Un algorithme pour la résolution optimale de problèmes de planification valués

Martin Cooper, Marie de Roquemaurel, Pierre Régnier

IRIT - Université Paul Sabatier
118, route de Narbonne
31062 Toulouse, cedex 9, France
{cooper,deroquemaurel,regnier}@irit.fr

Résumé : Nous montrons dans cet article comment utiliser le cadre des CSP pondérés (WCSP) pour trouver un plan-solution de coût optimal à des problèmes de planification avec actions valuées. La méthode proposée utilise un graphe de planification, structure disjonctive qui contient, pour un problème de planification donné, tous les plans-solutions potentiels d'une longueur k fixée. Ce graphe est codé en WCSP, puis nous utilisons l'algorithme de cohérence d'arc directionnelle FDAC à chaque noeud de la recherche pour trouver efficacement un plan-solution de coût optimal à un niveau k donné. Les bons résultats obtenus avec cette méthode nous permettent d'étendre ces travaux à la recherche d'un plan-solution de coût optimal. Notre algorithme poursuit la recherche dans les niveaux suivants du graphe après l'apparition d'une première solution et garantit l'obtention d'un plan-solution parallèle de nombre de niveaux minimum parmi ceux qui minimisent la métrique. Les résultats obtenus sont encourageants et montrent que l'utilisation des WCSP pour la résolution de problèmes de planification valués optimale est envisageable.

1 Introduction

Un des challenges actuels de la planification est la résolution de problèmes numériques pour lesquels on cherche à optimiser le coût d'un plan-solution. Parmi les planificateurs qui traitent ces problèmes numériques, certains font des choix heuristiques pour tenter d'optimiser le coût du plan produit (Do & Kambhampati, 2003; Gerevini *et al.*, 2004; Refanidis & Vlahavas, 2003; Haslum & Geffner, 2001; Chen *et al.*, 2004) alors que d'autres se contentent de calculer a posteriori le coût du plan-solution (Hoffmann, 2003).

En établissant des liens entre l'IA et la Recherche Opérationnelle, les CSP pondérés (WCSP) permettent de modéliser des contraintes strictes et des critères d'optimisation exprimés sous la forme d'une agrégation de fonctions de coût (Schiex *et al.*, 1995). Plusieurs codages de problèmes de planification en problèmes de satisfaction de contraintes ont déjà été proposés : CPLAN (van Beek & Chen, 1999), GP-CSP (Do & Kambhampati, 2001), puis CSPPLAN (Lopez & Bacchus, 2003). Aucune de ces approches n'a

été étendue aux problèmes de planification avec actions valuées.

L'article est organisé de la manière suivante : la section 2 définit la planification avec actions valuées, GRAPHPLAN, et la notion de métrique d'un plan. La section 3 introduit un exemple qui nous servira à illustrer le fonctionnement de notre méthode. Dans la section 4 nous montrons ensuite comment coder un graphe de planification en WCSP. La section 5 présente l'algorithme de cohérence d'arc directionnelle FDAC qui nous permet d'extraire efficacement une solution optimale de ces WCSP. Nous montrons enfin, dans la section 6, comment l'algorithme GP-WCSP* utilise ces résultats pour obtenir un plan-solution parallèle de coût optimal.

2 Définitions préliminaires

Définition 2.1 (problème de planification avec actions valuées)
Un problème de planification avec actions valuées est un triplet $\Pi = \langle A, I, B\rangle$ tel que :

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

Définition 2.2 (application d'une action)
L'application d'une action a à un état E (notée $E \uparrow a$) est possible ssi tous les fluents de prec(a) sont satisfaits dans E ; elle produit un état $E' = (E - del(a)) \cup add(a)$.

Définition 2.3 (graphe de planification)
Un graphe de planification (Blum & Furst, 1997) est un graphe orienté, construit en temps et en espace polynomial, composé de plusieurs niveaux qui comportent des noeuds d'actions et de fluents, et des arcs de préconditions, d'ajouts et de retraits. Le niveau 0 contient les fluents de l'état initial. Chaque niveau $i > 0$ est composé de toutes les actions applicables à partir des fluents du niveau $i-1$, et des fluents produits par les actions du niveau i. Les arcs représentent les relations entre les actions et les fluents : les actions du niveau i sont reliées aux fluents préconditions du niveau $i-1$ par des arcs préconditions, et à leurs ajouts et retraits du niveau i par des arcs d'ajouts et de retraits.

