) \vee (\tau_s(B_3) < \tau_s(A_2)+5))$
$(\mathrm{Link}(A_1,a,B_3) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(A_1)) \vee (\tau_s(B_3) < \tau_s(A_3)+5))$
$(\mathrm{Link}(A_2,a,B_3) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(A_2)) \vee (\tau_s(B_3) < \tau_s(A_1)+5))$
$(\mathrm{Link}(A_2,a,B_3) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(A_2)) \vee (\tau_s(B_3) < \tau_s(A_2)+5))$
$(\mathrm{Link}(A_2,a,B_3) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(A_2)) \vee (\tau_s(B_3) < \tau_s(A_3)+5))$
$(\mathrm{Link}(B_2,c,C_3) \wedge B_2) \Rightarrow ((\tau_s(B_2)+4 < \tau_s(B_2)) \vee (\tau_s(C_3) < \tau_s(B_2)+4))$
$(\mathrm{Link}(B_2,c,C_3) \wedge B_3) \Rightarrow ((\tau_s(B_3)+4 < \tau_s(B_2)) \vee (\tau_s(C_3) < \tau_s(B_3)+4))$
$(\mathrm{Link}(B_2,d,A_G) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(B_2)+4) \vee (\tau(A_G) < \tau_s(A_1)+5))$
$(\mathrm{Link}(B_2,d,A_G) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(B_2)+4) \vee (\tau(A_G) < \tau_s(A_2)+5))$
$(\mathrm{Link}(B_2,d,A_G) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(B_2)+4) \vee (\tau(A_G) < \tau_s(A_3)+5))$
$(\mathrm{Link}(B_3,d,A_G) \wedge A_1) \Rightarrow ((\tau_s(A_1)+5 < \tau_s(B_3)+4) \vee (\tau(A_G) < \tau_s(A_1)+5))$
$(\mathrm{Link}(B_3,d,A_G) \wedge A_2) \Rightarrow ((\tau_s(A_2)+5 < \tau_s(B_3)+4) \vee (\tau(A_G) < \tau_s(A_2)+5))$
$(\mathrm{Link}(B_3,d,A_G) \wedge A_3) \Rightarrow ((\tau_s(A_3)+5 < \tau_s(B_3)+4) \vee (\tau(A_G) < \tau_s(A_3)+5))$

$(A_1 \wedge A_2) \Rightarrow (\tau_s(A_1) \neq \tau_s(A_2)+5)$
$(A_1 \wedge A_3) \Rightarrow (\tau_s(A_1) \neq \tau_s(A_3)+5)$
$(A_1 \wedge C_3) \Rightarrow (\tau_s(A_1)+5 \neq \tau_s(C_3)+1)$
$(A_2 \wedge A_1) \Rightarrow (\tau_s(A_2) \neq \tau_s(A_1)+5)$
$(A_2 \wedge A_3) \Rightarrow (\tau_s(A_2) \neq \tau_s(A_3)+5)$
$(A_2 \wedge C_3) \Rightarrow (\tau_s(A_2)+5 \neq \tau_s(C_3)+1)$
$(A_3 \wedge A_1) \Rightarrow (\tau_s(A_3) \neq \tau_s(A_1)+5)$
$(A_3 \wedge A_2) \Rightarrow (\tau_s(A_3) \neq \tau_s(A_2)+5)$
$(A_3 \wedge C_3) \Rightarrow (\tau_s(A_3)+5 \neq \tau_s(C_3)+1)$
$(B_2 \wedge B_3) \Rightarrow (\tau_s(B_2) \neq \tau_s(B_3)+4)$
$(B_3 \wedge B_2) \Rightarrow (\tau_s(B_3) \neq \tau_s(B_2)+4)$
$(B_2 \wedge A_1) \Rightarrow (\tau_s(B_2)+4 \neq \tau_s(A_1)+5)$
$(B_2 \wedge A_2) \Rightarrow (\tau_s(B_2)+4 \neq \tau_s(A_2)+5)$
$(B_2 \wedge A_3) \Rightarrow (\tau_s(B_2)+4 \neq \tau_s(A_3)+5)$
$(B_3 \wedge A_1) \Rightarrow (\tau_s(B_3)+4 \neq \tau_s(A_1)+5)$
$(B_3 \wedge A_2) \Rightarrow (\tau_s(B_3)+4 \neq \tau_s(A_2)+5)$
$(B_3 \wedge A_3) \Rightarrow (\tau_s(B_3)+4 \neq \tau_s(A_3)+5)$

