Un planificateur temporel optimal basé sur la programmation par contraintes

Vincent Vidal
CRIL - Université d'Artois
rue de l'université
62307 Lens Cedex, France
email: `vidal@cril.univ-artois.fr`

Héctor Geffner
ICREA & Universitat Pompeu Fabra
Paseo de Circunvalacion 8
08003 Barcelona, Espagne
email: `hector.geffner@upf.edu`

Résumé

Une des principales caractéristiques des planificateurs optimaux modernes tels Graphplan et Blackbox est leur capacité d'élaguer de larges parts de l'espace de recherche. D'un autre côté, les planificateurs plus anciens effectuant une recherche dans les espaces de plans partiels offrent des possibilités d'exploration de l'espace de recherche intéressantes mais ne disposent pas de mécanismes d'élagage aussi performants. Nous proposons dans cet article une formulation de la planification temporelle indépendante du domaine basée sur la programmation par contraintes, qui combine avec succès les possibilités d'exploration de l'espace de recherche de la planification dans les espaces de plans partiels avec des règles d'élagage puissantes et saines. La principale innovation de cette formulation est la capacité de raisonner autour des supports, des relations de précédence et des liens causaux, en faisant intervenir les actions qui n'appartiennent pas encore à un plan partiel. Des expérimentations avec de nombreux benchmarks montrent que le planificateur temporel optimal que nous avons réalisé est beaucoup plus performant que les planificateurs temporels optimaux actuels, et est compétitif avec les meilleurs planificateurs parallèles dans le cas particulier où toutes les actions ont la même durée.

1 Introduction

La recherche de plans optimaux en planification de tâches, comme la recherche de solutions optimales pour beaucoup de problèmes d'optimisation combinatoire, peut être comprise selon deux dimensions : les *règles de branchement* utilisées pour compléter des solutions partielles, et les *règles d'élagage* utilisées pour en éliminer. La plupart des méthodes de recherche de plans peuvent être définies en ces termes. Les planificateurs optimaux fonctionnant par recherche dans les espaces d'états, par exemple, effectuent un branchement par progression ou régression d'états, et élaguent en comparant une estimation du coût du plan partiel avec une borne donnée [HG00]. Les planificateurs optimaux basés sur SAT et CSP, d'un autre côté, effectuent un branchement en choisissant une variable et en essayant chacune de ses valeurs ; puis en élaguant les branches de l'arbre de recherche et les valeurs des domaines qui mènent à une inconsistance [KS99, DK00]. L'élagage est une opération clé dans les deux cas : dans le premier, c'est le résultat de l'utilisation explicite d'heuristiques ; dans le second, de mécanismes de propagation sur des contraintes et d'heuristiques encodées dans le graphe de planification [BF95]. Cette puissance d'élagage distingue les planificateurs modernes tel Graphplan de ses prédécesseurs (optimaux ou non). En effet, la principale limitation des planificateurs traditionnels fonctionnant par recherche dans les espaces de plans partiels (planificateurs de type POCL : «Partial Order Causal Link»), qui fournissent des règles de branchement intéressantes, est qu'ils ne disposent pas de règles d'élagage comparables. La conséquence en est que les plans partiels ne pouvant conduire à un plan solution sont découverts tardivement, produisant une explosion rapide de l'espace de recherche.

Cependant, grâce à son pouvoir d'expression, la planification POCL reste une technique attirante ; en particulier pour la planification temporelle optimale [SFJ00]. Le défi est de combler l'écart de performances qui sépare les planificateurs POCL des planificateurs modernes tout en conservant les garanties d'optimalité. Dans cet article, nous relevons ce défi, en améliorant un planificateur POCL temporel avec des règles d'élagage puissantes et saines basées sur une formulation par programmation par contraintes qui intègre les heuristiques existantes avec des règles de propagation pour raisonner sur les supports, les précédences, et les liens causaux, de manière nouvelle et originale. Les expérimentations montrent que le planificateur qui en résulte est plus rapide que les planificateurs temporels optimaux et est compétitif avec les planificateurs parallèles dans le cas particulier où les actions ont une durée uniforme.

Quelques remarques avant de commencer. D'abord, la formulation et l'implémentation que nous présentons ne sont pas complètement générales : nous ne pouvons calculer que des plans dans lesquels chaque action n'apparaît qu'une seule fois ; plans que nous appelons «canoniques» [Gef01]. Les plans canoniques sont ainsi à mi-chemin entre la planification temporelle classique et l'ordonnancement de tâches : dans ce dernier chaque action est exécutée *exactement une fois*, en planification canonique elle est exécutée *au plus une fois*, et en planification temporelle classique, elle peut être exécutée *un nombre quelconque de fois*. Comme nous le verrons, dans beaucoup de benchmarks pour la planification séquentielle, parallèle ou temporelle, les plans optimaux sont canoniques ; bien que ce ne soit pas toujours le cas. Nous discuterons brièvement plus loin comment lever cette hypothèse de départ et présenterons quelques résultats préliminaires que nous avons obtenus dans cette direction.

L'intégration de fonctions d'évaluation heuristique dans la planification POCL a été commencé récemment dans [NK01, YS03]. Cependant, ces approches ne s'intéressent pas à la génération de plans optimaux. Ici, nous utilisons certaines idées de [NK01] comme l'utilisation de mutex structurels pour étendre la notion de menace dans la planification POCL, et l'utilisation de contraintes disjonctives pour exprimer la résolution possible des menaces. Les planificateurs POCL temporels utilisant des mécanismes de propagation de contraintes comprennent par exemple IxTeT [LG95] et RAX [JMMR00]. Ces planificateurs sont plus expressifs que le notre (en particulier, l'utilisation des ressources), mais leurs mécanismes d'élagage sont moins efficaces car ils raisonnent essentiellement avec les actions qui sont dans le plan partiel courant seulement. Ceci se passe de manière similaire avec les formulations de la planification POCL par CSP Dynamiques [JP96].

Une approche antérieure de la planification basée sur la programmation par contraintes pour plusieurs domaines *spécifiques* est donnée dans [VC99]. Nous utilisons quelques éléments de cette formulation comme l'utilisation de *distances* de diverses sortes ; mais notre approche est indépendante du domaine. Les idées générales sur lesquelles est basé ce travail ont été évoquées dans [Gef01], et une implémentation préliminaire pour la planification parallèle a été reportée dans [PG02].