Le maintien des fluents du niveau $i-1$ au niveau i est assuré par des actions appelées $no-op$ qui ont un unique fluent comme précondition et ajout. Pour chaque niveau du graphe, on calcule des exclusions binaires (mutex) entre paires d'actions et paires de fluents d'un même niveau (cf. définition 2.4). Le graphe est étendu niveau par niveau jusqu'à ce que les fluents du but soient présents dans le niveau courant et qu'il n'existe pas de mutex entre eux.

Définition 2.4 (exclusion mutuelle)
Deux actions a_1 et a_2 sont mutex à un niveau i du graphe (noté $mutex(a_1, a_2, i)$) ssi :

1. *une action retire un fluent requis ou ajouté par l'autre action :*
   $\Big((del(a_1) \cap (prec(a_2) \cup add(a_2))) \cup (del(a_2) \cap (prec(a_1) \cup add(a_1)))\Big) \neq \emptyset$ *ou*

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

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

Définition 2.5 (action et ensemble indépendant)
Deux actions a et b sont des actions indépendantes (notée $a\#b$) ssi elles ne vérifient pas (1) dans la définition 2.4. Un ensemble d'actions $Q_i = \{a_1, \ldots, a_n\}$ est un ensemble indépendant ssi tous les couples d'actions différentes qui le composent sont indépendantes deux à deux : $\forall\ a, b \in Q, a \neq b/a\#b$. *L'application d'un ensemble indépendant d'actions $Q = \{a_1, \ldots, a_n\}$ sur un état E (notée $E \uparrow Q$) est possible ssi* $\forall\ a \in Q$, $prec(a)$ *est satisfait dans E. Il en résulte l'état E' tel que* $E' = \left(E - \bigcup_{a \in Q} del(a)\right) \cup \bigcup_{a \in Q} add(a)$. *Lorsqu'un ensemble d'actions Q est indépendant, l'application des actions de Q conduit à un état identique quel que soit leur ordre d'exécution.*

Définition 2.6 (plan parallèle)
Un plan parallèle de n niveaux est une séquence d'ensembles d'actions indépendantes Q_i, noté $P =< Q_1, \ldots, Q_n >$. L'application d'un plan parallèle P de n niveaux à un état E_0 (noté $E_0 \uparrow P$) est possible ssi pour tout $i = 1, \ldots, n$, Q_i est applicable dans E_{i-1} et produit l'état E_i. Dans ce cas, on dit que E_n est atteignable à partir de E et P est appelé plan parallèle correct. Un plan parallèle correct P est un plan-solution parallèle (ou plan parallèle valide) ssi E_0 est l'état initial I et que le but B est satisfait dans l'état final E_n.
Remarque : GRAPHPLAN construit des plans-solutions parallèles optimaux en nombre de niveaux. Dans la suite de cet article, nous nous intéresserons uniquement à ce type de plan.

Définition 2.7 (stabilisation du graphe)
La stabilisation du graphe est atteinte lorsque deux niveaux successifs contiennent les mêmes actions et fluents, et les mêmes mutex d'actions et de fluents.

Définition 2.8 (graphe réduit)
Un graphe réduit est un graphe de planification dans lequel en partant des buts présents au niveau k, on supprime toutes les actions du même niveau (et leurs arcs de précondition) qui ne permettent pas l'obtention de ces fluents ; on recommence ensuite au niveau $k-1$ en supprimant toutes les actions qui ne permettent pas l'obtention de préconditions des actions du niveau k qui ont été conservées et ainsi de suite jusqu'au niveau 1.

