LCGP : une amélioration de Graphplan par relâchement de contraintes entre actions simultanées

LCGP : an improvement of Graphplan by relaxing constraints between simultaneous actions

M. Cayrol P. Régnier V. Vidal

IRIT
Université Paul Sabatier
118, route de Narbonne
31062 TOULOUSE CEDEX 04
{cayrol, regnier, vvidal}@irit.fr

Résumé

Les planificateurs de la famille de Graphplan sont aujourd'hui considérés comme étant les plus performants sur de nombreux domaines de planification. Les plans partiellement ordonnés qu'ils génèrent peuvent être représentés par des séquences d'ensembles d'actions simultanées. Conformément au modèle choisi, Graphplan contraint fortement - au travers du concept d'indépendance - le choix des actions au sein de ces ensembles d'actions. Nous montrons ici que l'on peut relâcher en partie ce critère d'indépendance de manière à produire des plans valides au sens de Graphplan. Leur génération par l'algorithme LCGP ainsi obtenu nécessite moins de niveaux qu'avec Graphplan (le même nombre dans les pires cas). Nous présentons ensuite une étude expérimentale portant sur des jeux de tests classiques qui montre que l'ensemble des problèmes résolus "pratiquement" par Graphplan est strictement inclus dans celui des problèmes résolus "pratiquement" par LCGP, avec des performances qui sont très nettement supérieures dans la plupart des cas.

Mots Clef

Planification, Graphplan, parallélisme, performances.

Abstract

GRAPHPLAN's family planners are presently considered as the most efficient ones on numerous planning domains. Their partially ordered plans can be represented as sequences of sets of simultaneous actions. Using this representation and the criterion of independance, Graphplan constraints the choice of actions in such sets. We demonstrate that this criterion can be partially relaxed in order to produce valid plans in the sense of Graphplan. Our planner LCGP needs less levels than Graphplan to generate these plans (the same number in the worst cases). Then we present an experimental study which demonstrates that, in classical planning domains, Graphplan's set of "practically" solved problems is strictly included in LCGP's one. In most cases, these tests also demonstrate the best performances of LCGP.

Keywords

Planning, Graphplan, parallelism, performances.

1 Introduction

Depuis quelques années, la planification classique par synthèse de plans a connu une profonde évolution avec le développement d'un nouveau type de générateur de plans basé sur le planificateur Graphplan [BF95] [BF97]. Jusque là, deux grandes familles d'algorithmes étaient utilisées (avec une nette préférence des chercheurs pour la seconde) : les algorithmes de recherche dans les espaces d'états [FAR95] et les algorithmes de recherche dans les espaces de plans [WEL94] [KAM97].

Graphplan développe dans un premier temps, niveau par niveau, un espace de recherche plus compact qu'un graphe d'états appelé graphe de planification. Pour cette phase (dite phase de construction), il n'utilise pas toutes les informations (indispensables à l'obtention d'un plan-solution) prises en compte au fur et à mesure dans les autres approches (exclusions mutuelles entre variables d'état ou entre actions). Ces contraintes sont simplement calculées et enregistrées à chaque niveau du graphe sous forme d'exclusions mutuelles à la manière des CSP [KAM99]. Cet espace de recherche se développe donc plus facilement, mais en contrepartie son achèvement ne coïncide plus avec l'obtention d'un plan-solution qu'une deuxième phase (dite phase d'extraction) cherchera à extraire à partir du graphe de planification et des contraintes enregistrées. Comme nous le verrons plus loin, les plans engendrés peuvent être représentés par des séquences d'ensembles d'actions simultanées, chaque ensemble correspondant à un niveau du graphe de planification.

L'évaluation de l'efficacité de ces diverses techniques de planification a donné lieu à de nombreuses études. Les comparaisons entre la planification dans les espaces d'états et la planification dans les espaces de plans ont ainsi fourni des résultats contradictoires. [MBD94] et [BW94] montrent que lorsqu'on assimile la planification dans les espaces d'états à la planification dans les espaces de plans totalement ordonnés, cette dernière est généralement moins efficace que la planification dans les espaces de plans partiellement ordonnés. Au contraire, [VB94] et [VR99] (qui contient une synthèse des études comparatives les plus importantes) montrent que la comparaison devient favorable à la planification dans les espaces d'états si l'on utilise l'état courant du monde pour diriger la recherche (planificateur TLPLAN [BK96]) ou élaguer l'espace de recherche (planificateur VVPLAN [VR99]). Par contre, les comparaisons entre les techniques précédentes et Graphplan sont largement favorables à ce dernier (lorsqu'on n'utilise pas de connaissances sur le domaine).

Plusieurs techniques ont été employées pour améliorer Graphplan : la réduction de l'espace de recherche avant la phase d'extraction du plan [FL98] [NDK97], l'amélioration du langage de représentation des domaines et des problèmes [KNHD97] [WAS98] [GA99] [KOE98], l'amélioration de la phase d'extraction de la solution [KAM99] [KS99] [LF99] [FL99] [ZK99].

2 Notre démarche

Un des problèmes qui occupent actuellement les chercheurs s'intéressant à Graphplan est l'amélioration de son efficacité. Pour notre part, nous avons voulu analyser et formaliser scrupuleusement les concepts mis en oeuvre : actions, indépendance, plans, parallélisme...

Nous avons remarqué que dans un plan de Graphplan (séquence d'ensembles d'actions), le calcul de la situation finale E_f - qui résulte de l'application simultanée des actions d'un ensemble Q du plan à partir d'une situation initiale E_i - est indépendant de l'ordre d'application des actions de Q. La situation finale reste la même lorsque ces actions sont appliquées en une séquence quelconque, à condition que Q satisfasse une propriété P simple à vérifier ($P(Q)$ dénote que Q vérifie la propriété P). Elle s'obtient directement à partir de la situation initiale E_i et de l'ensemble Q des actions appliquées simultanément, soit $E_f = \mathrm{g}(E_i, Q)$.

