Actes JFPC 2005

Plus d'inférence et moins de recherche pour la résolution de problèmes de planification simples

Vincent Vidal [1*] **Héctor Geffner** [2†]

[1] CRIL - Université d'artois
rue de l'université - SP16
62307 Lens Cedex, France
[2] ICREA & Universitat Pompeu Fabra
Paseo de Circunvalacion 8
08003 Barcelona, Espagne
vidal@cril.univ-artois.fr hector.geffner@upf.edu

Résumé

De nombreux problèmes utilisés en planification de tâches dans le domaine de l'Intelligence Artificielle comme Blocks, Logistics, Gripper, Satellite et d'autres, ne possèdent pas les interactions qui caractérisent les puzzles. Ils peuvent être résolus rapidement mais non optimalement en temps polynomial. Ce sont en effet des problèmes faciles pour les humains, mais comme beaucoup d'autres problèmes en Intelligence Artificielle, difficiles pour les machines. Dans ce travail, nous étudions le type d'inférences requises dans un planificateur indépendant du domaine pour résoudre des problèmes simples en évitant au maximum de faire des retours arrière, en ajoutant uniquement quelques opérations polynomiales à chaque nœud de l'arbre de recherche. A cette fin, nous utilisons le planificateur temporel optimal CPT qui combine un schéma de branchement de type POCL avec des mécanismes d'inférence puissants, et montrons que l'ajout de quelques règles d'inférence simples et générales suffisent pour éliminer les retours arrière pour de nombreux domaines. Il s'agit là d'un résultat empirique intéressant, à notre avis, qui pourrait contribuer au développement de planificateurs automatiques plus robustes, et à une meilleure compréhension de la façon de planifier des humains. Nous rapportons aussi une amélioration des performances significative par rapport à CPT.

*V. Vidal est en partie supporté par l'IUT de Lens, le CNRS et la Régon Nord/Pas-de-Calais sous le programme COCOA.

†H. Geffner est en partie supporté par le programme TIC2002-04470-C03-02, MCyT, Espagne.

Abstract

Many problems used in AI planning including Blocks, Logistics, Gripper, Satellite, and others lack the interactions that characterize puzzles and can be solved non-optimally in low polynomial time. They are indeed easy problems for people, although as with many other problems in AI, not always easy for machines. In this work, we study the type of inferences that are required in a domain-independent planner for solving simple problems such as these in a backtrack-free manner by performing polynomial node operations. For this, we make use of the optimal temporal planner CPT which combines a POCL branching scheme with strong inference mechanisms, and show that a few simple and general additional inference mechanisms suffice to render the search over various domains backtrack free. This is an interesting empirical finding, we believe, that may contribute to the development of more robust automated planners, and to a better understanding of human planning. Significant performance gains in relation to CPT are also reported.

1 Introduction

De nombreux problèmes utilisés en planification de tâches dans le domaine de l'Intelligence Artificielle comme Blocks, Logistics, Gripper, Satellite et d'autres, ne possèdent pas les interactions qui caractérisent les puzzles. Ils peuvent être résolus rapidement mais non optimalement en temps polynomial. Ce sont en effet des problèmes faciles pour les humains, mais comme beaucoup d'autres problèmes en Intelligence Artificielle, difficiles pour les machines. Dans ce travail, nous étudions le type d'inférences requises dans un planificateur indépendant du domaine pour résoudre des problèmes simples en évitant au maximum de faire des retours arrière, en ajoutant uniquement quelques opérations polynomiales à chaque nœud de l'arbre de recherche. Pour cela, nous utilisons le planificateur temporel optimal CPT qui combine un schéma de branchement de type POCL avec des mécanismes d'inférence puissants [26, 28], et montrons que l'ajout de quelques règles d'inférence simples et générales suffisent pour éliminer les retours arrière pour de nombreux domaines.

Pour discuter des planificateurs indépendants du domaine dont le but est de résoudre des problèmes de planification simples sans retour arrière en effectuant des opérations polynomiales à chaque nœud, nous utiliserons le terme de *planificateurs aisés*. Nous pensons que le développement de planificateurs aisés est une tâche particulièrement sensée et motivée, qui pourrait contribuer non seulement au développement de planificateurs automatiques plus robustes, mais aussi à une meilleure compréhension des mécanismes de planification de l'être humain. Les humains sont en effet capables de résoudre facilement ces problèmes ; et bien qu'il soit souvent considéré que cette capacité est le résultat de stratégies dépendantes du domaine, nos résultats suggèrent qu'elle peut aussi découler de mécanismes d'inférence simples et généraux.

Les planificateurs aisés sont des planificateurs non optimaux, mais tandis que les planificateurs non optimaux cherchent à résoudre des problèmes par n'importe quel moyen, et que les planificateurs optimaux cherchent à les résoudre optimalement, les planificateurs aisés cherchent à résoudre les problèmes simples avec des opérations polynomiales rapides et *sans recherche*. Cela ne signifie pas qu'ils doivent résoudre ces problèmes plus rapidement, ou qu'ils doivent en résoudre plus, mais qu'ils doivent tenir compte d'inférences qui rendent ces problèmes faciles. Nous pensons qu'une telle utilisation d'inférences peut être bénéfique pour les performances des planificateurs, et montrons que c'est le cas pour CPT. En lui-même, CPT, comme les autres planificateurs basés sur les contraintes ou sur SAT, n'est pas un bon planificateur non optimal et encore moins un planificateur aisé. En effet, les planificateurs optimaux basés sur les contraintes ou sur SAT [15, 23, 6] utilisés avec un horizon suffisamment large pour résoudre des problèmes non optimalement, rencontrent deux problèmes :

1. Les codages SAT et CSP basés sur une variable par unité de temps, les plus courants, acquièrent une taille trop importante pour un horizon élevé.
2. Les contraintes qui requièrent la validité des buts d'un problème à l'horizon deviennent inefficaces (i.e. perdent leur pouvoir d'élagage) lorsque cet horizon est trop élevé.