5. Littéraux bornes inf et sup : $(\tau(A_I) \leq \tau(A_G))$
$(\tau(A_I) \leq \tau_s(A_1)) \wedge (\tau(A_I) \leq \tau_s(A_2)) \ldots (\tau(A_I) \leq \tau_s(C_3))$
$(\tau_e(A_1) \leq \tau(A_G)) \wedge (\tau_e(A_2) \leq \tau(A_G)) \ldots (\tau_e(C_3) \leq \tau(A_G))$

Codage TLP-GP-2 – étape 3 (format SMT-LIB) :

```
(benchmark exemple_tlp-gp-2.smt
:source {
TLP-GP-2
F. Maris
IRIT - Universite Paul Sabatier - Toulouse - 2008
}
:logic QF_RDL
:extrafuns ((t_Init Int))
:extrafuns ((t_Goal Int))
:extrafuns ((t_A1 Int))
:extrafuns ((t_A2 Int))
:extrafuns ((t_A3 Int))
:extrafuns ((t_B2 Int))
:extrafuns ((t_B3 Int))
:extrafuns ((t_C3 Int))
:extrapreds ((Init))
:extrapreds ((Goal))
:extrapreds ((A1))
:extrapreds ((A2))
:extrapreds ((A3))
:extrapreds ((B2))
:extrapreds ((B3))
:extrapreds ((C3))
:extrapreds ((Link_A1.a.B2))
:extrapreds ((Link_A1.a.B3))
:extrapreds ((Link_A2.a.B3))
:extrapreds ((Link_B2.c.C3))
:extrapreds ((Link_A1.b.Goal))
:extrapreds ((Link_A2.b.Goal))
:extrapreds ((Link_A3.b.Goal))
:extrapreds ((Link_B2.d.Goal))
:extrapreds ((Link_B3.d.Goal))
:extrapreds ((Link_C3.e.Goal))
:formula
( and
Init Goal

(or (not B2) Link_A1.a.B2)
(or (not B3) Link_A1.a.B3 Link_A2.a.B3)
(or (not C3) Link_B2.c.C3)
(or (not Goal) Link_A1.b.Goal Link_A2.b.Goal Link_A3.b.Goal)
(or (not Goal) Link_B2.d.Goal Link_B3.d.Goal)
(or (not Goal) Link_C3.e.Goal)

(or (not Link_A1.a.B2) A1)
(or (not Link_A1.a.B2) B2)
(or (not Link_A1.a.B2) ( >= (- t_B2 t_A1) 0))
(or (not Link_A1.a.B3) A1)
(or (not Link_A1.a.B3) B3)
(or (not Link_A1.a.B3) ( >= (- t_B3 t_A1) 0))
(or (not Link_A2.a.B3) A2)
(or (not Link_A2.a.B3) B3)
(or (not Link_A2.a.B3) ( >= (- t_B3 t_A2) 0))
(or (not Link_B2.c.C3) B2)
(or (not Link_B2.c.C3) C3)
(or (not Link_B2.c.C3) ( >= (- t_C3 t_B2) 0))
(or (not Link_A1.b.Goal) A1)
(or (not Link_A1.b.Goal) Goal)
(or (not Link_A1.b.Goal) ( >= (- t_Goal t_A1) 5))
(or (not Link_A2.b.Goal) A2)
(or (not Link_A2.b.Goal) Goal)
(or (not Link_A2.b.Goal) ( >= (- t_Goal t_A2) 5))
(or (not Link_A3.b.Goal) A3)
(or (not Link_A3.b.Goal) Goal)
(or (not Link_A3.b.Goal) ( >= (- t_Goal t_A3) 5))
(or (not Link_B2.d.Goal) B2)
(or (not Link_B2.d.Goal) Goal)
(or (not Link_B2.d.Goal) ( >= (- t_Goal t_B2) 4))
(or (not Link_B3.d.Goal) B3)
(or (not Link_B3.d.Goal) Goal)
(or (not Link_B3.d.Goal) ( >= (- t_Goal t_B3) 4))
(or (not Link_C3.e.Goal) C3)
(or (not Link_C3.e.Goal) Goal)
(or (not Link_C3.e.Goal) ( >= (- t_Goal t_C3) 1))