2 Un aperçu

Afin d'illustrer les capacités du planificateur que nous proposons, considérons le problème TOWER-n dont l'objectif est de construire une tour avec n cubes $b_1, \ldots, b_n$ dans cet ordre, b_1 au sommet, initialement tous sur la table. Le seul plan optimal pour ce problème consiste à prendre chaque cube b_i depuis la table et à l'empiler sur le cube b_{i+1}, dans l'ordre depuis $i = n - 1$ jusqu'à 1. Les mécanismes de raisonnement du planificateur que nous proposons, appelé CPT, conduisent à une solution *par inférence pure et sans recherche*. Ceci est remarquable car les inférences ne sont pas triviales et les planificateurs optimaux ne sont pas très efficaces dans ce domaine (voir la Table 1).

Comment CPT fait-il cela ? D'abord, il infère que chaque sous-but $on(b_i, b_{i+1})$ doit être réalisé par l'action $stack(b_i, b_{i+1})$ et que ces actions doivent être ordonnées séquentiellement, $stack(b_{n-1}, b_n)$ en premier, puis $stack(b_{n-2}, b_{n-1})$, et ainsi de suite. CPT infère ensuite que la première de ces actions ne peut démarrer avant $t = 1$, la seconde pas avant $t = 3$, etc. Puis, après avoir défini la borne inférieure initiale $B = 2(n - 1)$ sur la durée d'exécution du plan et initialisé le temps d'exécution initial de l'action End à cette borne, il infère que le temps initial de toutes les actions doivent être fixés à leur plus basse valeur possible. Il ajoute enfin les actions $pick(b_i)$ à leur temps initial correct par l'effet de propagations supplémentaires qui élaguent les autres supports et temps initiaux possibles.

Problèmes	Temps CPU (sec.)				Durée d'exéc.
	CPT	BBOX	IPP	TP4	du plan
tower-8	0.33	2.95	0.05	17.68	14
tower-9	0.64	7.28	0.11	887.7	16
tower-10	1.01	13.6	0.38	-	18
tower-11	1.69	28.2	2.26	-	20
tower-12	3.61	-	15.35	-	22
tower-13	5.83	-	123.78	-	24
tower-14	9.70	-	-	-	26
tower-15	13.65	-	-	-	28

TAB. 1 – Résultats pour le domaine TOWER-n

La Table 1 montre les résultats de CPT en comparaison avec trois planificateurs modernes : deux planificateurs parallèles optimaux, Blackbox (avec Chaff) [KS99] et IPP [KNHD97], et le planificateur optimal temporel TP4 [HG01]. Bien que la plupart des domaines ne soient pas comme TOWER-n et nécessitent de la recherche, ce domaine illustre la puissance des mécanismes d'inférence de CPT qui réussissent souvent à élaguer considérablement l'espace de recherche.

3 Les ingrédients de base

Le schéma que nous proposons combine des heuristiques, des règles de branchement qui agissent comme celle utilisées en planification POCL, et une recherche de type branch-and-bound dirigée par les contraintes.

3.1 Les heuristiques

Un développement récent et important en planification concerne l'utilisation des *fonctions d'estimation heuristique* extraites automatiquement du codage des problèmes [McD96, BLG97]. Une famille d'heuristiques admissibles dont le calcul est polynomial h^m, $m = 1, 2, \ldots$, pour la planification séquentielle et parallèle est formulée dans [HG00]. Les heuristiques h^m calculent récursivement une approximation *du coût d'un ensemble d'atomes C par le coût du plus coûteux sous-ensemble de taille* m *dans* C. Pour $m = 2$, dans le cas parallèle, h^m est équivalente à l'heuristique encodée dans le graphe de planification. Les heuristiques h^m sont étendues dans [HG01] à l'estimation de la durée d'exécution du plan dans le cadre temporel. Les mesures $h^m_T(C)$ sont des bornes inférieures sur le temps nécessaire pour rendre C vrai depuis l'état initial. Dans CPT ces heuristiques sont utilisées de différentes manières.

3.2 Le schéma de branchement

Le branchement en planification est souvent explicité en terme d'*espace* dans lequel la recherche des plans est faite, avec des planificateurs directionnels effectuant une recherche dans les *espaces d'états*, et des planificateurs non directionnels effectuant une recherche dans les espaces de *plans partiels* [KKY95]. Tous ces planificateurs, cependant, peuvent aussi être vus comme effectuant une recherche dans les espaces de plans, avec les planificateurs directionnels prenant avantage d'une *propriété de décomposition* par laquelle le début ou la fin d'un plan partiel peuvent être représentés par l'état obtenu en faisant régresser le but ou progresser l'état initial. Cette décomposition est impossible dans les approches non directionnelles venant des formulations POCL, SAT, ou CSP. Dans tous les cas, cependant, afin de rechercher efficacement des plans optimaux, il est nécessaire de détecter et élaguer au plus tôt les plans partiels ne pouvant mener qu'à des solutions dont le coût dépasse une borne B donnée. Dans les planificateurs basés sur les états c'est accompli par la comparaison entre la borne B et la valeur donnée par une fonction d'évaluation ; dans les formulations SAT et CSP, c'est fait au moyen des clauses et des contraintes. Les méthodes de planification basées sur la planification POCL manquent de mécanismes d'élagage comparables et ne sont pas aussi efficaces. Des propositions récentes comme [NK01, YS03] étendent la planification POCL avec comme guide des heuristiques non admissibles, mais ne considèrent pas l'optimalité. Nous essayons ici d'obtenir à la fois des performances acceptables et des plans optimaux dans le cadre plus général de la planification temporelle.

3.3 La planification temporelle

Un problème de planification temporelle Strips est un n-uplet $P = \langle A, I, O, G \rangle$ où A est un ensemble d'atomes de base, $I \subseteq A$ et $G \subseteq A$ représentent les situations initiales et finales, et O est l'ensemble d'opérateurs Strips de base, chacun avec listes de précondition, d'ajout et de retrait $pre(a)$, $add(a)$, et $del(a)$, et *durée* $dur(a)$. Comme dans Graphplan, deux actions

a et a' interfèrent quand l'une retire une précondition ou un ajout de l'autre. Nous suivons le modèle temporel simple de [SW99], et définissons un plan valide comme un plan dans lequel les exécutions des actions interférentes ne se recouvrent pas dans le temps. Nous cherchons ici à calculer des plans ayant une *durée d'exécution* minimale. Quand toutes les actions ont une durée uniforme, ce modèle se réduit au modèle standard de la planification parallèle.

4 La planification POCL temporelle

Le branchement dans la planification POCL procède en choisissant un 'défaut' (précondition ouverte ou menace) et en essayant chacune des *réparations* possibles [Wel94, KKY95]. Un état ou plan partiel dans l'espace de recherche correspond à un ensemble d'engagements représentés par un n-uplet $\sigma = \langle Steps, Ord, CL, Open \rangle$, où $Steps$ est l'ensemble des actions du plan, Ord est l'ensemble des contraintes de précédence sur $Steps$, CL est l'ensemble des liens causaux, et $Open$ est l'ensemble des préconditions ouvertes (comme dans la plupart des planificateurs actuels, nous ne travaillons que sur des actions totalement instanciées). Un état est terminal s'il est inconsistant (i.e., l'ordre Ord est inconsistant ou cet état contient un défaut qui ne peut être réparé) ou s'il est un *état but* (il est consistant et ne contient plus de défaut).