Le premier point n'est pas un problème pour CPT, étant donné qu'il s'agit d'un planificateur temporel utilisant une représentation temporelle plutôt que booléenne. Ainsi, l'utilisation d'une borne élevée sur la durée totale d'exécution d'un plan (le *makespan*) a des conséquences directes sur le *domaine* des variables temporelles, et non sur leur *nombre*.

CPT, d'un autre côté, n'échappe pas au deuxième problème : avec une borne élevée sur le makespan, la recherche devient beaucoup moins contrainte et dirigée, et même des problèmes résolus optimalement sans retour arrière ne peuvent être résolus après un nombre considérable de retours arrière quand une borne élevée sur le makespan est spécifiée. Dans ce travail, nous nous attaquons à ce problème en étendant les capacités d'inférence de CPT de telle sorte qu'il dépende moins des inférences effectuées grâce à la borne sur le makespan et plus sur des inférences indépendantes du domaine qui ne sont pas capturées par CPT. La nouvelle version de CPT, que nous appelons eCPT, effectue un raisonnement simple mais plus étendu, tirant parti de l'adaptation de techniques comme les points de passage obligatoires [22, 30] et des distances [25], parmi d'autres.

Cet article est organisé de la façon suivante. Nous passons d'abord en revue le planificateur CPT, discutons de ses forces comme planificateur optimal et de ses faiblesses comme planificateur non optimal, et introduisons des extensions à son moteur d'inférences qui éliminent les retours arrière de la recherche sur un grand nombre de domaines de test. Nous évaluons enfin le planificateur qui en résulte, eCPT, et discutons les implications et pistes de recherche.

2 CPT

CPT est un planificateur temporel indépendant du domaine qui combine un schéma de branchement basé sur la planification dans les espaces de plans partiels avec liens causaux (POCL : « Partial Order Causal Link ») avec des règles d'élagage puissantes et saines implémentées par des contraintes [26]. La principale innovation de CPT par rapport à d'autres formulations [13, 19, 29] est la capacité de raisonner sur les supports, précédences et liens causaux impliquant aussi les actions qui n'appartiennent pas encore à un plan partiel. Ainsi, CPT peut réduire les bornes sur la date de début et le domaine des supports des actions qui ne sont pas encore dans le plan, éliminer des actions de tout plan partiel, détecter des inconsistances au plus tôt, etc. Les inférences dans CPT sont supportées par une représentation adaptée des plans partiels avec liens causaux en terme de variables et de contraintes. Par exemple, à chaque action a dans le domaine est associée une variable $T(a)$ qui représente la date de début de a ; et à chaque précondition p de a, est associée une variable $S(p, a)$ qui représente le support de la précondition p pour l'action a. Un lien causal $a'[p]a$ est ainsi représenté par la contrainte $S(p, a) = a'$, tandis que sa négation est représentée par la contrainte $S(p, a) \neq a'$. Cependant, à la différence d'autres planificateurs de type POCL basés sur les contraintes [11, 12, 16, 21], CPT représente et résonne avec toutes les variables, qu'une action appartienne ou non au plan partiel courant.

CPT utilise une extension simple du langage Strips qui combine les actions concurrentes avec des durées entières (bien que l'on puisse, par une simple transformation, utiliser des durées rationnelles). Un problème de planification temporel est un tuple $P = \langle A, I, O, G \rangle$ où A est un ensemble d'atomes de base, $I \subseteq A$ et $G \subseteq A$ représentent la situation initiale et le but, et O est l'ensemble des opérateurs Strips de base (totalement instanciés), chacun avec listes de préconditions, ajouts et retraits $pre(a)$, $add(a)$, et $del(a)$, et *durée* $dur(a)$. De manière classique en planification POCL, on trouve les actions $Start$ et End de durée nulle, la première sans précondition et comme ajouts les atomes de I, et la seconde avec préconditions les atomes de G et aucun effet. Comme dans GRAPHPLAN [3], deux actions a et a' interfèrent quand l'une retire une précondition ou un ajout de l'autre. CPT suit le modèle temporel simple de [24], dans lequel des actions interférentes ne peuvent se recouvrir dans le temps, et produit des plans avec durée d'exécution minimale (*makespan* minimal).

La formulation de base du planificateur CPT peut être décrite en quatre parties : *pré-traitement, variables, contraintes, et branchement*. Après le pré-traitement, les variables sont créées et les contraintes sont introduites et propagées. Si une inconsistance est rencontrée, il n'existe aucun plan valide pour le problème. Sinon, la contrainte $T(End) = B$, pour la borne B sur le makespan initialisée à la date de début minimale de l'action End, est introduite et propagée. Le schéma de branchement entre alors en action et si aucune solution n'est trouvée, ce procédé se répète en rétractant la contrainte $T(End) = B$ et en la remplaçant par $T(End) = B + 1$, et ainsi de suite. Pour plus de simplicité, nous suivrons le modèle de [26] et supposerons qu'aucune action du domaine ne peut être présente plus d'une fois dans le plan. Cette restriction est supprimée dans la dernière version de CPT qui est celle que nous utilisons, par la différenciation entre les types d'action et les instances d'action. Ces détails décrits dans [27] ne sont pas utiles ici et seront omis.

2.1 Pré-traitement

Dans la phase de pré-traitement, CPT calcule les valeurs heuristiques $h_T^2(a)$ et $h_T^2(\{p, q\})$ pour chaque action $a \in O$ et chaque paire d'atomes $\{p, q\}$ comme dans [7]. Ces valeurs procurent des bornes initiales sur la date minimale de production des préconditions de a et des paires d'atomes p, q, depuis la situation initiale I. Les *mutex structurels* sont alors identifiés comme les paires d'atomes p, q telles que $h_T^2(\{p, q\}) = \infty$. Une action a *e-retire* un atome p quand soit a retire p, soit a ajoute un atome q tel que q et p sont mutex, ou une précondition r de a est mutex avec p et a n'ajoute pas p. Dans tous les cas, si a e-retire p, p est faux après l'exécution de a [20].