Définition 2.9 (métrique d'un plan)
La qualité d'un plan P est estimée par une fonction appelée métrique du plan. Nous considèrerons ici les métriques additives telles que $metrique(P) = \sum_{a \in P} cout(a)$. *Nous noterons $\mathcal{P}_k$ l'ensemble des plans-solutions de longueur inférieure ou égale à k niveaux. $\mathcal{P}_k^*$ est l'ensemble des plans-solutions de $\mathcal{P}_k$ qui minimisent la métrique :* $\forall\ P \in \mathcal{P}_k^* : metrique(P) = \min_{P' \in \mathcal{P}_k} metrique(P')$. *$P_k^*$ est un plan appartenant à $\mathcal{P}_k^*$. C_k^* sera le coût d'un plan-solution P_k^* de coût optimal parmi les plans de longueur inférieure ou égale à k niveaux.*

Définition 2.10 (plan optimal / coût optimal)
P^* est un plan-solution de nombre de niveaux minimum parmi ceux qui minimisent la métrique. C^* est le coût d'un tel plan.

Le problème que nous cherchons à résoudre consiste, pour un problème de planification avec actions valuées $\Pi = \langle A, I, B \rangle$, à trouver un plan-solution P^* (nous parlerons ici d'un problème de planification optimal).

3 Exemple

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

- $\forall\, j \in \{A, B, C, D, E\}, trajet_{ij} : < \{v_i\}, \{v_j, \neg v_i\}, cout_{ij} >$. La figure 1 donne les trajets possibles et leurs coûts.
- $prendre_i : < \{v_i, c_i\}, \{c_v, \neg c_i\}, 5 >$.
- $poser_i : < \{v_i, c_v\}, \{c_i, \neg c_v\}, 3 >$.

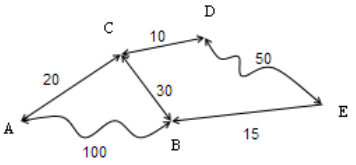

FIG. 1 – exemple

Dans un premier temps, la construction du graphe de planification se poursuit jusqu'au niveau 3, premier niveau dans lequel tous les buts apparaissent sans mutex. Le premier plan-solution trouvé $P_3^* = \langle \{prendre_A\}, \{trajet_{AB}, noop_{c_v}\}, \{poser_B\} \rangle$ correspond au seul plan séquentiel $\langle prendre_A, trajet_{AB}, poser_B \rangle$ qui a un coût $C_3^* = 108$. Ce plan-solution n'est cependant pas de coût optimal, puisqu'il existe un plan optimal en 4 niveaux $\langle prendre_A, trajet_{AC}, trajet_{CB}, poser_B \rangle$ de coût 58. Nous allons d'abord montrer comment obtenir un plan-solution de coût optimal de longueur donné puis, dans un deuxième temps, nous montrerons comment obtenir un plan optimal.

4 Codage du graphe en WCSP

Comme démontré dans (Do & Kambhampati, 2001), la phase d'extraction de GRAPHPLAN peut être vue comme un DCSP (Dynamic Constraint Satisfaction Problem) (Mittal & Falkenhainer, 1990). Initialement, seul un sous-ensemble de variables est actif, et l'objectif consiste à trouver une affectation pour toutes les variables actives telle

qu'elle soit consistante avec l'ensemble des contraintes. Pendant l'extraction, chaque fluent f_i à un niveau i du graphe de planification peut être assimilé à une variable d'un CSP ; l'ensemble des actions qui ajoutent f_i constituent son domaine et les mutex produits durant l'étape de construction peuvent être assimilés à des contraintes du CSP. À partir des fluents du but, GRAPHPLAN essaye d'extraire un plan à partir du graphe de planification en affectant une valeur (une action) à chaque variable (fluent) satisfaisant l'ensemble de contraintes (mutex). L'affectation des valeurs aux variables est un processus dynamique car chaque affectation d'une variable à un niveau i active d'autres variables du niveau précédent (les préconditions de l'action choisie).