Nous avons alors établi une propriété P' moins restrictive que P et plus facile à vérifier ($P(Q) \Rightarrow P'(Q)$) qui permet de garantir l'existence (sans nécessiter son calcul) d'au moins un séquencement S des actions de l'ensemble Q, séquencement dont l'application à E_i se calcule exactement de la même manière ($E_f = \mathrm{g}(E_i, Q)$). E_f ne peut toutefois plus être considéré comme résultant de l'exécution simultanée des actions de l'ensemble Q.

Nous avons ensuite développé une variante de Graphplan appelée LCGP (acronyme de Least Committed GraphPlan) qui procède de manière similaire en construisant incrémentalement un graphe "stratifié" avant d'en extraire un plan-solution (lorsqu'il en existe un). Le graphe qu'aurait construit Graphplan est un sous-graphe de celui construit par LCGP à profondeur égale. Ainsi les buts apparaissent-ils en général plus tôt (en même temps dans les pires des cas). LCGP transforme ensuite les plans qu'il a produits en plans conformes aux contraintes imposées par Graphplan. L'obtention plus précoce de plans-solutions se fait au détriment de la perte de l'optimalité de ces plans au sens où elle est définie par Graphplan (nombre de niveaux de Graphplan). Mais elle se traduit dans la pratique par des temps de réponse beaucoup plus réduits pour LCGP quand Graphplan est incapable de produire une solution au bout d'un temps considérable. Le dilemme disparaît quand il faut choisir entre un plan (pas forcément optimal...) qui existe et un plan optimal qui n'a pas encore été produit.

Dans le chapitre suivant (cf. § 3), nous allons formaliser la structure des plans générés par Graphplan avant de suggérer que, pour les obtenir, Graphplan contraint trop fortement - au travers du concept d'indépendance entre actions - le choix des actions au sein d'un ensemble d'actions simultanées. Nous montrerons ensuite (cf. § 4) que l'on peut relâcher ce critère d'indépendance entre actions en modifiant ainsi la structure des plans produits.

3 Sémantique et formalisation des plans de Graphplan

Le constituant principal d'un plan est l'*action*, qui est une instance de base d'un *opérateur*. Dans Graphplan, les opérateurs sont de type STRIPS, sans négation dans les préconditions. Nous nous plaçons dans un langage L de la logique du premier ordre construit à partir des vocabulaires Vx, Vc, Vp qui dénotent respectivement des ensembles finis disjoints deux à deux de symboles de variables, de constantes et de prédicats. Nous n'utiliserons pas ici de symboles de fonctions.

Définition 1 : Un *opérateur* dénoté par o est un triplet $\langle pr, ad, de\rangle$ où pr, ad et de dénotent des ensembles finis de formules atomiques du langage L. Prec(o), Add(o) et Del(o) dénotent respectivement les ensembles pr, ad, et de de o. O dénote l'ensemble (fini) des opérateurs.

Définition 2 : Un *état* est un ensemble fini de formules atomiques concrètes (i.e. ne contenant aucun symbole de variable). Une formule atomique concrète sera aussi appelée *proposition*. P dénote l'ensemble de toutes les propositions construites sur le langage L.

Définition 3 : Une *action* dénotée par a est une instance concrète $o\theta = \langle pr\theta, ad\theta, de\theta\rangle$ d'un opérateur o résultant de l'application d'une substitution θ définie dans le langage L telle que $ad\theta \cap de\theta = \emptyset$. Prec($a$), Add($a$), Del($a$) dénotent respectivement les ensembles $pr\theta$, $ad\theta$, $de\theta$ et représentent les préconditions, les ajouts et les retraits de l'action a.

Définition 4 : L'*ensemble des actions* provenant de toutes les instanciations concrètes possibles des opérateurs de O est fini et sera noté A.

La structure essentielle que nous définissons maintenant, la *séquence d'ensembles d'actions*, nous permettra par la suite de représenter les plans de Graphplan et de LCGP.

Définition 5 : Une *séquence d'ensembles d'actions* est une suite finie et ordonnée d'ensembles finis d'actions. Si $Q_1, Q_2, ..., Q_n$ sont des ensembles finis d'actions tels que Q_1 précède Q_2, ..., Q_{n-1} précède Q_n, la séquence d'ensembles d'actions correspondante sera notée $\langle Q_1, Q_2, ..., Q_n\rangle$. Si les ensembles d'actions sont des singletons (i.e. $Q_1 = \{a_1\}$, $Q_2 = \{a_2\}$, ..., $Q_n = \{a_n\}$), la séquence d'ensembles d'actions correspondante sera appelée une *séquence d'actions* et sera notée par abus de langage $\langle a_1, a_2, ..., a_n\rangle$. L'ensemble des séquences finies d'ensembles finis d'actions formées à partir d'un ensemble d'actions A sera noté $(2^A)^*$. L'ensemble des séquences d'actions formées à partir d'un ensemble d'actions A sera noté A^*.

Définition 6 : Soit $Q = \langle Q_1, ..., Q_n\rangle$ et $R = \langle R_1, ..., R_m\rangle$ deux séquences d'ensembles d'actions. La *concaténation* (notée $\oplus$) de deux séquences d'ensembles d'actions est définie par :

$$Q \oplus R = \langle Q_1, ..., Q_n, R_1, ..., R_m\rangle.$$

Propriété 1 : La concaténation de séquences d'ensembles d'actions est une loi de composition interne dans $(2^A)^*$ qui possède les propriétés suivantes :

1. élément neutre : $\forall Q \in (2^A)^*, Q \oplus \langle\rangle = \langle\rangle \oplus Q = Q$,
2. associativité : $\forall Q_1, Q_2, Q_3 \in (2^A)^*, (Q_1 \oplus Q_2) \oplus Q_3 = Q_1 \oplus (Q_2 \oplus Q_3) = Q_1 \oplus Q_2 \oplus Q_3$,
3. $\forall S_1, S_2 \in A^*, S_1 \oplus S_2 \in A^*$.