En addition, l'heuristique plus simple h_T^1 est utilisée pour définir des *distances* entre actions [25]. Pour chaque action $a \in O$, l'heuristique h_T^1 est calculée depuis la situation initial I_a qui inclut tous les atomes *excepté ceux qui sont e-retirés par a*. Les distances $dist(a, a')$ sont alors initialisées avec les valeurs résultantes $h_T^1(a')$. Ces distances encodent les bornes minimales sur l'*écart temporel* qui existe entre la fin de l'exécution de a et le début de l'exécution de a' dans tout plan valide dans lequel a' suit a. Elles ne sont en général pas symétriques et leur calcul, qui reste polynomial, implique $|O|$ exécutions de l'heuristique h_T^1.

2.2 Variables et domaines

Un état du planificateur est représenté par une collection de variables, de domaines, et de contraintes. Comme mis en valeur plus haut, les variables sont définies pour toutes les actions $a \in O$ et pas seulement pour les actions du plan partiel courant. De plus, les variables sont créées pour chaque précondition p de chaque action a comme précisé ci-dessous. Le domaine d'une variable X est noté $D[X]$ ou plus simplement $X :: [X_{min}, X_{max}]$ si X est une variable numérique. Les variables, leur domaine initial, et leur signification sont :

- $T(a) :: [0, \infty]$ encode la date de début de chaque action a, avec $T(Start) = 0$
- $S(p, a)$ encode le support de la précondition p de l'action a avec domaine initial $D[S(p, a)] = O(p)$ où $O(p)$ est l'ensemble des actions de O qui ajoutent p
- $T(p, a) :: [0, \infty]$ encode la date de début de $S(p, a)$
- $InPlan(a) :: [0, 1]$ indique la présence de a dans le plan ; $InPlan(Start) = InPlan(End) = 1$ (vrai)

Les variables $T(a)$, $S(p, a)$, et $T(p, a)$ associées aux actions a qui ne sont ni présentes dans le plan partiel ni exclues de tout plan partiel (i.e., les actions pour lesquelles la variable $InPlan(a)$ peut être affectée à 0 ou 1), sont *conditionnelles* dans le sens suivant : ces variables et leur domaine ne seront significatifs que sous l'hypothèse qu'ils font partie du plan. Afin de s'assurer de cette interprétation, certaines précautions dans la propagation des contraintes doivent être prises, comme décrit dans [26].

2.3 Contraintes

Les contraintes correspondent essentiellement à des disjonctions, des règles et des précédences temporelles, et à leurs combinaisons. Les contraintes temporelles sont propagées par consistance de borne [18]. Les contraintes s'appliquent à toutes les actions $a \in O$ et toutes les préconditions $p \in pre(a)$; nous utilisons la notation $\delta(a, a')$ pour $dur(a) + dist(a, a')$.

- **Bornes :** pour tout $a \in O$

$$T(Start) + dist(Start, a) \leq T(a)$$

$$T(a) + dist(a, End) \leq T(End)$$

- **Préconditions :** le support a' d'une précondition p de a doit précéder a d'une quantité qui dépend de $\delta(a', a)$

$$T(a) \;\geq \min_{a' \in [D(S(p,a)]} (T(a') + \delta(a', a))$$

$$T(a') + \delta(a', a) > T(a) \rightarrow S(p, a) \neq a'$$

- **Contraintes de liens causaux :** pour tout $a \in O$, $p \in pre(a)$ et a' qui e-retire p, a' précède $S(p, a)$ ou suit a

$$T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p, a)$$

$$\vee \;\; T(a) + \delta(a, a') \leq T(a')$$

- **Contraintes de mutex :** pour deux actions effet-interférentes a et a'[1]

$$T(a) + \delta(a, a') \leq T(a') \;\vee\; T(a') + \delta(a', a) \leq T(a)$$

- **Contraintes de support :** $T(p, a)$ et $S(p, a)$ sont reliées par

$$S(p, a) = a' \rightarrow T(p, a) = T(a')$$

$$T(p, a) \neq T(a') \rightarrow S(p, a) \neq a'$$

$$\min_{a' \in D[S(p,a)]} T(a') \;\leq\; T(p, a) \;\leq\; \max_{a' \in D[S(p,a)]} T(a')$$

[1]Deux actions sont effet-interférentes dans CPT quand l'une retire un ajout de l'autre, et aucune ne e-retire une précondition de l'autre.

2.4 Schéma de branchement

Comme en planification POCL, le branchement dans CPT fonctionne en sélectionnant et en réparant itérativement les défauts d'états non terminaux σ, en effectuant un retour arrière en cas d'inconsistance. Un état σ est décrit par les variables, leurs domaines, et les contraintes qui les lient. L'état initial σ_0 contient les variables, les domaines et les contraintes ci-dessus, ainsi que la *contrainte d'horizon* $T(End) = B$ où B est la borne courante sur le makespan, qui est dans le cas optimal initialisée à une valeur minimale, puis incrémentée jusqu'à l'obtention d'un plan. Un état est inconsistant quand une variable non conditionnelle se trouve avec un domaine vide, tandis qu'un état consistant σ sans défaut est un *état but* duquel un plan valide P de borne B peut être extrait en fixant la date de début des actions du plan à leur borne inférieure.

La définition des défauts est celle que l'on rencontre en planification POCL, exprimée à l'aide des variables temporelles et des variables de support, avec l'ajout des menaces de mutex.

- **Menaces de Support :** a' *menace un support* $S(p, a)$ quand les deux actions a et a' sont dans le plan partiel courant, a' e-retire p, et ni $T_{min}(a') + dur(a') \leq T_{min}(p, a)$ ni $T_{min}(a) + dur(a) \leq T_{min}(a')$ ne sont vérifiées,
- **Conditions Ouvertes :** $S(p, a)$ est une *condition ouverte* quand $|D[S(p, a)]| > 1$ est vérifiée pour une action a du plan,
- **Menace de Mutex :** a et a' constituent une *menace de mutex* quand les deux actions sont dans le plan, elles sont effet-interférentes, et ni $T_{min}(a) + dur(a) \leq T_{min}(a')$ ni $T_{min}(a') + dur(a') \leq T_{min}(a)$ ne sont vérifiées.