```

```
(or  (not Link_A1.a.B2) (not A1) (> (- t_A1 t_A1) 5) (< (- t_B2 t_A1) 5))
(or  (not Link_A1.a.B2) (not A2) (> (- t_A1 t_A2) 5) (< (- t_B2 t_A2) 5))
(or  (not Link_A1.a.B2) (not A3) (> (- t_A1 t_A3) 5) (< (- t_B2 t_A3) 5))
(or  (not Link_A1.a.B3) (not A1) (> (- t_A1 t_A1) 5) (< (- t_B3 t_A1) 5))
(or  (not Link_A1.a.B3) (not A2) (> (- t_A1 t_A2) 5) (< (- t_B3 t_A2) 5))
(or  (not Link_A1.a.B3) (not A3) (> (- t_A1 t_A3) 5) (< (- t_B3 t_A3) 5))
(or  (not Link_A2.a.B3) (not A1) (> (- t_A2 t_A1) 5) (< (- t_B3 t_A1) 5))
(or  (not Link_A2.a.B3) (not A2) (> (- t_A2 t_A2) 5) (< (- t_B3 t_A2) 5))
(or  (not Link_A2.a.B3) (not A3) (> (- t_A2 t_A3) 5) (< (- t_B3 t_A3) 5))
(or  (not Link_B2.c.C3) (not B2) (> (- t_B2 t_B2) 4) (< (- t_C3 t_B2) 4))
(or  (not Link_B2.c.C3) (not B3) (> (- t_B2 t_B3) 4) (< (- t_C3 t_B3) 4))
(or  (not Link_B2.d.Goal) (not A1) (> (- t_B2 t_A1) 1) (< (- t_Goal t_A1) 5))
(or  (not Link_B2.d.Goal) (not A2) (> (- t_B2 t_A2) 1) (< (- t_Goal t_A2) 5))
(or  (not Link_B2.d.Goal) (not A3) (> (- t_B2 t_A3) 1) (< (- t_Goal t_A3) 5))
(or  (not Link_B3.d.Goal) (not A1) (> (- t_B3 t_A1) 1) (< (- t_Goal t_A1) 5))
(or  (not Link_B3.d.Goal) (not A2) (> (- t_B3 t_A2) 1) (< (- t_Goal t_A2) 5))
(or  (not Link_B3.d.Goal) (not A3) (> (- t_B3 t_A3) 1) (< (- t_Goal t_A3) 5))

(or (not A1) (not A2) (distinct (- t_A1 t_A2) 5))
(or (not A1) (not A3) (distinct (- t_A1 t_A3) 5))
(or (not A1) (not C3) (distinct (- t_C3 t_A1) 4))
(or (not A2) (not A1) (distinct (- t_A2 t_A1) 5))
(or (not A2) (not A3) (distinct (- t_A2 t_A3) 5))
(or (not A2) (not C3) (distinct (- t_C3 t_A2) 4))
(or (not A3) (not A1) (distinct (- t_A3 t_A1) 5))
(or (not A3) (not A2) (distinct (- t_A3 t_A2) 5))
(or (not A3) (not C3) (distinct (- t_C3 t_A3) 4))
(or (not B2) (not B3) (distinct (- t_B2 t_B3) 4))
(or (not B3) (not B2) (distinct (- t_B3 t_B2) 4))
(or (not B2) (not A1) (distinct (- t_B2 t_A1) 1))
(or (not B2) (not A2) (distinct (- t_B2 t_A2) 1))
(or (not B2) (not A3) (distinct (- t_B2 t_A3) 1))
(or (not B3) (not A1) (distinct (- t_B3 t_A1) 1))
(or (not B3) (not A2) (distinct (- t_B3 t_A2) 1))
(or (not B3) (not A3) (distinct (- t_B3 t_A3) 1))

(>= t_Goal t_Init)
(>= t_A1 t_Init) (>= t_A2 t_Init) (>= t_A3 t_Init)
(>= t_B2 t_Init) (>= t_B3 t_Init) (>= t_C3 t_Init)
(>= (- t_Goal t_A1) 5) (>= (- t_Goal t_A2) 5) (>= (- t_Goal t_A3) 5)
(>= (- t_Goal t_B2) 4) (>= (- t_Goal t_B3) 4) (>= (- t_Goal t_C3) 1)
))
```