L'adaptation du branchement POCL au cadre temporel est assez direct (cf. [LG95]). Nous considérons ici une extension simple obtenue par l'ajout de variables temporelles $T(a)$ représentant le temps initial d'une action $a \in Steps$. Ces variables temporelles ont comme domaine initial $T(Start) = 0$, $T(End) = B$, et $T(a) :: [0, B - dur(a)]$ où B est la borne sur la durée d'exécution du plan ($Start$ et End sont les deux 'fausses' actions utilisées en planification POCL). Les états résultants ont la forme $\sigma = \langle Steps, Ord_T, CL, Open, T(\cdot) \rangle$ où l'ordre de précédence qualitative Ord est remplacé par les variables temporelles $T(a)$, $a \in Steps$, et l'ensemble Ord_T des contraintes temporelles (l'ordre de précédence Ord de la planification POCL classique peut être utilisé mais n'est pas strictement nécessaire).

Comme précédemment, le branchement fonctionne en choisissant un 'défaut' dans un état non terminal σ et en appliquant toutes les réparations possibles. Les *défauts de préconditions ouvertes* $[p]a$ dans σ sont résolus en sélectionnant une action a' qui supporte p, et en ajoutant le lien causal $a'[p]a$ à CL et la contrainte temporelle $T(a') + dur(a') \leq T(a)$ à Ord_T. L'action a' est ajoutée à $Steps$ si $a' \notin Steps$ et dans ce cas une variable $T(a')$ pour a' est créée. De la même façon, les *menaces de liens causaux*, i.e. les situations dans lesquelles une action $a \in Steps$ peut retirer une précondition $p \in del(a)$ dans un lien causal $a_1[p]a_2$ dans CL, sont résolues en ajoutant une des contraintes temporelles $T(a) + dur(a) \leq T(a_1)$ ou $T(a_2) + dur(a_2) \leq T(a)$ à Ord_T. Un état terminal dans l'espace résultant est soit un état comprenant un ensemble inconsistant de contraintes temporelles soit un état avec un ensemble consistant de contraintes temporelles et aucun défaut.

Les contraintes temporelles dans Ord_T forment un problème temporel simple («Simple Temporal Problem (STP)» [DMP91]) dont la consistance peut être testée efficacement en appliquant une forme limitée de propagation de contraintes connue comme la *consistance de bornes* [Lho93], où les bornes minimales et maximales $T_{min}(a)$ et $T_{max}(a)$ des variables $T(a)$ dans les contraintes de la forme $T(a) + dur(a) \leq T(a')$ sont mises à jour par $T_{max}(a) := \min[T_{max}(a), T_{max}(a') - dur(a)]$ et $T_{min}(a') := \max[T_{min}(a'), T_{min}(a) + dur(a)]$ jusqu'à atteindre un point fixe ou que le domaine d'une variable devienne vide.

Avec deux conditions additionnelles, il est possible de vérifier que le schéma de branchement résultant est *sain* et *complet* ; i.e., les états terminaux buts encodent des plans temporels valides P avec une durée d'exécution B où les actions sont exécutées à leur temps initial minimal, et un de ces états terminaux buts est atteint si un tel plan existe.

Les deux conditions requises sont les suivantes. Premièrement, en l'absence d'une relation de précédence qualitative sur les actions comme dans la planification POCL, nous avons besoin de considérer une action a retirant une précondition p dans un lien causal $a_1[p]a_2$ comme une *menace* quand aucune des deux conditions temporelles $T_{min}(a) + dur(a) \leq T_{min}(a_1)$ ou $T_{min}(a_2) + dur(a_2) \leq T_{min}(a)$ n'est vérifiée. La raison en est que les bornes inférieures T_{min} fournissent une solution consistante à un STP si le STP est consistant. Deuxièmement, en accord avec la sémantique, on doit s'assurer que les actions interférentes ne se recouvrent pas dans le temps. Pour vérifier ceci, nous dirons que deux actions interférentes sont *précondition-interférentes* si l'une d'entre elles retire une préconditions d'une autre, et *effet-interférentes* sinon. Il est alors suffisant de brancher sur une deuxième classe de menaces, les *menaces de mutex :* les paires d'actions effet-interférentes a et a' telles que ni $T_{min}(a) + dur(a) \leq T_{min}(a')$, ni $T_{min}(a') + dur(a') \leq T_{min}(a)$ ne sont vérifiés dans σ. De tels défauts sont résolus en ajoutant à Ord_T une des contraintes temporelles $T(a) + dur(a) \leq T(a')$ ou $T(a') + dur(a') \leq T(a)$.

Les planificateurs modernes de type CBI («Constraint-Based Interval») [JMMR00, SFJ00] sont basés sur des idées similaires et sont capables d'utiliser des langages plus expressifs. Cependant, comme pour les planificateurs standard POCL et CSP Dynamiques [JP96], le *problème de performances* suivant reste : éliminer des plans partiel dont le réseau STP n'est pas consistant ne suffit pas pour atteindre les performances des planificateurs modernes. Pour cela, nous avons besoin *de représentations et de méthodes d'inférence plus performantes, capables de prédire que tous les réseaux STP depuis le plan partiel courant jusqu'au but seront inconsistants.*

5 Une formulation par programmation par contraintes

Les performances peu satisfaisantes des planificateurs POCL basés sur les contraintes vient essentiellement de *leur limitation à raisonner uniquement sur les actions du plan courant*. Le plus souvent, rien n'est inféré à propos d'une action a avant que cette action ne soit considérée pour être insérée dans le plan. Cependant, comme nous avons vu en Section 2, on peut faire beaucoup d'inférences sur de telles actions, y compris la restriction de leur temps initial et de leurs supports possibles. Une partie de cette information peut actuellement être inférée avant qu'aucun engagement sur le plan ne soit pris ; les bornes inférieures sur les temps initiaux de *toutes* les actions comme calculées par Graphplan en étant un exemple. Mais ceci n'est pas suffisant ; si des performances similaires et une garantie d'optimalité peuvent être atteintes dans le cadre POCL, les inférences qui utilisent les engagements pris sont alors nécessaires. Afin d'effectuer de telles inférences, la représentation de l'espace des engagements possibles est crucial. Nous devons ainsi faire deux changements par rapport au planificateur POCL temporel ci-dessus. Premièrement, nous introduisons et raisonnons avec des variables qui impliquent *toutes* les actions a du domaine ; pas seulement celles présentes dans le plan courant. Et deuxièmement, pour chacune de ces actions nous introduisons les variables $S(p, a)$ et $T(p, a)$ qui représentent l'action (potentiellement indéterminée) qui supporte la précondition p de a et le temps initial (potentiellement indéterminé) d'une telle action, et nous effectuons sur ces variables certains raisonnements limités mais qui s'avèrent très utiles. Un lien causal $a'[p]a$ devient ainsi une contrainte $S(p, a) = a'$, qui à son tour implique que le support a' d'une précondition p de a commence au temps $T(p, a) = T(a')$.[1]

[1] Des codages 'causaux' propositionnels de la planification Strips ont été formulés et analysés dans [KMS96, MK99]. Notre codage partage certains aspects avec ces formulations mais est plus compact de par l'utilisation d'une représentation temporelle.