Les défauts sont sélectionnés pour réparation dans l'ordre suivant : d'abord les Menaces de Support (MS), puis les Conditions Ouvertes (OC), et enfin les Menaces de Mutex (MM). Les MS et MM sont réparées en introduisant et propageant des contraintes de précédence, tandis que les OC sont réparées en choisissant un support, comme en planification POCL classique. On trouvera plus de détails dans [26].

3 eCPT

CPT est un planificateur temporel optimal avec de bonnes performances, compétitif avec les meilleurs planificateurs parallèles basés sur SAT quand les actions ont une durée uniforme. De plus, pour la planification non optimale, CPT possède l'avantage suivant : la taille du codage n'augmente pas avec la borne. En effet, la borne dans CPT est représentée entièrement par la contrainte $T(End) = B$, ce qui affecte le domaine des variables mais pas leur nombre. En dépit de ceci toutefois, CPT n'est pas un bon planificateur non optimal, car comme les planificateurs SAT et CSP il dépend énormément du pouvoir d'élagage de la contrainte d'horizon qui devient inefficace pour des valeurs élevées de B.

La figure 1 montre les performances de CPT pour le problème Tower-n, pour différentes valeurs de n et plusieurs horizons B. Tower-n est le problème de l'assemblage d'une tour de n cubes initialement posés sur la table. Il s'agit d'un problème trivial pour un humain, mais comme on le montre dans [26], il est loin d'être trivial pour les planificateurs optimaux. CPT cependant, résout ce problème optimalement et sans retour arrière pour n'importe quelle valeur de n. Mais comme le montre la figure, le temps de calcul (et les retours arrière) augmente beaucoup quand l'horizon B dépasse la borne optimale ; et pour des valeurs élevées de B, CPT ne peut résoudre ces problèmes après des milliers de secondes et de retours arrière.

La figure montre aussi le comportement de eCPT : tandis que les performances de CPT se dégradent avec l'augmentation de la borne B, celles de eCPT restent stables et sans retour arrière pour les différentes valeurs de B. eCPT exploite la flexibilité que procure la formulation basée sur la programmation par contraintes qui sous-tend CPT, l'étendant par des inférences qui ne dépendent pas autant de la contrainte d'horizon, et qui produisent un comportement sans retour arrière parmi un large spectre de domaines simples. Dans cette section nous nous concentrons sur ces inférences.

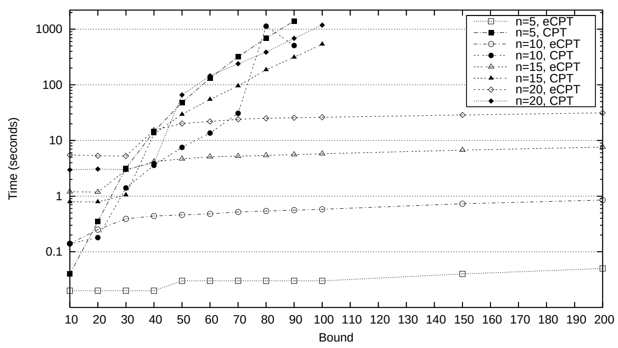

FIG. 1 – Performances de CPT et eCPT sur Tower-n pour divers nombre n de cubes et bornes B. Les courbes qui divergent correspondent à CPT ; celles qui restent stables à eCPT.

3.1 Supports impossibles

De nombreux supports peuvent être éliminés par pré-traitement, permettant d'éviter pendant la recherche des états ne pouvant mener à une solution. Par exemple, l'action $a' = putdown(b1)$ ne peut jamais supporter la précondition $p = handempty$ d'une action comme $a = unstack(b1, b3)$. La raison en est que l'action a a une autre précondition $p' = on(b1, b3)$ qui est e-retirée par a' (fausse après a') et qui devrait donc être produite à nouveau par une autre action b précédant a.

Plus généralement, soit $dist(a', p, a)$ une borne inférieure sur l'écart temporel entre les actions a' et a dans tout plan valide dans lequel *a' est un support de la précondition p de a*. Dans certains cas, lors du prétraitement, on peut montrer que $dist(a', p, a) = \infty$; et ainsi, que a' peut être sans danger retirée du domaine de la variable $S(p, a)$ encodant le support de la précondition p de a.

Ceci se produit actuellement quand une précondition p' de a n'est pas *atteignable* depuis la situation initiale qui inclut tous les atomes exceptés ceux qui sont e-retirés par a' et où *les actions qui ajoutent ou retirent p sont exclues*. La raison pour cette exclusion est que si a' supporte la précondition p de a, alors on peut déduire qu'aucune action ajoutant ou retirant p ne peut se produire entre a' et a (la première partie correspond à la condition de systématicité [19]). Par atome atteignable, nous entendons qu'il fait partie du graphe de planification relaxé : le graphe de planification dans lequel les retraits des actions sont exclus [10].

Ce test simple interdit l'action $putdown(b_1)$ comme support possible de la précondition *handempty* de l'action $unstack(b_1, b_3)$, l'action $stack(b_1, b_3)$ comme support possible de la précondition $clear(b_1)$ de $pickup(b_1)$, etc.

3.2 Supports uniques

Nous disons qu'une action *consomme* un atome p quand elle requiert p en précondition et qu'elle le retire. Par exemple, les actions $unstack(b_3, b_1)$ et $pickup(b_2)$ consomment toutes les deux l'atome *handempty*. Dans ce cas-là, si ces actions sont introduites dans le plan, on peut montrer que leur précondition commune p doit avoir des supports différents. En effet, si une action a retire une précondition de a', et a' retire une précondition de a, alors a et a' sont incompatibles et ne peuvent pas se recouvrir dans le temps selon la sémantique sur les actions concurrentes. Ainsi, soit a doit précéder a', soit a' doit précéder a, et dans tous les cas l'atome p doit être produit au moins deux fois : une fois pour la première action, et une deuxième fois pour la seconde action. La contrainte $S(p, a) \neq S(p, a')$ pour les paires d'actions a et a' qui consomment p assure que, quand une des variables support $S(p, a)$ ou $S(p, a')$ est instanciée par une valeur b, b est immédiatement retirée du domaine de l'autre variable.