Annexe 7 : Interface graphique de TLP-GP

Environnement :

L'interface graphique de TLP-GP codée en JAVA fonctionne sous les systèmes d'exploitations courantes (Windows, Linux, MacOS) et peut être lancée indépendamment du planificateur. Elle utilise, pour l'affichage d'un plan-solution flottant, le fichier .smt issu de l'exécution de TLP-GP ainsi que le fichier .pddl décrivant le domaine afin d'effectuer un affichage compact des actions. Ces fichiers sont chargés par l'utilisateur. L'interface permet également de lancer directement la planification en choisissant le domaine et le problème étudié avec une configuration possible de l'algorithme. L'affichage du plan solution est alors instantané après exécution de TLP-GP et l'utilisateur peut par la suite interagir avec le plan solution avant de pouvoir ou non sauvegarder les modifications effectuées.

Fonctionnalités :

Voici les différentes fonctionnalités qui sont offertes par l'interface graphique de TLP-GP :

Fonctions d'édition :

- lancer le planificateur avec configuration de l'algorithme et importer un fichier au format .pddl, .smt ;
- imprimer la représentation du plan solution ;
- annuler la dernière action effectuée par l'utilisateur ;
- revenir sur l'action annulée ;
- sauvegarder et charger la représentation du plan solution au format .txt.

Fonctions d'affichage :

- afficher toutes les actions du plan solution en respectant les contraintes ;
- afficher une arborescence de toutes les actions du plan solution ;
- afficher/masquer les différents effets et préconditions (tout afficher, tout masquer ou afficher seulement ceux en lien avec le(s) action(s) sélectionnée(s)) ;
- sélectionner une action afin d'afficher ses informations (couleur utilisée, code PDDL associé) par un double clic ;
- mettre en surbrillance l'action sélectionnée ainsi que ses effets et ses préconditions si ceux-ci sont visibles par un simple clic ;
- sélectionner plusieurs actions à la fois avec mise en surbrillance ;
- afficher une barre du temps avec un indicateur marquant l'instant où l'on se situe par rapport à l'état initial et afficher les différents fluents vrais à cet instant ;
- afficher des événements exogènes représentés par des segments colorés le long de la barre du temps ;
- faire défiler en temps réel l'indicateur avec surbrillance des actions en cours et des fluents vérifiés ;

- déplacer la vue du plan solution de manière verticale et horizontale ;
- zoomer/dézoomer sur le plan solution selon l'axe horizontal, vertical ou global avec une granularité des détails affichés adaptée au niveau de zoom.