Le codage, par un WCSP, d'un plan issu d'un graphe de planification avec actions valuées nécessite plusieurs modifications de la méthode initialement proposée par (Do & Kambhampati, 2001) pour le codage d'un graphe de planification classique en CSP. Une fois le graphe de planification construit et réduit, son codage en WCSP nécessite sept étapes :

1. *Réécriture du graphe :* en partant du dernier niveau du graphe réduit, on renomme chaque fluent en f_i et on numérote chaque action en j, i et j étant des entiers strictement positifs. L'ordre dans lequel cette numérotation est réalisée est important car il détermine en partie l'ordre dans lequel les variables et les valeurs pourront être ordonnées pour la résolution du WCSP associé.
2. *Création des variables et des domaines :* pour chaque fluent n'appartenant pas à l'état initial, nous créons une variable dont le domaine est l'ensemble des actions qui produisent ce fluent. Pour les fluents n'appartenant pas au but, nous ajoutons la valeur -1 pour représenter sa non-activation éventuelle.
3. *Traduction des mutex entre fluents :* pour tous les fluents mutex f_i et f_j, l'exclusion mutuelle est traduite par une contrainte qui exprime le fait que f_i et f_j ne doivent pas être activés en même temps : $(f_i = -1) \vee (f_j = -1))$
4. *Traduction des mutex entre actions :* pour toutes les actions mutex a et b d'effets respectifs f_i et f_j $(f_i \neq f_j)$, l'exclusion mutuelle est traduite par une contrainte qui exprime le fait que f_i et f_j ne peuvent pas être activés en même temps par les actions a et b : $\neg((f_i = a) \wedge (f_j = b))$.
5. *Traduction des arcs d'activité :* l'activation d'un fluent f_i produit par une action a_i entraîne l'activation des préconditions de cette action. Ceci se traduit par la contrainte d'activité : $\forall f_j \in prec(a), (f_i = a) \Rightarrow (f_j \neq -1)$.
6. *Traduction du coût des actions :* Pour chaque valeur a, nous ajoutons une contrainte unaire pour chaque affectation $f = a$ correspondant au coût de l'action a. Les No-ops ont un coût nul.
7. *Prise en compte des ajouts multiples :* Lorsqu'une action a produit plusieurs fluents $f_i \in effet(a)$, le coût de cette action est compté plusieurs fois si plusieurs d'entre eux sont utiles à la résolution du WCSP. Pour résoudre ce problème, nous créons un fluent intermédiaire f^{int} de domaine $\{a, -1\}$. Ainsi l'action a est remplacée par une action fictive a^{int} (de coût nul) dans les domaines des fluents f_i. Nous ajoutons également une contrainte d'activité $(f_i = a^{int}) \Rightarrow (f^{int} = a)$ entre le fluent f^{int} et chaque fluent $f_i \in effet(a)$.

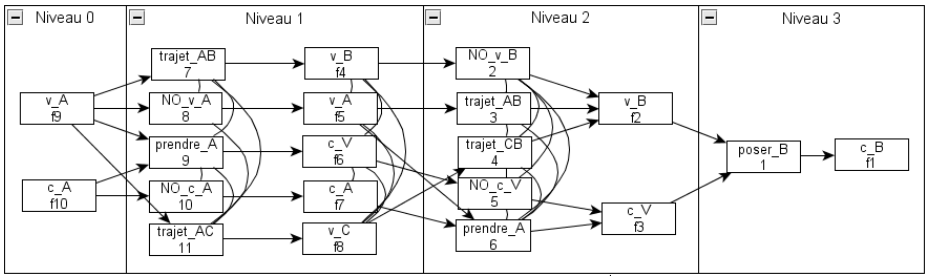

FIG. 2 – graphe de planification réduit et renommé de 3 niveaux.

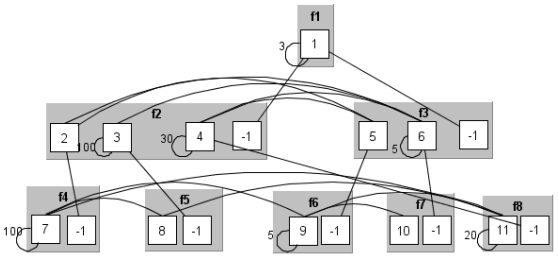

FIG. 3 – WCSP correspondant au graphe de la figure 2.