3.3 Amélioration des distances

Les distances $dist(a, a')$ pré-calculées pour toutes les paires d'action a et a' fournissent une borne inférieure sur l'écart temporel entre la fin de a et le début de a'. Dans certains cas, cette borne inférieure peut être facilement améliorée, induisant des inférences plus puissantes. Par exemple, la distance entre les actions $putdown(b_1)$ et $pickup(b_1)$ est égale à 0, car il est actuellement possible d'exécuter une action après l'autre. Cependant l'action $putdown(b_1)$ suivie par $pickup(b_1)$ est utile seulement si une autre action utilisant les effets de la première apparaît entre ces deux actions, comme dans le cas où le cube b_1 est sur le cube b_2 mais doit être déplacé sur le cube se trouvant sous b_2.

Nous dirons qu'une action a *annule* une action a' quand 1) tous les atomes ajoutés par a' sont e-retirés par a, et 2) tous les atomes ajoutés par a sont des préconditions de a'. Ainsi, quand a annule a', la séquence a', a ne produit rien qui n'était déjà vrai avant a'. Par exemple, $pickup(b_1)$ annule l'action $putdown(b_1)$.

Quand une action a annule une action a', et qu'une précondition p de a est produite par a' (i.e., p est ajoutée par a' et est mutex avec une précondition de a'), la distance $dist(a', p, a)$ introduite ci-dessus devient égale à ∞ si toutes les actions qui utilisent un ajout de a' e-retirent p. Dans ce cas, comme précédemment, l'action a' peut être exclue du domaine de la variable $S(p, a)$. Sinon, la distance $dist(a', a)$ peut être augmentée de la valeur $\min_b[dist(a', b) + dist(b, a)]$ avec b appartenant à l'ensemble des actions différentes de a et a' qui soit utilisent un ajout de a' mais ne e-retirent pas p, soit n'utilisent pas obligatoirement un ajout de a' mais ajoutent p (car a' peut être suivie par une action c précédant a qui e-retire p, mais seulement s'il y a une autre action b entre c et a qui ré-établit p).

Ainsi, la distance entre les actions $putdown(a)$ et $pickup(a)$ dans le domaine Blocks est augmentée de 2, la distance entre $sail(a, b)$ et $sail(b, a)$ dans le domaine Ferry est augmentée de 1, etc. L'effet réel de ce procédé est identique à celui de l'élimination de cycles de taille deux dans une recherche heuristique standard. L'élimination de cycles de taille plus importante, cependant, semble plus difficile dans le cadre POCL, bien que des idées similaires puissent éventuellement être utilisées pour éliminer certaines séquences d'actions commutatives.

3.4 Précédences qualitatives

A la différence des planificateurs POCL traditionnels, CPT raisonne avec des *précédences temporelles* de la forme $T(a) + \delta(a, a') \leq T(a')$ plutôt qu'avec des *précédences qualitatives*. CPT est un planificateur temporel et un tel choix paraît naturel d'après la représentation utilisée. Toutefois, le mécanisme de propagation de contraintes, la consistance de bornes, est incomplet ; et dans un contexte de planification, est souvent trop faible. En particulier, la consistance de bornes ne capture pas la *transitivité :* par exemple pour les contraintes temporelles $A < B$ et $B < C$, elle n'infère pas $A < C$. En effet si le domaine initial des variables A, B, et C est $[1, \ldots, 100]$, la consistance de bornes réduit les domaines à $[1, \ldots, 98]$, $[2, \ldots, 99]$, et $[3, \ldots, 100]$ respectivement, ce qui ne rend pas $A < C$ vraie pour toutes les combinaisons de valeurs. La transitivité, cependant, est importante en planification ; ainsi, eCPT comprend, en plus des précédences temporelles, des précédences qualitatives de la forme $a \prec a'$ *qui ne sont pas limitées aux actions a et a' présentes dans le plan partiel courant.* Ces précédences qualitatives sont obtenues à chaque fois qu'une précédence temporelle est rendue vraie ou inférée.[2] La fermeture transitive de cette relation est systématiquement mise à jour. Quand une nouvelle précédence qualitative $a \prec a'$ est trouvée, la fermeture transitive est calculée de la façon suivante : si a appartient au plan partiel courant, alors pour tout a'' telle que $a'' \prec a$, $a'' \prec a'$ est enregistrée ; et si a' appartient au plan, alors pour tout a'' telle que $a' \prec a''$, $a \prec a''$ est enregistrée. Les mêmes mises à jour sont exécutées incrémentalement pour une relation existante $a \prec a'$ avec a ou a' pas encore dans le plan, dès qu'une des deux actions a ou a' est ajoutée au plan.

Deux règles d'inférence utilisent ces précédences qualitatives pour réduire encore le domaine des variables de support :

- pour une action a' dans le plan qui ajoute une précondition p d'une action a : si $a \prec a'$ alors $S(p, a) \neq a'$
- pour une action a' qui ajoute une précondition p d'une action a et une action b dans le plan qui e-retire p : si $a' \prec b$ et $b \prec a$, alors $S(p, a) \neq a'$

3.5 Actions obligatoires

Comme tous les planificateurs POCL, CPT commence la recherche avec un plan partiel contenant

[2] Les précédences temporelles sont rendue vraies comme résultat d'une décision de branchement correspondant à une menace de support ou de mutex, et sont inférées quand l'une des parties de la disjonction d'une menace de supports ou de mutex devient fausse.

uniquement les deux actions *Start* et *End*. Dans de nombreux cas cependant, il est possible d'inférer facilement que certaines autres actions doivent aussi nécessairement appartenir au plan. Par exemple, si un cube b_1 doit être déplacé mais se trouve sous un cube b_2 se trouvant lui-même sous un cube b_3, alors les actions $unstack(b_3, b_2)$ et $unstack(b_2, b_1)$ devront être utilisées quelque part ; de plus, la première doit précéder la deuxième. Dans eCPT nous identifions de telles actions nécessaires et un ordre partiel entre elles dans la phase de pré-traitement, suivant l'idée des *points de passage obligatoires* introduite dans [22], sous la forme présentée dans [30]. Une action a est une *action obligatoire* si l'action *End* n'est pas *atteignable* quand l'action a est exclue du domaine (comme mentionné plus haut, une action est atteignable quand elle est introduite dans le graphe de planification relaxé). De plus, une action obligatoire a *précède* une action obligatoire b, si b n'est pas atteignable quand l'action a est exclue. Les actions obligatoires et un ordre partiel entre elles sont calculés dans l'étape de pré-traitement et sont inclus dans l'état initial du planificateur en même temps que les actions *Start* et *End*. Ceci implique le calcul de $|O|$ graphes de planification relaxés, un pour chaque action du domaine.