Définition 7 : Une *linéarisation* d'un ensemble d'actions $Q = \{a_1, ..., a_n\}$ est une permutation de Q, c'est-à-dire une séquence d'actions S telle qu'il existe une bijection $b : [1, n] \to Q$ avec $S = \langle b(1), ..., b(n)\rangle$. La linéarisation de l'ensemble vide $\{\}$ est la séquence vide $\langle\rangle$. L'ensemble de toutes les linéarisations de Q sera noté $\mathrm{Lin}(Q)$.

Notations : Soit l'ensemble d'actions $Q = \{a_1, ..., a_n\}$:

- l'union des préconditions des éléments de Q sera notée $\mathrm{Prec}(Q)$: $\mathrm{Prec}(Q) = \mathrm{Prec}(a_1) \cup ... \cup \mathrm{Prec}(a_n)$,
- l'union des ajouts des éléments de Q sera notée $\mathrm{Add}(Q)$: $\mathrm{Add}(Q) = \mathrm{Add}\ (a_1) \cup ... \cup \mathrm{Add}(a_n)$,
- l'union des retraits des éléments de Q sera notée $\mathrm{Del}(Q)$: $\mathrm{Del}(Q) = \mathrm{Del}\ (a_1) \cup ... \cup \mathrm{Del}(a_n)$.

Ceci s'applique aussi dans le cas où Q est la séquence d'actions $Q = \langle a_1, ..., a_n\rangle$.

Dans un plan de Graphplan, un ensemble d'actions représente des actions qui peuvent être exécutées dans n'importe quel ordre, voire même en parallèle, le résultat étant indépendant du mode d'application. Dans cet article, nous emploierons systématiquement le terme *d'ensemble d'actions simultanées* - et non pas d'ensemble d'actions parallèles - pour mettre l'accent sur le fait que les actions appartiennent à un même ensemble (et sont issues d'un même niveau du graphe de planification) sans préjuger de l'ordre selon lequel on pourrait les exécuter par la suite. Le terme *d'ensemble d'actions parallèles* sera lui, réservé à des ensembles d'actions qu'il est possible d'exécuter en parallèle.

Graphplan n'impose aucune hypothèse sur la façon dont l'exécution peut se dérouler dans le monde réel : on ne connaît pas la durée d'exécution d'une action, on ne sait pas combien de temps met un effet pour être établi, ni combien de temps une précondition doit rester vraie pour permettre d'exécuter une action... Toutes ces incertitudes obligent Graphplan, comme beaucoup de planificateurs qui génèrent des plans partiellement ordonnés (UCPOP, SNLP...), à contraindre fortement le choix des actions *afin que le résultat de l'exécution des actions d'un ensemble en série (dans n'importe quel ordre) conduise au même état résultant que l'exécution en parallèle de ces mêmes actions*.

Pour atteindre cet objectif avec une description STRIPS des actions, il faut que deux actions soient *indépendantes* (cf. Définition 8) ; c'est-à-dire que leurs effets ne se contredisent pas (une action ne doit pas retirer une proposition ajoutée par l'autre action) et qu'elles n'aient pas d'interactions croisées (une action ne doit pas retirer une proposition qui est une précondition de l'autre). Un ensemble *indépendant* (cf. Définition 9) représentera donc un ensemble d'actions parallèles puisque les actions de cet ensemble y seront toutes indépendantes deux à deux.

Remarquons qu'il semble manquer une condition : pour être exécutables en parallèle, deux actions d'un même ensemble doivent aussi ne pas avoir de préconditions incompatibles. Graphplan et LCGP détectent et exploitent ces cas d'incompatibilité.

Une séquence $\langle Q_1, ..., Q_n\rangle$ d'ensembles d'actions simultanées définit partiellement l'ordre d'exécution des actions des ensembles. La fin de l'exécution de chaque action de Q_i doit précéder le début de celle de chaque action de Q_{i+1}, ce qui implique que l'exécution de toutes les actions de Q_i précède celle des actions de Q_{i+1}.

Définition 8 : Deux actions $a_1 \neq a_2$ sont *indépendantes* ssi :

$$(\mathrm{Add}(a_1) \cup \mathrm{Prec}(a_1)) \cap \mathrm{Del}(a_2) = \varnothing$$
$$\text{et } (\mathrm{Add}(a_2) \cup \mathrm{Prec}(a_2)) \cap \mathrm{Del}(a_1) = \varnothing.$$

Définition 9 : Un ensemble d'actions Q est un *ensemble indépendant* ssi les actions de cet ensemble sont indépendantes deux à deux, c'est-à-dire :

$$\forall a_1 \neq a_2 \in Q, (\mathrm{Prec}(a_1) \cup \mathrm{Add}(a_1)) \cap \mathrm{Del}(a_2) = \varnothing.$$

Nous allons maintenant préciser la formalisation d'un plan de Graphplan en définissant une application qui permet d'appliquer une séquence d'ensembles d'actions à un état. Cette application permet de simuler l'exécution d'un plan à partir d'une représentation initiale du monde. Si une séquence d'ensembles d'actions n'est pas applicable à un état pour une raison ou pour une autre, le résultat retourné sera $\perp$, que l'on peut dénoter comme étant l'état impossible.

Définition 10 : Soit $\mathrm{Res} : (2^P \cup \{\perp\}) \times (2^A)^* \to (2^P \cup \{\perp\})$ définie par :

$\mathrm{Res}(E, Q) =$
Si $Q = \langle\rangle$ ou $E = \perp$
  alors $\mathrm{Res}(E, Q) = E$
  sinon
    Si $Q = \langle Q_1\rangle$
      alors Si Q_1 est **indépendant** et $\mathrm{Prec}(Q_1) \subseteq E$

alors Res(E, Q) = (E - Del(Q_1)) $\cup$ Add(Q_1)
sinon Res(E, Q) = $\perp$
sinon $Q = \langle Q_1, ..., Q_n \rangle$ et
Res(E, Q) = Res(Res(E, $\langle Q_1, ..., Q_{n-1}\rangle$), $\langle Q_n \rangle$).