La figure 2 donne le graphe réduit de niveau 3 de l'exemple avec le renommage correspondant. Le codage du graphe en WCSP est présenté dans la figure 3 où toutes les contraintes binaires sont des mutex et ont donc des coûts infinis.

5 Recherche d'une solution optimale dans le WCSP

Dans cette section, nous utilisons c_{ij} pour représenter une fonction de coût binaire sur les variables X_i, X_j dans un WCSP, c_i représente une fonction unaire de coût sur X_i et $c_\emptyset$ représente une fonction de coût sans argument (indépendante des valeurs affectées aux variables). Une solution à un WCSP binaire est un n-tuple x qui minimise $\sum_i c_i(x_i) + \sum_{ij} c_{ij}(x_i, x_j) + c_\emptyset$.

La réduction de l'espace de recherche par propagation des inconsistances peut être généralisée des CSP aux WCSP, directement par propagation de coûts infinis. Les coûts finis peuvent aussi être propagés mais de telles propagations doivent être compensées : si l'on soustrait α de $c_{ij}(a, u)$ (pour chaque valeur u dans le domaine de la variable X_j), on doit simultanément ajouter α à $c_i(a)$ afin de conserver un WCSP équivalent. Quand $\alpha > 0$, l'opération est une projection ; si $\alpha < 0$ alors l'opération est une extension. Une troisième opération possible est la projection unaire, qui soustrait simultanément α de $c_i(u)$ (pour chaque valeur u dans le domaine de X_i) tout en ajoutant α à $c_\emptyset$. Dans ces trois cas, l'opération peut être appliquée seulement si tous les coûts résultants sont non négatifs. la propagation de coûts finis incrémente la borne inférieure $c_\emptyset$ et réduit l'espace de recherche exploré par le Branch-and-Bound.

En appliquant toutes les projections unaires possibles, nous établissons la cohérence de noeuds (Larrosa, 2002). En propageant tous les coûts infinis (entre les contraintes binaires et unaires) et en appliquant les opérations de projection jusqu'à la convergence,

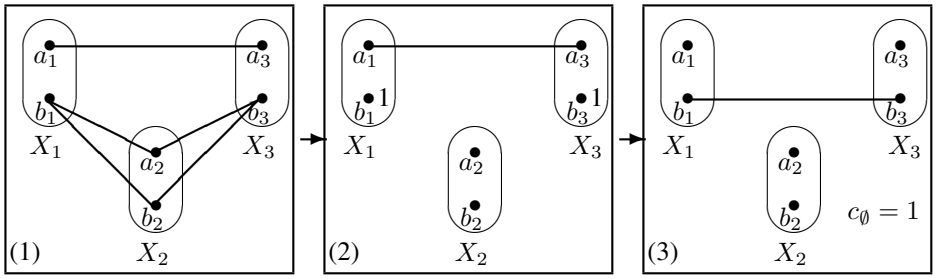

FIG. 4 – Exemple de simplification d'un WCSP par la méthode de propagation de contraintes FDAC.

nous établissons la cohérence d'arc (SAC) (Cooper & Schiex, 2004). Afin d'avoir des coûts non nuls disponibles le plus tôt possible pendant la recherche, la cohérence d'arc directionnelle (DAC) choisit toujours d'envoyer les coûts (par l'intermédiaire d'opérations de projection et d'extension) vers les variables qui apparaissent le plus tôt lors de l'instanciation. Ceci a également tendance à regrouper les coûts sur les mêmes variables, et permet ainsi d'arriver à une plus grande valeur de $c_\emptyset$ après l'établissement de la cohérence de noeud. L'algorithme de cohérence d'arc directionnelle totale FDAC (Cooper, 2003) est la combinaison de la cohérence d'arc directionnelle et de la cohérence d'arc.