3.6 Branchement et heuristiques

eCPT utilise le même schéma de branchement que CPT et dans le même ordre : il branche d'abord sur les menaces de support, puis sur les conditions ouvertes, et enfin sur les menaces de mutex. Les heuristiques pour le choix des menaces de support et les conditions ouvertes sont cependant légèrement différentes.

Les menaces de support $\langle a', S(p, a)\rangle$ sont choisies dans eCPT en minimisant $T_{min}(a)$, en cassant les égalités en minimisant d'abord $T_{max}(p, a)$, et ensuite le critère d'écart temporel utilisé dans CPT. Les conditions ouvertes $S(p, a)$ sont choisies en minimisant $T_{max}(p, a)$, en cassant les égalités en minimisant $slack(a', a) = T_{max}(a) - (T_{min}(a') + \delta(a', a))$ où a' produit p pour a ($a' \in D[S(p, a)]$), en minimisant $T_{min}(a')$. La contrainte propagée dans le second cas est $S(p, a) = a'$, et en cas d'échec, $S(p, a) \neq a'$.

4 Résultats expérimentaux

Nous analysons eCPT selon trois dimensions : comme un « planificateur aisé » capable de résoudre des problèmes sans retour arrière, comme un planificateur non optimal en le comparant avec le planificateur récent et très performant FF [10], comme un planificateur temporel optimal en le comparant avec CPT et comme un planificateur parallèle optimal en le comparant avec SATPLAN04 [14]. Les instances et les domaines sont ceux des deuxième et troisième compétitions internationales de planification IPC2 et IPC3 [1, 17]. Les résultats ont été obtenus sur un Pentium IV à 2.8Ghz, avec 1Go de RAM, sous Linux. La limite maximale en temps de calcul pour chaque problème a été fixée à 30 minutes. La borne B sur le makespan pour les versions aisée/non optimale de eCPT est fixée à 200, toutes les actions étant considérée comme ayant une durée unitaire (planification parallèle type GRAPHPLAN).

La table 1 montre pour chaque domaine le nombre total d'instances, le nombre d'instances résolues par eCPT, le nombre d'instances résolues sans retour arrière (et entre parenthèses, le nombre maximum de retours arrière pour les problèmes en comportant), et le nombre maximum de nœuds générés (en planification de type POCL, ce nombre est différent du nombre total d'actions du plan même dans le cas où il n'y a pas de retour arrière). Pour illustrer ce propos, le nombre d'instances résolues et le nombre maximum de nœuds générés par FF sont aussi rapportés. *Comme on peut le voir, eCPT résout 339 instances sur 350, 336 sans retour arrière, dont toutes celles de Blocks, Ferry, Logistics, Gripper, Miconic, Rovers et Satellite.* Les 11 instances qui ne sont pas résolues par eCPT sont dues à des limitations de mémoire inhérentes au langage Claire, et non à des retours arrière ou au dépassement de la limite de temps. Nous pensons qu'il s'agit d'un résultat remarquable, et étonnant ; jusqu'il y a peu, ces instances étaient considérées comme difficiles. eCPT résout actuellement 3 instances de plus que FF sur cette série de problèmes, eCPT ayant des résultats plus robustes pour Blocks et DriverLog, et FF pour Depots et Zeno. Pour le dernier domaine de IPC3, Freecell, FF résout plus d'instances que eCPT qui n'exhibe plus de comportement sans retour arrière. Ce domaine, toutefois, procure aussi des difficultés à FF dues en particulier à la présence d'impasses lors de la recherche [9].

Toutes les nouvelles règles d'inférence ne sont pas nécessaires pour générer un comportement sans retour arrière dans tous les domaines : pourtant, les supports impossibles apparaissent comme critiques pour Depots, l'amélioration des distances pour Depots, DriverLog et Ferry, les précédences qualitatives pour tous les domaines excepté Blocks, les actions obligatoires pour Blocks. De plus, des disjonctions de règles apparaissent aussi comme critiques. Par exemple, si les précédences qualitatives ou les supports uniques peuvent être supprimés individuellement pour Blocks sans générer de retour arrière, la suppression des deux en produit. Enfin, les nouvelles heuristiques sont absolument nécessaires pour tous les domaines (bien que certaines variantes donnent aussi de bons résultats, eCPT se montrant particulièrement robuste).

	nombre de pbs	eCPT			FF	
		résolus	sans r.a. (max r.a.)	max nds	résolus	max nds
blocks	50	50	50 (0)	275	42	146624
depots	20	18	16 (4)	285	19	166141
driver	20	17	16 (5)	176	15	4657
ferry	50	50	50 (0)	1176	50	201
gripper	50	50	50 (0)	201	50	200
logistics	50	50	50 (0)	273	50	2088
miconic	50	50	50 (0)	131	50	76
rovers	20	20	20 (0)	207	20	3072
satellite	20	20	20 (0)	249	20	5889
zeno	20	14	14 (0)	70	20	933

TAB. 1 – eCPT vs. FF : résultats sur divers domaines « simples », montrant le nombre de problèmes résolus, ceux résolus sans retour arrière (nombre max de r.a.), et nombre max de nœuds générés.

	temps CPU (sec.)		actions		nœuds	
	eCPT	FF	eCPT	FF	eCPT (r.a.)	FF
bw-ipc48	59,51	-	74	-	281 (0)	-
bw-ipc49	78,37	-	80	-	282 (0)	-
bw-ipc50	85,09	0,02	88	86	235 (0)	195
log-ipc48	50,56	0,20	164	142	261 (0)	515
log-ipc49	51,54	0,50	176	171	273 (0)	1252
log-ipc50	50,39	0,43	161	154	245 (0)	1147
depots06	66,23	-	68	-	160 (0)	-
depots07	1,27	0,01	28	25	68 (0)	142
depots08	13,13	579,89	75	43	206 (0)	172478
driver14	5,40	0,09	48	45	75 (0)	1209
driver15	39,91	0,03	69	44	130 (0)	161
driver16	147,15	-	107	-	163 (5)	-

TAB. 2 – eCPT vs. FF : détails sur quelques instances.