Définition 11 : Une séquence d'ensembles d'actions Q constitue un *plan* pour un état E relativement à Res ssi Res(E, Q) $\neq \perp$.

En effet, dans ce cas, on peut associer à Q une sémantique qui correspond à l'exécution d'actions du monde réel, car on est sûr (du moins dans un monde statique, sans imprévu) que la prédiction que l'on fait de l'état final est bonne.

Nota bene : Par manque de place et pour ne pas compliquer inutilement la lecture, nous ne donnerons que des squelettes de preuves ; pour les démonstrations complètes, le lecteur se reportera à [VCR99].

Propriété 2 : L'application successive de Res à deux séquences d'ensembles d'actions et à un état donne le même résultat que l'application de la concaténation de ces deux séquences à ce même état. Soient deux séquences d'ensembles d'actions $S_1 = \langle Q_1, ..., Q_n \rangle$ et $S_2 = \langle R_1, ..., R_n \rangle$. On a alors :

$$\forall E \in (2^P \cup \{\perp\}), \mathrm{Res}(E, S_1 \oplus S_2) = \mathrm{Res}(\mathrm{Res}(E, S_1), S_2)$$

Preuve : Par induction sur la taille de S_2 ♦

Nous allons maintenant établir, dans le Théorème 1, la propriété qui sert de base à Graphplan : *les actions d'un plan de Graphplan qui peuvent être exécutées en parallèle donnent le même résultat lorsqu'elles sont exécutées en série, quel que soit l'ordre d'exécution.*

Théorème 1 : Soient un état $E \in 2^P$ et une séquence d'ensembles d'actions $Q \in (2^A)^*$, avec $Q = \langle Q_1, ..., Q_n \rangle$. On a alors :

$$\mathrm{Res}(E, Q) \neq \perp \Rightarrow \forall S_1 \in \mathrm{Lin}(Q_1), ..., \forall S_n \in \mathrm{Lin}(Q_n),$$
$$\mathrm{Res}(E, S_1 \oplus ... \oplus S_n) = \mathrm{Res}(E, Q).$$

Preuve : Elle s'appuie sur les deux lemmes suivants.

Lemme 1 : Soient $A, A_1, ..., A_n, B_1, ..., B_n$ des ensembles tels que $\forall i \in [1, n\text{-}1], A_{i+1} \cap (B_1 \cup ... \cup B_i) = \emptyset$. On a alors :

$$(A - (A_1 \cup ... \cup A_n)) \cup (B_1 \cup ... \cup B_n)$$
$$= ((...((A - A_1) \cup B_1) - ...) - A_n) \cup B_n.$$

Le lemme qui suit va nous être utile pour calculer l'application d'une séquence d'actions à un état (différent de $\perp$) lorsque ce dernier contient toutes les préconditions de chaque action de la séquence et lorsque une action ne détruit jamais les préconditions d'une action qui lui succède (immédiatement ou non). Dans ce cas particulier, le résultat est toujours différent de $\perp$.

Lemme 2 : Soient un état $E \in 2^P$ et une séquence d'actions $S \in A^*$, avec $S = \langle a_1, a_2, ..., a_n \rangle$, tels que : Prec($S$) $\subseteq E$ et $\forall i \in [1, n\text{-}1]$ Prec(a_{i+1}) $\cap$ (Del(a_1) $\cup ... \cup$ Del(a_i)) = $\emptyset$. On a alors :

$$\mathrm{Res}(E, S) = ((...((((E - \mathrm{Del}(a_1)) \cup \mathrm{Add}(a_1)) - \mathrm{Del}(a_2))$$
$$\cup \mathrm{Add}(a_2)) \cup ...) - \mathrm{Del}(a_n)) \cup \mathrm{Add}(a_n)).$$

Pour prouver le Théorème 1, on montre d'abord que si $Q = \langle Q_1 \rangle$, Res(E, Q) $\neq \perp \Rightarrow \forall S \in$ Lin(Q_1), Res(E, Q) = Res(E, S). Ceci vient du fait que Q_1 est indépendant et que Prec(Q_1) $\subseteq E$ (sinon on aurait Res(E, Q) = $\perp$). En utilisant le Lemme 1, on calcule Res(E, Q) ; et en utilisant le Lemme 2, on calcule Res(E, S) avec $S \in$ Lin(Q_1). On trouve bien que Res(E, Q) = Res(E, S). Dans le cas général, lorsque $Q = \langle Q_1, ..., Q_n \rangle$, on effectue une induction sur Q en utilisant la Propriété 2 et le résultat que l'on vient de démontrer ♦

Notre travail va maintenant consister à remettre en cause cette propriété. Observons que pour $S = \langle a_1, a_2, ..., a_n \rangle$, Res($E$, S) = (E - Del(S)) $\cup$ Add(S) quand $\forall i \in [1, n\text{-}1]$, Del($a_{i+1}$) $\cap$ (Add(a_1) $\cup ... \cup$ Add(a_i)) = $\emptyset$ et $\forall i \in [1, n\text{-}1]$, Prec($a_{i+1}$) $\cap$ (Del(a_1) $\cup ... \cup$ Del(a_i)) = $\emptyset$. Remarquons que dans ce cas, le calcul de Res(E, $\langle a_1, ..., a_n \rangle$) peut se faire sans connaître l'ordre des éléments de la séquence $\langle a_1, a_2, ..., a_n \rangle$.

4 Vers une nouvelle structure des plans

Ainsi que nous l'avons vu (§ 3), Graphplan impose des contraintes fortes sur les plans par le biais de la propriété d'indépendance entre les actions simultanées d'un ensemble. Nous allons montrer que l'on peut modifier cette propriété pour lever une partie de ces contraintes.