Voici un exemple d'application de FDAC. Le WCSP initial décrit par la figure 4 (1) est composé de trois variables X_i de domaines $\{a_i, b_i\}$ ($i = 1, 2, 3$). Un arc entre deux valeurs du WCSP représente un coût de 1. Par exemple, l'arc (a_1, a_3) signifie que $c_{13}(a_1, a_3) = 1$. FDAC commence par projeter un coût de 1 à partir de $c_{12}(b_1, a_2)$, $c_{12}(b_1, b_2)$ vers $c_1(b_1)$ (qui établit la cohérence d'arc directionnelle DAC) et puis à partir de $c_{23}(a_2, b_3)$, $c_{23}(b_2, b_3)$ vers $c_3(b_3)$ (une opération de cohérence d'arc (SAC)). Le WCSP résultant, montré dans la figure 4 (2), est presque arc consistant directionnel : étendre le coût de 1 à partir de $c_3(b_3)$ jusqu'à c_{13} nous permet de projeter un coût de 1 de c_{13} vers $c_1(a_1)$. La figure 4 (3) montre le WCSP obtenu après application de la projection unaire de c_1. Ce WCSP est totalement arc consistant directionnel par rapport à l'ordre des variables X_1, X_2, X_3, et le coût minorant de la solution $c_\emptyset$ est de 1.

Soit m le coût de la meilleure solution trouvé durant l'algorithme Branch-and-Bound. En posant que le but est de trouver une première solution optimale, nous pouvons couper les branches de l'arbre de recherche dès lors que $c_\emptyset \geq m$. De plus, nous pouvons attribuer ∞ à un coût unaire $c_i(a) \geq m - c_\emptyset$ et à un coût binaire $c_{ij}(a, b) \geq m - (c_i(a) + c_j(b) + c_\emptyset)$. Appliquer ces dernières opérations fournit une forme plus forte de cohérence d'arc, mais augmente la complexité en temps du calcul du pire cas de FDAC en passant de $O(ed^2)$ (Cooper, 2003) à $O(end^3)$ (Larrosa & Schiex, 2003), où e est le nombre de contraintes binaires, n le nombre de variables et d la taille maximum de domaine. FDAC a été récemment étendu à la cohérence d'arc directionnelle existentielle (EDAC) (de Givry *et al.*, 2005) qui exécute l'opération suivante : si pour *chaque* valeur a dans le domaine d'une variable X_i, il existe un voisin X_j tel qu'il est possible d'augmenter le coût de la contrainte unaire portant sur a en projetant le coût de c_j et de la contrainte c_{ij}, alors effectuer toutes ces opérations et appliquer une opération unaire de projection sur X_i, pour augmenter $c_\emptyset$. Comme tous les algorithmes de propagation de contraintes présentés précédemment, FDAC n'a pas de fermeture unique ; le résultat

dépend de l'ordre dans lequel les différentes extensions et projections sont appliquées mais surtout de l'ordre des variables.

Les expérimentations que nous avons réalisées (Cooper *et al.*, 2006) nous ont permis de comparer l'efficacité des algorithmes de cohérence de noeud, EDAC et FDAC pour la résolution des WCSP issus des graphes de planification. Nos résultats montrent clairement que c'est l'utilisation de FDAC à chaque noeud qui permet de résoudre le plus efficacement ces WCSP en termes de temps de calcul. Ils montrent que la taille des WCSP qui peuvent être résolus en pratique est beaucoup plus importante que la taille du plus grand WCSP aléatoire résolu dans la littérature (de Givry *et al.*, 2005). Par exemple, l'espace de recherche d'une instance est de l'ordre de $3 * 10^{221}$ tandis que celui du plus gros problème aléatoire résolu dans (de Givry *et al.*, 2005) est de l'ordre de 10^{36}. Ceci peut être dû à la structure particulière du WCSP produit par le codage du graphe de planification : on retrouve la structure en niveau du graphe de planification avec une forte connectivité interne et entre niveaux adjacents).

Le wcsp de l'exemple est résolu avec FDAC appliqué à chaque noeud de l'arbre de recherche branch-and-bound. Le plan-solution optimal de niveau 3 correspondant est $P_3^* = \langle \{prendre_A\}, \{trajet_{AB}, noop_{c_v}\}, \{poser_B\} \rangle = \langle prendre_A, trajet_{AB}, poser_B \rangle$ de coût 108. Sachant efficacement trouver un plan-solution P_k^* de coût optimal à un niveau donné k, nous pouvons chercher un plan-solution optimal P^*.