Fonctions de modification :

- déplacer une (des) action(s) de manière stricte en respectant les contraintes auxquelles elle(s) est(sont) liée(s) sans déplacer les autres actions lors des cas limites ;
- déplacer une action avec déplacement contraint des autres actions jusqu'au seuil minimum (compactage maximum des actions précédent l'action sélectionnée) ou au seuil maximum (durée limite du plan solution) ;
- compacter au maximum un ensemble d'actions ou toutes les actions du plan solution ;
- ouvrir une fenêtre de modification lors du double clic sur un objet (action, effet, précondition, événement exogène) pour modifier son nom, sa couleur, sa transparence, la durée dans le cas d'une action et ajouter des commentaires ;
- modifier à la souris la durée d'une action par une sélection d'un bord (au choix entre l'instant de début et l'instant de fin de l'action) de celle-ci et le déplacer selon la variation souhaitée .

TITLE: SAT Planning and Temporally Expressive Planning. The TSP and TLP-GP systems.

ABSTRACT

This thesis deals with Artificial Intelligence planning. After introducing the domain and the main algorithms in the classical framework of planning, we present a state of the art of SAT planning.

We analyse in detail this approach which allows us to benefit directly from improvements brought regularly to SAT solvers. We propose new encodings integrating a least-commitment strategy postponing as much as possible the scheduling of actions. We then introduce the TSP system which we have implemented to equitably compare the different encodings and we detail the results of numerous experimental tests which show the superiority of our encodings in comparison with the existing ones.

We introduce then a state of the art of temporal planning by analysing algorithms and expressiveness of their representation languages. The great majority of these planners do not allow us to solve real problems for which the concurrency of actions is required. We then detail the two original approaches of our TLP-GP system which allow us to solve this type of problem. As with SAT planning, a large part of search work is delegated to a SMT solver. We then propose extensions of the PDDL planning language which allows us to a certain extent to take into account uncertainty, choice, or continuous transitions. We show finally, thanks to an experimental study, that our algorithms allow us to solve real problems requiring numerous concurrent actions.

KEYWORDS

Planning, SAT encodings, least-commitment, temporally-expressive problems, planning graph.

AUTEUR : Frédéric MARIS

TITRE : Planification SAT et Planification Temporellement Expressive. Les Systèmes TSP et TLP-GP.

DIRECTEUR DE THESE : Pierre REGNIER

RESUME

Cette thèse s'inscrit dans le cadre de la planification de tâches en intelligence artificielle. Après avoir introduit le domaine et les principaux algorithmes de planification dans le cadre classique, nous présentons un état de l'art de la planification SAT.

Nous analysons en détail cette approche qui permet de bénéficier directement des améliorations apportées régulièrement aux solveurs SAT. Nous proposons de nouveaux codages qui intègrent une stratégie de moindre engagement en retardant le plus possible l'ordonnancement des actions. Nous présentons ensuite le système TSP que nous avons implémenté pour comparer équitablement les différents codages puis nous détaillons les résultats de nombreux tests expérimentaux qui démontrent la supériorité de nos codages par rapport aux codages existants.

Nous présentons ensuite un état de l'art de la planification temporelle en analysant les algorithmes et l'expressivité de leurs langages de représentation. La très grande majorité de ces planificateurs ne permet pas de résoudre des problèmes réels pour lesquels la concurrence des actions est nécessaire. Nous détaillons alors les deux approches originales de notre système TLP-GP permettant de résoudre ce type de problèmes. Ces approches sont comparables à la planification SAT, une grande partie du travail de recherche étant déléguée à un solveur SMT. Nous proposons ensuite des extensions du langage de planification PDDL qui permettent une certaine prise en compte de l'incertitude, du choix, ou des transitions continues. Nous montrons enfin, grâce à une étude expérimentale, que nos algorithmes permettent de résoudre des problèmes réels nécessitant de nombreuses actions concurrentes.

MOTS–CLES

Planification, codages SAT, relations d'autorisation, problèmes temporellement expressifs, graphe de planification.

DISCIPLINE ADMINISTRATIVE

Informatique

INTITULE ET ADRESSE DE L'U.F.R. OU DU LABORATOIRE

Institut de Recherche en Informatique de Toulouse, UMR 5505 CNRS–INP–UPS, Université Paul Sabatier, 118 route de Narbonne, 31062 Toulouse Cedex 4