Ce faisant, on ne pourra plus assurer que des actions d'un même ensemble d'actions (du même niveau) peuvent être exécutées en parallèle puisqu'elles ne seront plus indépendantes. Mais l'idée essentielle consiste à conserver ce qui fait la puissance de Graphplan : nous continuerons à traiter ces nouveaux ensembles d'actions en les considérant "en bloc", c'est-à-dire en cherchant à établir toutes les préconditions de toutes les actions d'un ensemble grâce aux effets des actions d'un autre ensemble d'actions du niveau précédent.

En relâchant les contraintes existant sur les actions indépendantes, nous définirons une relation plus souple entre les actions, qui n'est plus symétrique : la relation d'*autorisation*. Nous dirons qu'une action a_1 autorise une action a_2 si a_2 peut être exécutée (en l'absence d'autres informations) soit en même temps que a_1, soit après. Pour que ceci soit possible, il nous suffit de ne garder que deux des conditions qui existaient sur les actions indépendantes : a_1 ne doit pas détruire une précondition de a_2 (a_2 doit rester exécutable) et a_2 ne doit pas détruire un fait établit par a_1 (les effets des deux actions sont l'union des effets de chacune). Cette définition implique une notion de précédence sur l'exécution de deux actions. En effet, dire que a_1 autorise a_2 signifie que si l'on exécute a_1 avant a_2, les ajouts de a_1 sont conservés par l'exécution de a_2 et les préconditions de a_2 sont conservées par l'exécution de a_1. Par contre, si a_1 n'autorise pas a_2 et que l'on veuille toujours exécuter a_1 avant a_2, soit un ajout de a_1 est détruit par a_2 (ce qui pose un problème pour déterminer l'état résultant), soit une précondition de a_2 est détruite par a_1 (rendant impossible l'exécution de a_2).

Définition 12 : Une action a_1 *autorise* une action a_2 (noté $a_1 \angle a_2$) ssi (1) $a_1 \neq a_2$ et (2) $\mathrm{Add}(a_1) \cap \mathrm{Del}(a_2) = \varnothing$ et $\mathrm{Prec}(a_2) \cap \mathrm{Del}(a_1) = \varnothing$. Une action a_1 *interdit* une action a_2 ssi l'action a_1 n'autorise pas l'action a_2, c'est-à-dire si non($a_1 \angle a_2$).

Nota bene : La relation *autorise* n'est pas en général une relation d'ordre.

Ceci nous conduit à une nouvelle définition des ensembles pouvant faire partie d'un plan, qui ne seront plus des ensembles indépendants. Ce que nous voulons, c'est que l'on puisse trouver pour chaque ensemble d'actions au moins une linéarisation qui puisse être un plan. C'est par la recherche d'une telle linéarisation qu'est introduite la notion de précédence entre les actions.

Définition 13 : Une séquence $S = \langle a_1, ..., a_n \rangle$ sera dite *autorisée* ssi $\forall\, i, j \in [1, n],\, i < j \Rightarrow a_i \angle a_j$, c'est-à-dire :

$$\forall\, i \in [1,n\text{-}1],\ \mathrm{Del}(a_{i+1}) \cap (\mathrm{Add}(a_1) \cup ... \cup \mathrm{Add}(a_i)) = \varnothing \text{ et } \mathrm{Prec}(a_{i+1}) \cap (\mathrm{Del}(a_1) \cup ... \cup \mathrm{Del}(a_i)) = \varnothing.$$

Définition 14 : Un ensemble d'actions Q est dit *autorisé* (dans le cas contraire, il est *interdit*) ssi on peut trouver une linéarisation $S \in \mathrm{Lin}(Q)$ qui soit autorisée. Nous noterons $\mathrm{LinA}(Q)$ l'ensemble de toutes les linéarisations autorisées de Q : $\mathrm{LinA}(Q) = \{S \in \mathrm{Lin}(Q) \mid S$ est une linéarisation autorisée$\}$.

Cette définition signifie qu'un ensemble d'actions est autorisé si on peut trouver un ordre entre les actions de cet ensemble, tel qu'aucune action ne détruise ni l'ajout d'une action qui la précède, ni une précondition d'une action qui lui succède.

Nous allons maintenant définir Res*, nouvelle application d'une séquence d'ensembles d'actions à un état, qui va faire intervenir non plus l'indépendance, mais l'autorisation entre actions. Res* servira de base à l'algorithme LCGP. Nous verrons ensuite que cette nouvelle définition permet de démontrer un nouveau théorème de détermination de l'état résultant qui ne fait plus intervenir toutes les linéarisations des ensembles d'actions indépendantes, mais uniquement les linéarisations qui respectent les contraintes d'autorisation entre les actions (les linéarisations autorisées). La Propriété 2 et les lemmes 1 et 2 qui constituaient l'essentiel de la preuve du Théorème 1 seront conservés.

Définition 15 : Soit $\mathrm{Res}^* : (2^P \cup \{\perp\}) \times (2^A)^* \to (2^P \cup \{\perp\})$ définie par :

$\mathrm{Res}^*(E, Q) =$
Si $Q = \langle\rangle$ ou $E = \perp$
  alors $\mathrm{Res}^*(E, Q) = E$
  sinon
    Si $Q = \langle Q_1 \rangle$
      alors Si Q_1 est **autorisé** et $\mathrm{Prec}(Q_1) \subseteq E$
          alors $\mathrm{Res}^*(E, Q) = (E - \mathrm{Del}(Q_1)) \cup \mathrm{Add}(Q_1)$
          sinon $\mathrm{Res}^*(E, Q) = \perp$
      sinon $Q = \langle Q_1, ..., Q_n \rangle$ et
        $\mathrm{Res}^*(E, Q) = \mathrm{Res}^*(\mathrm{Res}^*(E, \langle Q_1, ..., Q_{n-1} \rangle), \langle Q_n \rangle)$.

Définition 16 : Une séquence d'ensembles d'actions Q constitue un *plan* pour un état E relativement à Res* ssi $\mathrm{Res}^*(E, Q) \neq \perp$.