6 Algorithme GP-WCSP*

L'algorithme anytime GP-WCSP* que nous avons développé est basé sur celui de GRAPHPLAN. Une fois la phase d'expansion du graphe de planification terminée, le graphe est codé en un WCSP équivalent (cf. partie 4) puis résolu (cf. partie 5). Le coût du plan-solution optimal trouvé à un niveau donné du graphe nous sert ensuite de majorant pour relancer la recherche d'une meilleure solution au niveau suivant.

Notre approche donne un planificateur complet dans le sens où l'algorithme s'arrête en retournant un plan-solution optimal P^* s'il existe. Par contre, comme pour tous les planificateurs de ce type (SATPLAN'06 (Kautz & Selman, 1999), (Kautz *et al.*, 2006), GP-CSP (Do & Kambhampati, 2001), MaxPlan (Chen *et al.*, 2007)...) le problème de l'arrêt lorsqu'il n'y a pas de solution n'est pas résolu.

L'algorithme garantit l'obtention d'un plan-solution P^* de coût optimal lorsqu'il atteint une borne supérieure au nombre de niveaux à développer (borne appelée $NivMax$). Une première estimation de cette borne peut-être facilement calculée à partir du premier plan-solution P_k^* obtenu au niveau k : dans le pire des cas, un plan optimal P^* aurait un coût $C^* = C_k^* - \epsilon$ (où $\epsilon > 0$) et serait un plan séquentiel composé de $(C_k^* - \epsilon)/C_{min}$ actions dont le coût ne pourrait être inférieur à $C_{min} = \min_{a \in G} cout(a)$, par conséquent $NivMax = \lfloor (C_k^* - \epsilon)/C_{min} \rfloor = \lceil C_k^*/C_{min} \rceil - 1$. Avec cette méthode de calcul, la valeur de $NivMax$ calculée au niveau 3 est 35, on construit donc le niveau suivant du graphe et on en extrait $P_4^* = \langle prendre_A, trajet_{AC}, trajet_{CB}, poser_B \rangle$ de coût $C_4^* = 58$. La valeur de $NivMax$ peut alors être diminuée ($NivMax \leftarrow 20$) et l'algorithme poursuit la recherche jusqu'au 20ème niveau afin de garantir que P_4^* est un plan optimal. En pratique les valeurs obtenues pour $NivMax$ ne permettent pas de résoudre des problèmes de planification optimaux de taille significative. Nos recherches actuelles (Cooper *et al.*, 2007) portent donc sur les moyens de diminuer efficacement la valeur

de $NivMax$. Pour l'exemple, nous montrons ainsi que le développement du graphe au niveau 4 est suffisant pour garantir l'obtention d'un plan optimal.

Algorithme 1 Résolution optimale d'un problème de planification valué $\Pi\langle A, I, B\rangle$

Fonction :
- $constructionGraphe$: construit le graphe jusqu'à l'obtention de tous les buts sans mutex entre eux dans le niveau courant. Si le graphe ne contient pas tous les buts sans mutex au niveau de stabilisation alors il retourne un échec.
- $constructionGrapheNiveauSuivant(G_k)$: retourne le graphe de $k+1$ niveaux.
- $codageWCSP(G)$: retourne le wcsp correspondant au codage du graphe G réduit.
- $resolutionWCSP(wcsp, C)$: retourne une solution optimale au wcsp et son coût, strictement inférieur à C. Sinon retourne echec.
- $NivMax(C) = \lceil C/(min_{a\in A} cout(a)) \rceil - 1$.