Une hypothèse importante que nous faisons est qu'*aucune action dans le domaine ne peut apparaître plus d'une fois dans le plan.* Cette *hypothèse de canonicité* nous permet de regrouper les notions d'action et d'occurrence d'action, conduisant à plusieurs simplifications. On peut considérer qu'il s'agit d'une extension raisonnable d'une hypothèse plus restrictive que l'on trouve souvent dans la recherche en ordonnancement, où chaque action doit apparaître *exactement une fois*. D'un point de vue pratique beaucoup de benchmarks actuels, temporels ou non, admettent des solutions optimales de cette sorte, comme nous le verrons plus loin. Nous exposerons ensuite brièvement comment relâcher cette hypothèse.

La formulation CP de base du planificateur CPT est composée de quatre parties : *pré-traitement, variables, contraintes, et branchement*. Après le pré-traitement, les variables sont créées et les contraintes postées et propagées. Si une inconsistance est trouvée, aucun plan valide n'existe pour le problème. Sinon, la contrainte $T(End) = B$ pour la borne B initialisée au temps initial possible minimal de l'action End est postée et propagée. Les règles de branchement prennent alors le contrôle et si aucune solution n'est trouvée, le processus recommence en rétractant la contrainte $T(End) = B$ et en la remplaçant par $T(End) = B + 1$, et ainsi de suite.

5.1 Le pré-traitement

Dans la phase de pré-traitement, le planificateur calcule les valeurs heuristiques $h^2_T(a)$ et $h^2_T(\{p, q\})$ pour chaque action $a \in O$ et chaque paire d'atomes comme dans [HG01]. Ces valeurs fournissent des bornes inférieures sur le temps pour lequel les préconditions de a seront satisfaites et sur les paires p, q, depuis la situation initiale I. De plus, nous identifions les *mutex structurels* comme les paires d'atomes p, q pour lesquelles $h^2_T(\{p, q\}) = \infty$. Nous disons alors qu'une action a *e-retire* un atome p quand a retire p, ou 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 p est faux après avoir exécuté a ; cf. [NK01]).

En complément à ceci, l'heuristique simple h^1_T est utilisée pour définir des *distances* entre actions [VC99] de la façon suivante. Pour chaque action $a \in O$, nous calculons l'heuristique h^1_T depuis la situation initiale I_a, qui comprend tous les faits *sauf ceux qui sont e-retirés par* a. Nous initialisons ensuite les distances $dist(a, a')$ aux valeurs résultantes $h^1_T(a')$. Ainsi, ces distances encodent des bornes inférieures sur l'écart devant être préservé entre la fin de l'exécution de a et le début de celle de a' dans tous les plans valides dans lesquels a' suit a.

Les distances $dist(a, End)$ et $dist(a, Start)$ sont définies d'une manière légèrement différente. Les premières sont obtenues grâce à un algorithme de plus court chemin sur un 'graphe de pertinence' où les noeuds sont les actions $a \in O$ et l'action End est le noeud initial. Un arc $a \to a'$ dans ce graphe signifie que a' est 'pertinente' pour a (car elle produit une précondition p de a) et son coût est donné par $\delta(a', a) = dur(a') + dist(a', a)$. Les distances $dist(a, End)$ sont alors initialisées au coût du plus court chemin qui connecte End à a dans ce graphe. Les distances $dist(Start, a)$ sont initialisées à $h^2_T(a)$.

5.2 Les variables et les domaines

Un état σ du planificateur est représenté par une collection de variables, de domaines et de contraintes. Comme nous l'avons vu, les variables sont définies pour chaque action $a \in O$ et pas seulement pour les actions du plan courant. De plus, les variables sont créées pour chaque précondition p de chaque action a. Le domaine d'une variable X est dénoté par $D[X]$ ou simplement par $X :: [X_{min}, X_{max}]$ si X est une variable numérique. Les variables, leur domaine initial, et leur signification sont :

- $T(a) :: [0, \infty]$ encode le temps initial de chaque action a, avec $T(Start) = 0$.
- $S(p, a)$ encode le support d'une précondition p de l'action a avec comme 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 le temps initial de $S(p, a)$.
- $InPlan(a) :: [0, 1]$ indique la présence de a dans le plan ; $InPlan(Start) = 1$ (vrai) et $InPlan(End) = 1$.

En plus de ceci, l'ensemble des actions du plan courant est conservé dans la variable $Steps$; i.e., $Steps = \{a \mid InPlan(a) = 1\}$. Les variables $T(a)$, $S(p, a)$, et $T(p, a)$ associées aux actions a qui ne sont pas encore dans le plan sont *conditionnelles* dans le sens suivant : ces variables et leur domaine sont significatifs seulement sous l'hypothèse qu'ils font partie du plan. Afin de s'assurer de cette interprétation, certaines précautions doivent être prises dans la propagation des contraintes, comme nous le verrons ensuite.

5.3 Les contraintes

Les contraintes correspondent essentiellement à des disjonctions, des implications et des contraintes temporelles, et à leurs combinaisons. Les disjonctions sont interprétées constructivement : quand une partie est fausse, l'autre est renforcée. De même pour les implications : quand l'antécédent est vérifié, le conséquent est renforcé. Les conditions selon lesquelles une contrainte est vue comme nécessairement vraie ou fausse dans un état sont déterminées par la nature de la contrainte et le domaine des variables ; en gros, une contrainte est vraie (fausse) si elle est vraie (fausse) pour chaque assignation possible suivant les valeurs des domaines.