En effet, dans ce cas, on peut associer à Q une sémantique (différente de celle associée aux plans reconnus par Res) qui correspond à l'exécution d'actions du monde réel, car on est sûr (du moins dans un monde statique, sans imprévu) que la prédiction que l'on fait de l'état final est bonne.

Nous pouvons maintenant montrer un théorème proche du Théorème 1 : le résultat de l'application de toutes les linéarisations **autorisées** d'un plan de Res* est toujours le même. Sa démonstration utilise la Propriété 1 et les Lemmes 1 et 2 que nous avons démontrés pour Res et qui restent vrais pour Res*. Nous pouvons poser :

Proposition 1 : La Propriété 2, le Lemme 1 et le Lemme 2 restent vrais en remplaçant Res par Res*.

Nous pouvons maintenant écrire le nouveau théorème d'unicité de l'état résultant. La preuve de ce théorème suit le même schéma que celle du Théorème 1.

Théorème 2 : Soient un état $E \in 2^P$ et une séquence d'ensembles d'actions $Q \in (2^A)^*$, avec $Q = \langle Q_1, ..., Q_n \rangle$:

$$\mathrm{Res}^*(E, Q) \neq \perp \Rightarrow \forall\, S_1 \in \mathrm{LinA}(Q_1), ..., \forall\, S_n \in \mathrm{LinA}(Q_n),\ \mathrm{Res}^*(E, S_1 \oplus ... \oplus S_n) = \mathrm{Res}^*(E, Q).$$

5 Correspondance entre les deux formalismes

En raison de la forte similitude qui existe entre les relations de dépendance et d'autorisation, les deux formalismes que nous venons de présenter sont très liés. Il en résulte qu'un plan pour Res est un plan pour Res* :

Théorème 3 : Soit un état $E \in 2^P$ et une séquence d'ensembles d'actions $Q \in (2^A)^*$. On a alors :

$$\mathrm{Res}(E, Q) \neq \perp \Rightarrow \mathrm{Res}^*(E, Q) = \mathrm{Res}(E, Q).$$

Preuve : On montre qu'un ensemble d'actions indépendant est autorisé. La seule différence entre Res et Res* se situant à ce niveau, la preuve en découle immédiatement ♦

On peut aussi montrer que si une séquence d'ensembles d'actions Q n'est pas un plan pour une situation E relativement à Res*, alors elle ne l'est pas pour E relativement à Res : $\mathrm{Res}^*(E, Q) = \perp \Rightarrow \mathrm{Res}(E, Q) = \perp$.

Il existe un autre lien très fort entre les plans reconnus par Res* et ceux reconnus par Res : tous les plans construits en utilisant les linéarisations autorisées des ensembles d'actions d'un plan reconnu par Res* sont des plans reconnus par Res. De plus, l'état résultant de l'application de Res* sur le plan originel est le même que l'état résultant de l'application de Res sur n'importe quel plan construit en utilisant des linéarisations autorisées des ensembles d'actions du plan :

Théorème 4 : Soit un état $E \in 2^P$ et une séquence d'ensembles d'actions $Q \in (2^A)^*$, avec $Q = \langle Q_1, ..., Q_n \rangle$. On a alors :

$$\text{Res}^*(E, Q) \neq \bot \Rightarrow \forall S_1 \in \text{LinA}(Q_1), ..., \forall S_n \in \text{LinA}(Q_n), \text{Res}^*(E, Q) = \text{Res}(E, S_1 \oplus ... \oplus S_n).$$

Preuve : D'après le Théorème 2, on a : $\forall S_1 \in \text{LinA}(Q_1)$, ..., $\forall S_n \in \text{LinA}(Q_n)$, $\text{Res}^*(E, Q) = \text{Res}^*(E, S_1 \oplus ... \oplus S_n)$. Soit $S \in A^*$ avec $S = S_1 \oplus ... \oplus S_n$ et $S_1 \in \text{LinA}(Q_1), ..., S_n \in \text{LinA}(Q_n)$. Comme $\text{Res}^*(E, Q) \neq \bot$, on a forcément $\text{Res}^*(E, S) \neq \bot$. Or $S \in A^*$ donc tous les ensembles d'actions qui composent S sont des singletons : ce sont des ensembles indépendants. On a donc $S = \langle a_1, ..., a_m\rangle$ et tous les $\{a_i\}$ sont des ensembles d'actions indépendants. De plus, comme $\text{Res}^*(E, S) \neq \bot$, après chaque application d'une action a_i de S, les préconditions de a_{i+1} sont vérifiées. De ces deux conditions, on peut déduire que $\text{Res}(E, S) \neq \bot$; d'après le Théorème 1, on peut en déduire que $\text{Res}^*(E, S) = \text{Res}(E, S)$ et donc $\text{Res}^*(E, Q) = \text{Res}(E, S)$ ♦

Ce théorème est essentiel puisqu'il donne un sens aux plans que reconnaît Res* : une simple transformation (la recherche d'une linéarisation autorisée pour chaque ensemble d'actions) fournit un plan, qui est une séquence d'actions, reconnu par Res. Il s'agit donc de plans qui pourraient être trouvés par l'algorithme originel de Graphplan.

6 Intégration de la nouvelle structure des plans dans Graphplan

Nous allons maintenant étudier la façon de modifier Graphplan afin d'implémenter notre nouveau formalisme dans LCGP. Nous ne détaillerons ici ni nos algorithmes, ni les définitions qui s'y attachent (pour plus de détails, le lecteur se reportera à [VCR99]). De manière informelle, on retiendra qu'un graphe de planification est un graphe constitué de niveaux successifs repérés par des entiers naturels, chaque niveau étant composé d'un ensemble d'actions et d'un ensemble de propositions. Le niveau 0 fait exception et ne contient que des propositions qui représentent les faits de l'état initial.