Algorithme GP-WCSP* :

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

7 Conclusions et perspectives

Nous avons introduit une nouvelle méthode pour extraire un plan-solution de coût optimal à partir d'un graphe de planification avec actions valuées en utilisant des techniques de CSP pondérés. On obtient ainsi un plan-solution parallèle, optimal en nombre de niveaux, et de coût minimum parmi les plans d'une longueur donnée. Les très bons résultats obtenus nous permettent d'utiliser cette méthode dans l'algorithme GP-WCSP* pour obtenir une solution de coût optimal. Le nombre de niveaux à construire pour garantir l'obtention de cette solution optimale est fini mais inconnu au départ. Une borne supérieure à ce nombre peut être calculée à partir du premier plan-solution obtenu. Pour pouvoir résoudre pratiquement des problèmes de planification valués de taille significative, nous cherchons désormais à diminuer la valeur de cette borne en étudiant les propriétés des problèmes.

Références

BLUM A. & FURST M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence (AI)*, **90**(1-2), 281–300.

CHEN Y., HSU C. & WAH B. (2004). Sgplan : Subgoal partitioning and resolution in planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, p. 30–33.

CHEN Y., ZHAO X. & ZHANG W. (2007). Long distance mutual exclusion for propositional planning. In *IJCAI*, p. 1840–1845.

COOPER M., CUSSAT-BLANC S., DE ROQUEMAUREL M. & RÉGNIER P. (2006). Soft arc consistency applied to optimal planning. In *International Conference on Principles and Practice of Constraint Programming, LNCS 4204*, p. 680–684.

COOPER M., DE ROQUEMAUREL M. & RÉGNIER P. (2007). Recherche d'une solution optimale dans des graphes de planification avec actions valuées. In *Journées Francophones Planification, Décision, Apprentissage pour la conduite de systèmes*.

COOPER M. C. (2003). Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, **134**(3), 311–342.

COOPER M. C. & SCHIEX T. (2004). Arc consistency for soft constraints. *Artificial Intelligence*, **154**(1-2), 199–227.

DE GIVRY S., HERAS F., ZYTNICKI M. & LARROSA J. (2005). Existential arc consistency : Getting closer to full arc consistency in weighted csps. In *International Joint Conference on Artificial Intelligence (IJCAI)*, p. 84–89.

DO M. B. & KAMBHAMPATI S. (2001). Planning as constraint satisfaction : Solving the planning graph by compiling it into csp. *Artificial Intelligence*, **132**(2), 151–182.

DO M. B. & KAMBHAMPATI S. (2003). Sapa : A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research (JAIR)*, **20**, 155–194.

GEREVINI A., SAETTI A. & SERINA I. (2004). Planning with numerical expressions in lpg. In *European Conference on Artificial Intelligence (ECAI)*, p. 667–671.

HASLUM P. & GEFFNER H. (2001). Heuristic planning with time and resources. In *Proceedings of the Sixth European Conference on Planning*, p. 121–132.

HOFFMANN J. (2003). The metric-ff planning system : Translating "ignoring delete lists" to numeric state variables. *JAIR*, **20**, 291–341.

KAUTZ H. & SELMAN B. (1999). Unifying SAT-based and graph-based planning. In *IJCAI*, p. 318–325.

KAUTZ H., SELMAN B. & HOFFMANN J. (2006). Satplan : Planning as satisfiability. In *Abstracts of the 5th International Planning Competition*.

LARROSA J. (2002). Node and arc consistency in weighted csp. In *AAAI*, p. 48–53.

LARROSA J. & SCHIEX T. (2003). In the quest of the best form of local consistency for weighted csp. In *IJCAI*, p. 239–244.

LOPEZ A. & BACCHUS F. (2003). Generalizing graphplan by formulating planning as a csp. In *IJCAI*, p. 954–960.

MITTAL S. & FALKENHAINER B. (1990). Dynamic constraint satisfaction problems. In *National Conference on Artificial Intelligence (AAAI)*, p. 25–32.

REFANIDIS I. & VLAHAVAS I. P. (2003). Multiobjective heuristic state-space planning. *Artificial Intelligence*, **145**(1-2), 1–32.

SCHIEX T., FARGIER H. & VERFAILLIE G. (1995). Valued constraint satisfaction problems : Hard and easy problems. In *IJCAI*, p. 631–639.

VAN BEEK P. & CHEN X. (1999). Cplan : A constraint programming approach to planning. In *AAAI*, p. 585–590.