Par exemple, $T(a) < T(a')$ est vraie si les domaines des variables sont tels que $T_{max}(a) < T_{min}(a')$ est vraie, est faux si $T_{min}(a) \geq T_{max}(a')$ est vraie, et sinon est *indéterminée*[2]. Les contraintes temporelles sont propagées par consistance de bornes comme montré ci-dessus. Dans les contraintes possédant des termes de la forme $op_{a' \in D[S(p,a)]}$, l'information est propagée *depuis* $S(p, a)$ mais pas *vers* $S(p, a)$; la propagation vers de telles variables est effectuée par des règles explicites avec les variables $S(p, a)$ du côté droit. Les contraintes sont exprimées pour toutes les actions $a \in O$ et toutes les préconditions $p \in pre(a)$; nous utilisons $\delta(a, a')$ pour dénoter $dur(a) + dist(a, a')$.

- **Bornes :** pour tout $a \in O$, $T(Start) + dist(Start, a) \leq T(a)$ et $T(a) + dist(a, End) \leq T(End)$.
- **Préconditions :** le support a' d'une précondition p de a doit précéder a d'une quantité qui dépend de $\delta(a', a)$
$$T(a) \ \geq \min_{a' \in [D(S(p,a))]} (T(a') + \delta(a', a))$$
- **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
$$T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p, a)$$
$$\vee \ T(a) + dur(a) + dist(a, a') \leq T(a')$$

[2] De façon similaire, $T(a) = T(a')$ est vraie si $T_{min}(a) = T_{max}(a) = T_{min}(a') = T_{max}(a')$ est vraie, et est fausse si soit $T(a) < T(a')$ soit $T(a) > T(a')$ est vraie. Les conditions pour les variables énumérées comme $S(p, a)$ sont similaires ; $S(p, a) = a'$ est vraie si $D[S(p, a)] = \{a'\}$ et est fausse si $a' \notin D[S(p, a)]$. Dans tous les cas, la contrainte $\neg C$ est vraie (fausse) si C est fausse (vraie). Nous pouvons aussi noter que $T(a) < T(a')$ est vraie dans notre moteur CP modifié quand $a' = End$, quel que soit le domaine de $T(a)$.

– **Contraintes de mutex :** pour deux actions effet-interférentes a et a'

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

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

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

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

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

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

Comme mentionné plus haut, les variables $T(a)$, $T(p, a)$, et $S(p, a)$ sont *conditionnelles* quand $InPlan(a) = 1$ n'est ni vrai ni faux. Nous les appellerons des variables *in-plan* quand $InPlan(a) = 1$ devient vrai, et *out-plan* quand $InPlan(a) = 1$ devient faux. Les contraintes impliquant des variables in-plan sont propagées de manière classique, un domaine vide provoquant une inconsistance. Les contraintes impliquant une variable out-plan, d'un autre côté, ne sont pas propagées. Finalement, et plus important, les contraintes impliquant des variables conditionnelles associées à la *même action* et par conséquent soumies à la même hypothèse (que a fait partie du plan) sont propagées mais *seulement dans la direction des variables conditionnelles*. Ceci permet de s'assurer que le domaine d'une variable conditionnelle ne dépend que de l'hypothèse selon laquelle cette variable est dans le plan. Une conséquence est que *si le domaine d'une variable conditionnelle associée à une action a devient vide, on peut inférer que l'action a ne peut pas faire partie du plan courant et non que ce plan est inconsistant*. Plus précisément, $InPlan(a)$ est mis à 0 si le domaine de la variable conditionnelle associée à a devient vide ; et dans ce cas, l'action a est enlevée du domaine de toutes les variables de support $S(p, a')$ telles que a ajoute p. D'un autre côté, quand $S(p, a') = a$ est vrai pour une action a' du plan, $InPlan(a)$ est automatiquement mis à 1. Les variables conditionnelles de ce type en programmation par contraintes ont été considérées dans [FM01].

5.4 Branchement

La définition des 'défauts' suit celle considérée plus haut pour la planification POCL temporelle :

– **Menaces de Support :** a' *menace un support* $S(p, a)$ quand les deux actions a et a' sont dans le plan 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 vraies.
– **Préconditions Ouvertes :** $S(p, a)$ est une *précondition ouverte* quand $|D[S(p, a)]| > 1$ est vraie pour une action a dans le plan.
– **Menaces de Mutex :** a et a' constituent une *menace de mutex* quand les deux actions sont dans le plan, 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 vraies (deux actions sont effet-interférentes dans CPT quand l'une retire un effet positif de l'autre, et aucune des deux ne retire de précondition de l'autre).

Quand un défaut est sélectionné dans un état σ, une *division binaire* est créée, dénotée par $[C_1; C_2]$ où C_1 et C_2 sont des contraintes. Le premier fils σ_1 de σ est obtenu en ajoutant C_1 à σ et en fermant le résultat par les règles de propagation ; le second fils σ_2 de σ est généré en ajoutant la contrainte C_2, quand la recherche avec σ_1 échoue. Les divisions binaires générées pour chaque type de menace sont les suivantes :

– Une **Menace de Support** $\langle a', S(p,a)\rangle$ génère la division $[T(a') + dur(a') + \min_{a'' \in D[S(p,a)]} dist(a', a'') \leq T(p,a); T(a) + \delta(a,a') \leq T(a')]$
– Une **Précondition Ouverte** $S(p,a)$ génère la division $[S(p,a) = a'; S(p,a) \neq a']$ pour une action donnée a'
– Une **Menace de Mutex** $\langle a, a'\rangle$ génère la division $[T(a) + \delta(a,a') \leq T(a'); T(a') + \delta(a',a) \leq T(a)]$

Ce schéma de branchement est sain et complet. Le fait qu'il soit sain vient de la validité du plan P obtenu depuis un état consistant σ avec aucun défaut en ordonnançant les actions in-plan a_i à leur temps initial minimal $t_i = T_{min}(a_i)$. La complétude vient de la validité des règles de propagation et des disjonctions $C_1 \vee C_2$ associées à chaque division binaire $[C_1; C_2]$.

Heuristiques de branchement

A chaque étape, le défaut sélectionné est une menace de support s'il en existe une, sinon une précondition ouverte s'il en existe une, sinon une menace de mutex. Les heuristiques pour choisir une menace parmi celles qui existent sont les suivantes :
– **Menace de Support** $\langle a', S(p,a)\rangle$ avec écart minimum $\max[slack(a' \prec S(p,a)), slack(a \prec a')]$ sélectionné d'abord [SC93], où $slack(a \prec a') = T_{max}(a') - (T_{min}(a) + \delta(a,a'))$ et $slack(a' \prec S(p,a)) = T_{max}(p,a) - (T_{min}(a') + \min_{a' \in [D(S(p,a))]} \delta(a',a))$.
– **Précondition Ouverte** $S(p,a)$ sélectionnée la plus tardive en premier ; i.e. maximiser l'expression $\min_{a' \in D[S(p,a)]} T_{min}(a')$, puis effectuer la division par rapport à l'action a' ayant le temps initial minimum (i.e., création de la division $[S(p,a) = a'; S(p,a) \neq a']$.
– **Menaces de Mutex** $\langle a, a'\rangle$ sélectionnées comme elles sont rencontrées.