6.1 Construction du graphe de planification

Lors de cette phase, la seule différence entre Graphplan et LCGP se situe au niveau du calcul des exclusions mutuelles entre actions. Dans Graphplan, deux actions a_1 et a_2 s'excluent mutuellement ssi (1) elles sont distinctes et (2) ssi elles ne sont pas indépendantes (i.e. l'une des deux interdit l'autre : non($a_1 \angle a_2$) **ou** non($a_2 \angle a_1$)), ou si une précondition de l'une est mutuellement exclusive avec une précondition de l'autre. Dans LCGP, l'exclusion mutuelle entre actions est définie ainsi :

Définition 17 : Deux actions sont mutuellement exclusives ssi (1) elles sont distinctes, et (2) si elles s'interdisent mutuellement : non($a_1 \angle a_2$) **et** non($a_2 \angle a_1$), ou si une précondition de l'une est mutuellement exclusive avec une précondition de l'autre.

Cette nouvelle définition de l'exclusion mutuelle (**ou** dans Graphplan, **et** dans LCGP), implique que LCGP trouve moins de paires d'actions mutuellement exclusives que Graphplan (au pire, autant). Un niveau n de LCGP comprendra donc plus d'actions et de propositions qu'un niveau n de Graphplan, puisque les actions peuvent parfois être déclenchées plus tôt dans LCGP. Le graphe de planification de ce dernier va donc grossir plus vite et contiendra, pour un même nombre de niveaux, plus de plans potentiels que celui de Graphplan (au pire, autant). L'arrêt de la construction de ce graphe se produit également plus tôt car les buts y apparaissent en général avant d'être produits par Graphplan (en même temps dans les pires des cas).

6.2 Recherche du plan-solution

Après avoir construit le graphe de planification, Graphplan cherche à en extraire un plan-solution. Pour cela, il part de l'ensemble des propositions construit au dernier niveau (qui contient les buts) et considère les différents ensembles d'actions qui permettent d'établir les buts. Il choisit l'un de ces ensembles (point de backtrack de l'algorithme) et, en passant au niveau précédent, cherche à nouveau les différents ensembles d'actions qui permettent d'établir l'ensemble des préconditions des actions précédemment choisies... A chaque niveau, les actions d'un ensemble doivent être indépendantes deux à deux et leurs préconditions ne doivent pas être contradictoires afin de se conformer à la sémantique associée (parallélisme des actions, cf. § 3). Pour cela, Graphplan vérifie simplement, en utilisant les exclusions mutuelles enregistrées pendant la construction, qu'il n'existe aucune paire d'actions mutuellement exclusives.

Dans LCGP, les exclusions mutuelles ne suffisent pas pour retenir un ensemble d'actions. Il faut en plus que cet ensemble soit autorisé (au sens de la Définition 14), c'est-à-dire que l'on puisse trouver une séquence d'actions (séquence autorisée) telle qu'une action ne détruise ni la précondition d'une action qui la suit, ni un ajout d'une action qui la précède. Ceci peut être réalisé en vérifiant que le graphe suivant, pour l'ensemble d'actions considéré, est un graphe dirigé acyclique (GDA) :

Définition 18 : Soit $Q \in 2^A$ un ensemble d'actions, avec $Q = \{a_1, ..., a_n\}$. Le *graphe d'autorisation* $GA(N, C)$ de l'ensemble Q est un graphe orienté tel que :

- N est l'ensemble des noeuds tels qu'à chaque action a_i correspond un noeud unique de N noté n(a_i) : $N = \{\text{n}(a_1), ..., \text{n}(a_n)\}$,
- C est l'ensemble des arcs, qui traduisent les contraintes de précédence entre les actions : il y a un arc d'un noeud associé à une action a_i vers un noeud associé à une action a_j si et seulement si l'exécution de a_i **doit précéder** celle de a_j, c'est-à-dire ssi a_j interdit a_i :
$$\forall a_i \neq a_j \in Q, (\text{n}(a_i), \text{n}(a_j)) \in C \Leftrightarrow \text{non}(a_j \angle a_i).$$

En effet, on peut montrer que :

Théorème 5 : Soit $Q \in 2^A$ un ensemble d'actions et $GA(N, C)$ le graphe d'autorisation de Q. On a alors :

$$GA \text{ n'a pas de circuit} \Leftrightarrow Q \text{ est autorisé.}$$

Un algorithme de tri topologique légèrement modifié (polynomial) nous permet de détecter simplement la présence d'un circuit dans le graphe.

6.3 Retour du plan-solution

Le plan-solution que retourne LCGP n'est pas reconnu par Res (qui reconnaît les plans de Graphplan) mais par Res*. On peut transformer un plan que retourne LCGP en un plan reconnu par Res en procédant en deux temps :

1. On transforme d'abord le plan retourné par LCGP en un plan linéaire reconnu par Res (cf. Théorème 4). Pour ceci, on modifie l'algorithme du § 6.2 (recherche d'un cycle dans un graphe d'autorisation) afin qu'il retourne une séquence autorisée pour chaque ensemble d'actions du plan. On peut même faire en sorte qu'il retourne une séquence d'ensembles d'actions telle que toutes ses linéarisations soient des linéarisations autorisées.
2. Le plan obtenu peut ensuite être transformé en un plan parallèle grâce à l'algorithme polynomial de [RF91], révisé et formalisé par [BAC98] qui montre qu'il trouve le réordonnancement optimal en nombre d'étapes (i.e. en nombre d'ensembles d'actions indépendantes) du plan fourni.

6.4 Exemple

Afin d'illustrer la différence de fonctionnement entre Graphplan et LCGP, étudions l'exemple suivant. L'ensemble des propositions est $P = \{a, b, c, d\}$ et l'ensemble des actions est $\{A, B, C\}$, avec :

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

L'état initial du problème est $I = \{a\}$, et le but est $B = \{d\}$. La Figure 1 représente le graphe de planification obtenu avec Graphplan.