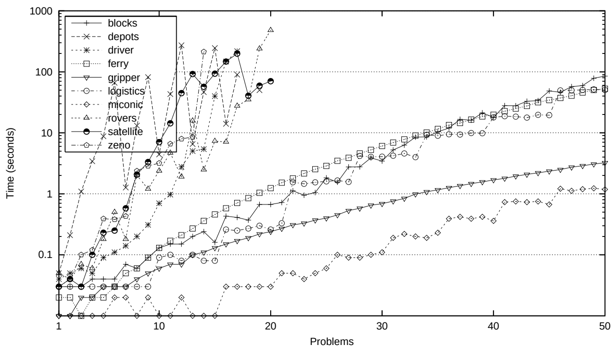

FIG. 2 – Temps de calcul de eCPT sur tous les domaines.

Les informations sur le temps de calcul de eCPT sur les divers domaines sont aussi représentées dans la Figure 2, la Table 2 donnant plus de détails pour quelques instances sélectionnées en comparaison avec FF. Comme on peut le voir, le temps de calcul de eCPT se comporte bien, bien qu'il ne soit pas compétitif avec celui de FF (à part pour quelques instances de Depots) : FF génère beaucoup plus de nœuds mais le fait plus rapidement. La qualité du plan mesurée en nombre d'actions est meilleure pour FF dans les domaines Logistics ou DriverLog, ce qui a probablement un lien avec le fait que eCPT calcule des plan parallèles.

Finalement, la Figure 3 compare eCPT et CPT comme des *planificateurs temporels optimaux* et la Figure 4 compare eCPT et SATPLAN04 comme des *planificateurs parallèles optimaux* sur toutes les instances. eCPT résout 207 instances sur 350, tandis que CPT en résout 179 et SATPLAN04 en résout 180. eCPT génère beaucoup moins de nœuds que CPT, souvent avec un facteur exponentiel (bien que le coût additionnel des inférences supplémentaires dégrade parfois un peu les performances), *le rendant de loin le planificateur tem-*

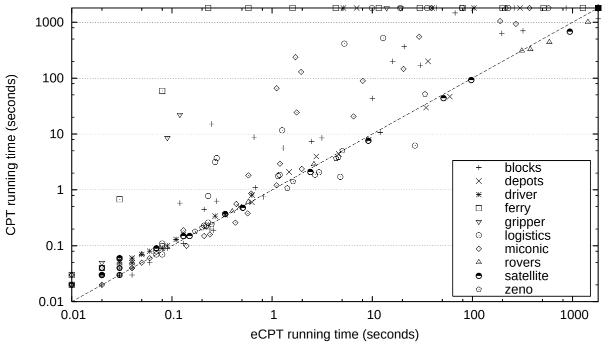

FIG. 3 – eCPT vs. CPT pour la planification optimale. eCPT est plus rapide que CPT dans la zone située au dessus de la diagonale, et moins rapide au dessous. Les problèmes sur la bordure supérieure ne sont pas résolus par CPT, ceux sur la bordure de droite ne le sont pas par eCPT.

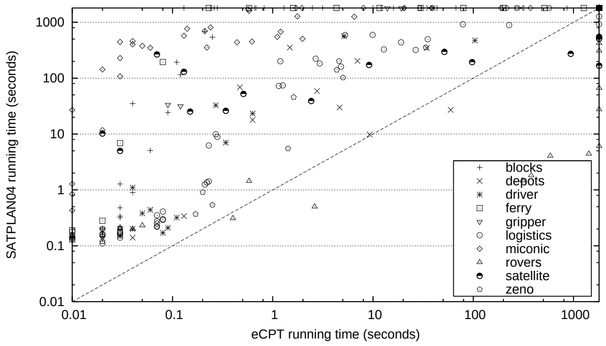

FIG. 4 – eCPT vs. SATPLAN04 pour la planification optimale. eCPT est plus rapide que SATPLAN04 dans la zone située au dessus de la diagonale, et moins rapide au dessous. Les problèmes sur la bordure supérieure ne sont pas résolus par SATPLAN04, ceux sur la bordure de droite ne le sont pas par eCPT.

porel optimal actuel le plus efficace.

Comme un *planificateur non optimal*, eCPT résout 339 instances sur 350, tandis que CPT résout seulement 66 instances en dépassant souvent 1000 secondes, produisant la plupart du temps des plans comportant beaucoup trop d'actions.

5 Discussion

Nous pensons que le développement de planificateurs aisés est une tâche particulièrement sensée et motivée, qui pourrait contribuer non seulement au développement de planificateurs automatiques plus robustes, mais aussi à une meilleure compréhension des mécanismes de planification de l'être humain. Dans ce travail nous avons montré que ce but peut être atteint avec le planificateur temporel CPT, sur un grand nombre de domaines, par l'addition de quelques mécanismes d'inférence simples et généraux. Les inférences sont introduites pour éliminer les retours arrière de la recherche, sans nécessairement la rendre plus rapide que les meilleurs planificateurs. Elles ont été obtenues en observant le comportement de CPT sur divers domaines, pour prendre en compte des raisonnements simples qui manquaient et que nous pensions responsables de retours arrière évitables. Le fait que cette analyse de bas niveau et locale soit possible, et que les résultats soient facilement incorporables dans le planificateur, est clairement un atout de la formulation en programmation par contraintes, qui procure ainsi la capacité d'utiliser des analyses (humaines) *dépendantes du domaine* pour améliorer les performances d'un planificateur *indépendant du domaine*. Nous avons aussi évalué expérimentalement le planificateur qui en résulte, eCPT, comme un planificateur non optimal et optimal, et avons montré des gains importants par rapport à CPT et SATPLAN04.