Les heuristiques pour les menaces de support et les préconditions ouvertes ont une influence significative sur les performances, mais pas les heuristiques sur les menaces de mutex (le plus souvent, il n'en reste plus après avoir résolu les deux premières).

5.5 Ensembles Mutex

Le planificateur proposé contient une amélioration qui est utile dans certains domaines sans ajouter une difficulté supplémentaire importante quand ce n'est pas le cas. Elle est en rapport avec l'idée d'*ensembles mutex :* ensembles M d'actions *dans le plan* (pas seulement des paires) telles que toutes les actions de cet ensemble sont mutex deux à deux. Comme deux de ces actions ne peuvent se recouvrir dans le temps, la fenêtre temporelle associée à l'ensemble M, $\max_{a \in M}(T_{max}(a) + dur(a)) - \min_{a \in M} T_{min}(a)$, doit fournir suffisamment de 'place' pour ordonnancer toutes les actions $a \in M$ en séquence. En prenant en compte les distances, une borne inférieure $\Delta(M)$ pour le temps nécessaire pour ordonnancer toutes les actions de M, l'une après l'autre à part pour l'action finale, est donnée par

$$\Delta(M) = \sum_{a \in M} [dur(a) + \min_{a' \in M | a' \neq a} dist(a,a')] - \max_{\{a,a'\} \subseteq M} dist(a,a')$$

Avec ces bornes inférieures, nous définissons les *Contraintes d'Ensembles Mutex* par

$$\max_{a' \in M} T(a') - \min_{a'' \in M} T(a'') \geq \Delta(M)$$

et les appliquons à *certains* ensembles mutex M identifiés de manière gloutonne en utilisant les actions de $Steps$, décrits ci-après (calculer les ensembles mutex les plus larges semble trop coûteux pour l'instant). L'idée des ensembles mutex est adaptée de concepts similaires utilisés en ordonnancement comme *edge-finding* ; cf. [CP89, BPN01, Lab03].

– Les **Ensembles Mutex Globaux** M_i sont construits de manière gloutonne à chaque fois qu'une action est ajoutée à $Steps$. Initialement un ensemble mutex simple M_0 avec les actions $Start$ et End est défini ; puis chaque fois qu'une action a est ajoutée à $Steps$, a est ajoutée à chaque ensemble mutex existant M_i, $i = 0, \ldots, k$ tel que a est mutex avec chaque action a' de M_i. Un nouvel ensemble mutex M_{k+1} est créé contenant seulement a quand a ne peut être ajoutée à aucun ensemble mutex existant. La contrainte d'ensemble mutex est renforcée pour chacun de ces ensembles M_i.
– Les **Ensembles Mutex de Liens Causaux** M^- et M^+ sont définis pour chaque lien causal $S(p, a)[p]a$ dans le plan. Initialement, ces ensembles sont vides ; puis pour chaque nouvelle action a' qui e-retire p et ne peut pas suivre a (resp. ne peut pas précéder $S(p, a)$), a est ajoutée à M^- (resp. à M^+) si a est mutex avec toutes les actions de M^- (resp. de M^+). Pour ces ensembles mutex M^+ et M^-, la *contrainte d'Ensemble Mutex de Lien Causal* est renforcée, ce qui contrairement aux contraintes d'ensembles mutex globaux, permet non seulement de détecter des inconsistances, mais aussi de renforcer les bornes des variables temporelles $T(p, a)$ et $T(a)$:

$$\min_{a' \in M^-} T(a') + \Delta(M^-) \leq T(p, a) \ \wedge$$

$$T(a) + dur(a) \leq \max_{a' \in M^+} [T(a') + dur(a')] - \Delta(M^+)$$

De plus, pour chaque action a' du plan qui e-retire p, peut suivre $S(p, a)$, et peut précéder a, nous testons la consistance de l'ensemble mutex $M^- \cup \{a'\}$ (resp. $M^+ \cup \{a'\}$) si a' interfère avec chaque action de M^- (resp. M^+). Si l'ensemble est inconsistant (i.e., s'il viole la contrainte de mutex), alors on infère que a' doit suivre a (resp. doit précéder $S(p, a)$).

5.6 Implémentation

Le planificateur CPT est implémenté en utilisant la librairie CP Choco [Lab00] écrite dans le langage de programmation Claire [CJL99], qui effectue une traduction en C++. Dans les premiers temps de l'implémentation, nous avons écrit les contraintes avec Choco d'une façon très fidèle à la formulation décrite ici ; puis nous avons progressivement évolué vers une implémentation basée sur des règles de propagation qui évitent de nombreux tests et déclenchements de propagations. L'implémentation actuelle est une collection de règles qui sont déclenchées par le mécanisme d'événements de Choco. La mise à jour des bornes inférieures, bornes supérieures, et valeurs des domaines sont enregistrées dans des files d'événement, dans lesquelles des événements similaires sont regroupés ; ainsi, si la borne inférieure d'une variable X est augmentée successivement de 1 à 2, puis de 2 à 3 avant que le premier événement ne soit traité, un seul événement est conservé, signifiant que la borne inférieure de X est incrémentée de 1 à 3. Quand un événement est traité, les règles appropriées sont déclenchées, effectuant les propagations correspondantes (mise à jour des bornes des variables concernées, etc.). Les seules contraintes qui ne sont pas réimplémentées en terme de règles de propagation sont les contraintes dynamiques ; c'est-à-dire celles qui sont postées suite aux règles de branchement de menaces de support et de mutex. Nous avons modifié le moteur de recherche de Choco pour que de telles contraintes puissent être backtrackées lors d'inconsistances, et pour prendre en compte la sémantique des variables conditionnelles. Comme décrit précédemment pour ces dernières, un domaine vide ne signifie pas qu'il y a inconsistance mais permet d'exclure l'action hors du plan partiel courant (et de tous ses descendants). Pour les variables temporelles, le comportement conditionnel est obtenu en effectuant nous-mêmes la mise à jour des bornes ; les variables conditionnelles de support, d'un autre côté, sont traitées comme des variables CP normales à l'aide d'une 'fausse' action α ajoutée à leurs domaines, avec $D[S(p,a)] = \{\alpha\}$ signifiant que p ne peut être supportée par aucune action. Les variables $InPlan(a)$ ne sont pas implémentées par des variables CP ; l'information sur le statut d'une action est compilée dans le code des règles de propagation. Le code comprend un certain nombre d'autres optimisations que nous n'avons pas la place de décrire ici, comme l'utilisation d'un ensemble plus important (parfois redondant mais utile) de menaces sur lesquelles effectuer des branchements. Le code sera rendu disponible pour téléchargement.

6 Résultats expérimentaux

Les tests ont été effectués sur un Pentium IV à 1.6Ghz avec 512Mo de RAM sous Linux. La limite de temps pour chaque problème est de une heure. La Table 3 compare CPT , Blackbox (avec Chaff), IPP et TP4 sur des domaines parallèles, et la Table 4 compare CPT et TP4 sur des domaines temporels. Ce sont tous des planificateurs optimaux. Les temps indiqués incluent toutes les opérations, y compris la lecture des fichiers et le pré-traitement. Les domaines parallèles comprennent les instances Blocks et Logistics de la distribution Blackbox. Le reste des instances vient de la 3ème Compétition Internationale de Planification [FL03]. Les tables montrent que les temps d'exécution de CPT sont proches de ceux de Blackbox sur les domaines parallèles, et que CPT peut même résoudre des problèmes que Blackbox ne peut résoudre comme bw-large.c et satellite11 (les bonnes performances de CPT dans le domaine Blocks viennent en partie de l'utilisation des ensembles mutex). CPT est parfois plus lent que IPP mais résout beaucoup plus de problèmes, tandis qu'il domine sans aucun doute TP4 sur les domaines parallèles et temporels, et calcule beaucoup moins de noeuds (nombres indiqués entre parenthèses). Comme discuté dans [HG01], le problème des planificateurs temporels dans les espaces d'états comme TP4 est leur *facteur de branchement* qui augmente de manière exponentielle par rapport au nombre d'actions primitives dans le domaine. Pour CPT le facteur de branchement est égal à 2 ; et après chaque décision de branchement, un mécanisme performant d'élagage est appliqué.

Comme le montre la Table 2, CPT semble aussi dominer LPGP, un planificateur temporel récent qui optimise le nombre d'étapes dans le plan (au sens de Graphplan) plutôt que la durée d'exécution du plan [LF03]. Nous reportons ici les résultats de LPGP effectués sur une machine similaire. Les durées d'exécution de plan plus courtes pour LPGP dans le domaine Satellite viennent d'une légère différence dans la sémantique des plans temporels : LPGP suit la sémantique de PDDL2.1 [FL03] dans laquelle des actions interférentes peuvent se recouvrir dans le temps, par exemple quand les préconditions n'ont pas à être préservées pendant toute la durée d'exécution d'une action.

7 Discussion

Nous avons développé un planificateur POCL temporel, optimal, et indépendant du domaine basé sur la programmation par contraintes qui intègre les heuristiques existantes avec une nouvelle représentation et des règles de propagation qui parviennent à élaguer considérablement l'espace de recherche. Les expérimentations montrent que ce planificateur est plus performant que les planificateurs actuels temporels optimaux et compétitif avec les meilleurs planificateurs

Problèmes temporels	Temps CPU (sec.) LPGP	CPT	Durée exéc. plan LPGP	CPT
zeno4	65.32	4.59	740	522
zeno5	43.83	3.83	583	400
zeno6	57.61	1.78	350	323
driver1	0.33	0.06	91	91
rover1	0.30	0.12	55	53
rover2	0.24	0.07	44	43
rover3	0.44	0.11	58	53
rover4	0.40	0.09	47	45
satellite1	0.17	0.05	41	46
satellite2	24.15	0.95	65	70
satellite3	62.22	0.20	29	34

TAB. 2 – Comparaison CPT vs. LPGP

Problèmes parallèles	Temps CPU (sec.) CPT	BBOX	IPP	TP4	Durée exéc. plan
bw.12step	0.21	0.26	0.03	0.08	12
bw.large.a	0.44	1.13	0.07	0.08	12
bw.large.b	1.75	17.94	2.33	-	18
bw.large.c	231.22	-	-	-	28
rocket.a	0.28	0.38	7.97	44.20	7
rocket.b	0.24	0.45	11.95	31.83	7
log.a	0.70	0.47	781.13	-	11
log.b	0.90	0.91	2099.89	-	13
log.c	1.43	1.46	-	-	13
log.d	29.03	3.73	-	-	14
zeno7	0.84	0.67	0.05	1.76	6
zeno8	5.39	1.59	0.22	166.22	5
zeno9	6.41	2.54	0.68	-	6
zeno10	6.84	4.01	221.32	-	6
zeno11	14.90	5.60	31.06	-	6
zeno12	16.39	11.10	-	-	6
zeno13	45.97	11.42	-	-	7
driver7	0.24	0.24	0.15	22.98	6
driver8	0.30	0.40	3.53	33.59	7
driver9	1.46	1.55	11.26	2979.66	10
driver10	1.02	1.00	17.06	1823.16	7
driver11	4.33	2.67	2.26	1259.06	9
satellite3	0.12	0.26	0.03	0.08	6
satellite4	0.40	1.39	7.28	755.08	10
satellite5	0.99	1.50	145.67	-	7
satellite6	0.56	1.34	90.46	-	8
satellite7	1.55	1.80	1039.23	-	6
satellite8	101.18	235.13	-	-	8
satellite9	8.52	4.68	-	-	6
satellite10	185.90	42.35	-	-	8
satellite11	22.51	-	-	-	8

TAB. 3 – Résultats pour les domaines parallèles

parallèles. La formulation exploite la restriction de canonicité interdisant à une action d'un domaine d'être présente plus d'une fois dans un plan. Cette restriction nous permet de regrouper les notions d'action et d'occurrence d'action, conduisant à plusieurs simplifications. Nous tra-

Problèmes temporels	Temps CPU (sec.) (+nombre d'états) CPT	TP4	Durée exéc. plan
zeno1	0.06 (2)	0.05 (4)	173
zeno2	0.95 (892)	1.23 (17124)	592
zeno3	0.50 (4)	0.05 (618)	280
zeno4	4.59 (2233)	-	522
zeno5	3.83 (124)	34.78 (595988)	400
zeno6	1.78 (54)	6.03 (116715)	323
zeno7	77.58 (45187)	-	665
zeno8	265.93 (78044)	-	522
zeno9	1522.24 (432210)	-	522
zeno10	82.62 (12692)	-	453
zeno11	116.15 (874)	-	423
driver1	0.06 (6)	0.05 (49)	91
driver2	734.98 (724327)	458.19 (17444608)	92
driver3	0.12 (11)	0.05 (621)	40
driver4	91.32 (54350)	-	52
driver5	0.40 (152)	-	51
driver6	111.10 (59702)	-	52
driver7	0.59 (103)	20.79 (323963)	40
driver8	-	-	-
driver9	493.91 (137716)	-	92
driver10	8.75 (1517)	-	38
satellite1	0.05 (5)	0.05 (80)	46
satellite2	0.95 (1435)	8.45 (712294)	70
satellite3	0.20 (26)	0.05 (21143)	34
satellite4	4.36 (5257)	-	58
satellite5	2.32 (1191)	-	36
satellite6	0.82 (47)	-	46
satellite7	2.36 (325)	-	34
satellite8	3324.92 (827408)	-	46
satellite9	8.84 (516)	-	34
satellite10	2160.24 (261474)	-	43