La découverte que quelques *règles d'inférence* est tout ce qu'il faut pour éliminer les retours arrière de la recherche dans des domaines considérés peu de temps avant comme difficiles pour les planificateurs, partage des similarités avec l'*observation expérimentale* faite dans [4] qu'une *fonction heuristique* simple et indépendante du domaine puisse en pratique guider efficacement la recherche dans de nombreux domaines, une idée exploitée par la suite dans plusieurs planificateurs performants.

Les deux procédés sont cependant différents : les fonctions heuristique fournissent des informations *numériques* pour ordonner les alternatives, tandis que les règles d'inférence produisent des informations *structurelles* permettant d'éliminer des alternatives. Nous pensons qu'il est possible de *prouver* que certains domaines ont un comportement sans retour arrière pour eCPT, et identifier ainsi de nouvelles classes de problèmes traitables. Les classes existantes, comme définies dans [2, 5], restent assez étroites, et ne prennent pas en compte les benchmarks existants [8]. Dans nos travaux futurs, nous envisageons de rechercher les conditions minimales de eCPT qui éliminent les retours arrière sur les domaines étudiés, d'autres inférences simples actuellement non prises en compte qui produisent des retours arrière dans certains domaines, et la façon dont on pourrait rendre l'implémentation plus efficace. Par exemple, il arrive que la consistance de bornes détecte l'inconsistance de deux contraintes $A < B$ et $B < A$ en passant en revue toutes les valeurs des domaines, ce qui est source d'inefficacité dans eCPT quand les contraintes temporelles impliquent des actions qui ne sont pas encore dans le plan (pour les actions présentes dans le plan, ces inconsistance sont détectées efficacement au moyen des précédences qualitatives).

Références

[1] F. Bacchus. The 2000 AI Planning Systems Competition. *Artificial Intelligence Magazine*, 22(3) :47–56, 2001.

[2] C. Bäckström and B. Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4) :625–655, 1995.

[3] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995.

[4] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, pages 714–719. MIT Press, 1997.

[5] R. Brafman and C. Domshlak. Structure and complexity of planning with unary operators. *JAIR*, 18 :315–349, 2003.

[6] M. B. Do and S. Kambhampati. Solving planning-graph by compiling it into CSP. In *Proceedings of AIPS-00*, pages 82–91, 2000.

[7] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proceedings of European Conference of Planning (ECP-01)*, pages 121–132, 2001.

[8] M. Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2) :219–262, 2003.

[9] J. Hoffmann. Local search topology in planning benchmarks : An empirical analysis. In *Proc. IJCAI-2001*, pages 453–458, 2001.

[10] J. Hoffmann and B. Nebel. The FF planning system : Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 2001 :253–302, 2001.

[11] A. Jonsson, P. Morris, N. Muscettola, and K. Rajan. Planning in interplanetary space : Theory and practice. In *Proceedings of AIPS-2000*, pages 177–186, 2000.

[12] D. Joslin and M. E. Pollack. Is "early commitment" in plan generation ever a good idea ? In *Proceedings of AAAI-96*, pages 1188–1193, 1996.

[13] S. Kambhampati, C. Knoblock, and Q. Yang. Planning as refinement search : A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2) :167–238, 1995.

[14] H. Kautz. SATPLAN04 : Planning as satisfiability. In *4th. Int. Planning Competition Booklet (ICAPS-04)*, 2004.

[15] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In T. Dean, editor, *Proceedings of IJCAI-99*, pages 318–327. Morgan Kaufmann, 1999.

[16] P. Laborie and M. Ghallab. Planning with sharable resources constraints. In C. Mellish, editor, *Proceedings of IJCAI-95*, pages 1643–1649. Morgan Kaufmann, 1995.

[17] D. Long and M. Fox. The 3rd international planning competition : Results and analysis. *Journal of Artificial Intelligence Research*, 20 :1–59, 2003.

[18] K. Marriot and P. Stuckey. *Programming with Constraints.* MIT Press, 1999.

[19] D. McAllester and D. Rosenblitt. Systematic non-linear planning. In *Proceedings of AAAI-91*, pages 634–639, Anaheim, CA, 1991. AAAI Press.

[20] X. L. Nguyen and S. Kambhampati. Reviving partial order planning. In *Proceedings of IJCAI-01*, pages 459–466, 2001.

[21] J. S. Penberthy and D. S. Weld. Temporal planning with continous change. In *Proceedings of AAAI-94*, pages 1010–1015, 1994.

[22] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Recent Advances in AI Planning. 6th European Conference on Planning (ECP-01)*, pages 37–48, 2001.

[23] J. Rintanen. A planning algorithm not based on directional search. In *Proceedings of KR'98*, pages 617–624. Morgan Kaufmann, 1998.

[24] D. Smith and D. S. Weld. Temporal planning with mutual exclusion reasoning. In *Proceedings of IJCAI-99*, pages 326–337, 1999.

[25] P. Van Beek and X. Chen. CPlan : a constraint programming approach to planning. In *Proceedings of National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590. AAAI Press/MIT Press, 1999.

[26] V. Vidal and H. Geffner. Branching and pruning : An optimal temporal POCL planner based on constraint programming. In *Proceedings of AAAI-2004*, pages 570–577, 2004.

[27] V. Vidal and H. Geffner. Branching and pruning : An optimal temporal POCL planner based on constraint programming (long version). Technical report, 2004.

[28] V. Vidal and H. Geffner. Un planificateur temporel optimal basé sur la programmation par contraintes. In *Actes de JNPC-2004*, pages 347–362, 2004.

[29] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4) :27–61, 1994.

[30] L. Zhu and R. Givan. Heuristic planning via roadmap deduction. In *4th. Int. Planning Competition Booklet (ICAPS-04)*, 2004.