TAB. 4 – Résultats pour les domaines temporels

vaillons actuellement sur une formulation qui supprime cette restriction. Elle est basée sur la distinction entre les *types d'action* et les *instances d'action*. Les plans contiennent seulement des instances d'action qui sont toutes des copies de l'ensemble fixe des types d'action définis grâce à l'ensemble initial des opérateurs du domaine. Les instances d'action sont créées dynamiquement depuis les types d'action comme dans la planification POCL, à chaque fois qu'une nouvelle instance supporte une précondition ouverte. Dans ce cadre, cependant, la création de nouvelles instances prend la forme d'une opération de 'clonage' : pour la nouvelle instance a' d'un type a, les variables $T(a')$, $S(p, a')$, et $T(p, a')$ sont créées comme de nouvelles copies des variables $T(a)$, $S(p, a)$, et $T(p, a)$ avec leurs domaines respectifs, où p est une précondition de a. De plus, la nouvelle instance a' est ajoutée comme une nouvelle action indépendante à tous les domaines des variables de support qui contiennent le type d'action a. Le schéma résultant peut actuellement être vu comme une implémentation paresseuse d'un domaine de planification avec un nombre infini d'instances d'action, les types d'action résumant le domaine des instances d'action qui n'ont pas encore été utilisées dans le plan. Pour les domaines parallèles, la dégradation de performances semble modérée (les temps d'exécution ne semblent pas augmenter de plus de 15% pour les instances considérées dans cet article) ; tandis que pour les domaines temporels, elle semble plus importante (bien que le planificateur qui en résulte soit toujours plus performant que les planificateurs optimaux temporels actuels). Nous travaillons actuellement à améliorer ce planificateur non-canonique qui participe à la 4ème Compétition Internationale de Planification.

Références

[BF95] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995.

[BLG97] 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.

[BPN01] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems*. Kluwer, 2001.

[CJL99] Y. Caseau, F. X. Josset, and F. Laburthe. Claire : Combining sets, search and rules to better express algorithms. In *Proceedings of the Int. Conf. on Logic Programming*, 1999.

[CP89] J. Carlier and E. Pinson. An algorithm for solving the job shop scheduling problem. *Management Science*, 35(2), 1989.

[DK00] Minh Binh Do and Subbarao Kambhampati. Solving planning-graph by compiling it into CSP. In *Proc. AIPS-00*, pages 82–91, 2000.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49 :61–95, 1991.

[FL03] M. Fox and D. Long. PDDL2.1 : An extension to PDDL for expressing temporal planning domains. *Journal of AI Research*, 2003.

[FM01] F. Focacci and M. Milano. Connections and integrations of dynamic programming and constraint programming. In *Proc. of the Int. Workshop on Integration of AI and OR techniques (CP-AI-OR'01)*, 2001.

[Gef01] H. Geffner. Planning as branch and bound and its relation to constraint-based approaches. Technical report, Universidad Simón Bolívar, 2001. At `www.tecn.upf.es/~hgeffner`.

[HG00] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, pages 70–82, 2000.

[HG01] P. Haslum and H. Geffner. Heuristic planning with time and resources. In *Proc. European Conference of Planning (ECP-01)*, pages 121–132, 2001.

[JMMR00] A. Jonsson, P. Morris, N. Muscettola, and K. Rajan. Planning in interplanetary space : Theory and practice. In *Proc. AIPS-2000*, pages 177–186, 2000.

[JP96] D. Joslin and M. E. Pollack. Is "early commitment" in plan generation ever a good idea ? In *Proceedings AAAI-96*, pages 1188–1193, 1996.

[KKY95] 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.

[KMS96] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings Int. Conf. on Principles of KR and Reasoning (KR'96)*, pages 374–384, 1996.

[KNHD97] J. Koehler, B. Nebel, J. Hoffman, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. Proc. 4th European Conf. on Planning (ECP-97). Lect. Notes in AI 1348*, pages 273–285. Springer, 1997.

[KS99] H. Kautz and B. Selman. Unifying SAT-based and Graph-based planning. In T. Dean, editor, *Proceedings IJCAI-99*, pages 318–327. Morgan Kaufmann, 1999.

[Lab00] F. Laburthe. Choco : implementing a cp kernel. In *Proceedings CP-00, Lecture Notes in CS, Vol 1894*. Springer, 2000.

[Lab03] P. Laborie. Algorithms for propagating resource constraints in ai planning and scheduling. *Artificial Intelligence*, 143 :151–188, 2003.

[LF03] D. Long and M. Fox. Exploiting a graphplan framework in temporal planning. In *Proceedings ICAPS 2003*, pages 52–61, 2003.

[LG95] P. Laborie and M. Ghallab. Planning with sharable resources constraints. In C. Mellish, editor, *Proc. IJCAI-95*, pages 1643–1649. Morgan Kaufmann, 1995.

[Lho93] O. Lhomme. Consistency techniques for numeric CSPs. In *Proceedings IJCAI-93*, pages 232–238. Morgan Kaufmann, 1993.

[McD96] D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*, 1996.

[MK99] A. Mali and S. Kambhampati. On the utility of plan-space (causal) encodings. In *Proc. AAAI-99*, pages 557–563, 1999.

[NK01] Xuan Long Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *Proc. IJCAI-01*, 2001.

[PG02] H. Palacios and H. Geffner. Planning as branch and bound : A constraint programming implementation. In *Proc. XXVIII Conf. Latinoamericana de Informática*, pages 239–251, 2002.

[SC93] S. Smith and C. Cheng. Slack-based heuristics for the constraint satisfaction scheduling. In *Proc. AAAI-93*, pages 139–144, 1993.

[SFJ00] D. Smith, J. Frank, and A. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[SW99] D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI-99*, pages 326–337, 1999.

[VC99] P. Van Beek and X. Chen. CPlan : a constraint programming approach to planning. In *Proc. National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590. AAAI Press/MIT Press, 1999.

[Wel94] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4) :27–61, 1994.

[YS03] B. L. S. Younes and R. G. Simmons. VHPOP : Versatile heuristic partial order planner. *Journal of AI Research*, 20 :405–430, 2003.