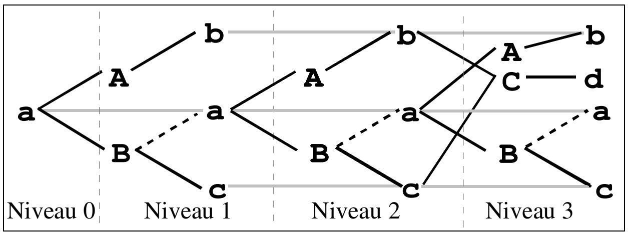

Figure 1 : Le graphe de planification avec Graphplan

Les actions A et B sont mutuellement exclusives, car B enlève a qui est une précondition de A. Au niveau 1, les paires de propositions mutuellement exclusives sont donc $\{a, c\}$ et $\{b, c\}$. On ne peut donc pas utiliser l'action C (qui donne le but) au niveau 2. Dans ce niveau, b et c ne sont plus mutuellement exclusives. On peut donc appliquer l'action C au niveau 3. Le plan-solution ainsi produit est $\langle A, B, C\rangle$.

La Figure 2 représente maintenant le graphe de planification obtenu avec LCGP.

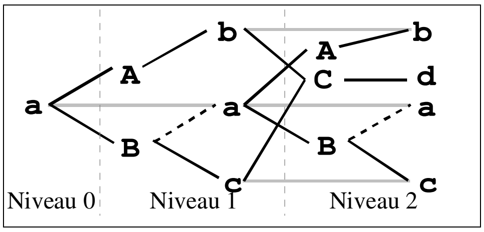

Figure 2 : Le graphe de planification avec LCGP

La différence essentielle est le fait que A et B ne sont plus mutuellement exclusives, car A autorise B ($A \angle B$). Les propositions b et c ne sont donc plus mutuellement exclusives au niveau 1, et on peut appliquer l'action C au niveau 2. Le plan obtenu est $\langle\{A, B\}, \{C\}\rangle$, qui n'est pas reconnu par Res : $\{A, B\}$ n'est pas un ensemble d'actions indépendant, mais autorisé. En appliquant la transformation détaillée au § 6.3, on obtient le même plan-solution que Graphplan : $\langle A, B, C\rangle$.

7 Évaluation expérimentale

Nous allons maintenant présenter les résultats de notre étude expérimentale sur les performances comparées de LCGP et Graphplan. Pour obtenir des résultats significatifs (il est toujours délicat de comparer deux programmes implémentés par des personnes différentes dans des langages différents), nous avons réimplémenté Graphplan (implémentation validée en la comparant avec le planificateur STAN [LF99] version 3, cf. Table 1). Ainsi, notre version de Graphplan et LCGP partagent la majeure partie de leur code et les différences sont donc peu importantes (cf. § 6) : dans la phase de construction du graphe, l'autorisation entre actions remplace l'indépendance ; la phase d'extraction du plan nécessite la vérification des ensembles autorisés et, avant de le retourner, on convertit le plan-solution. La partie commune comporte des améliorations connues comme les techniques CSP de [KAM99] et une construction du graphe inspirée de [LF99]. Graphplan et LCGP sont implémentés en CMU Common Lisp, et les tests ont été exécutés sur un PC Intel PII 450 Mhz avec 256Mo de RAM tournant sous Debian GNU/Linux 2.0.

7.1 Le domaine Logistics

La version de ce domaine et les problèmes des tests sont ceux de la version 3.4 du système BLACKBOX [KS99]. Ce domaine permet d'exprimer des problèmes de transport d'objets dans des villes découpées en quartiers (aéroport, centre ville...). Les moyens de transport utilisables sont l'avion, qui permet de transporter des objets d'un aéroport vers un autre, et le camion, qui permet de transporter des objets d'un quartier vers un autre à l'intérieur d'une même ville. Avions et camions peuvent contenir un nombre d'objets quelconque. Un problème classique consistera à transporter une dizaine d'objets répartis dans différents quartiers de différentes villes vers d'autres endroits, à l'aide des moyens de transport disponibles (typiquement, 5 ou 6 villes, 2 ou 3 quartiers par ville, 1 à 3 avions, 1 camion par ville).

Dans ce domaine, les plans possèdent beaucoup d'actions parallèles : charger un objet dans l'avion 1 peut se faire en même temps que déplacer un camion de l'aéroport vers le centre ville, alors que l'avion 2 vole d'un aéroport vers un autre. Graphplan va donc trouver un grand nombre d'actions indépendantes, et par conséquent il y aura beaucoup moins d'exclusions mutuelles entre actions que dans un domaine très contraint, comme par exemple le monde des cubes avec une pince unique. LCGP va pouvoir relâcher certaines des contraintes entre actions trouvées par Graphplan qui deviendront alors des contraintes d'autorisation. Par exemple, pour Graphplan, charger un objet dans un avion et faire voler cet avion d'un aéroport *A* vers un aéroport *B* ne sont pas des actions indépendantes ; elles s'excluent donc mutuellement. En effet, une des préconditions pour charger l'objet est que l'avion soit dans l'aéroport de départ ; or ceci est détruit par l'action de faire voler l'avion. Dans Graphplan, ces deux actions ne peuvent donc pas faire partie d'un ensemble d'actions indépendantes. Par contre pour LCGP, elles sont autorisées et peuvent apparaître au sein d'un même ensemble autorisé : le chargement de l'objet ne détruit pas de précondition du vol de l'avion, et l'action de voler ne détruit pas de fait établi par le chargement.

La Table 1 donne les résultats obtenus pour Graphplan et LCGP sur 30 problèmes testés. En ce qui concerne le temps de calcul :

- 18 problèmes sont résolus par Graphplan en un temps qui est, en moyenne, de 1 heure et 26 minutes contre 4,80 secondes pour LCGP. Ce dernier est donc environ 1000 fois plus rapide.
- 12 problèmes ne sont pas résolus par Graphplan : 2 par manque de mémoire vive, et 10 après expiration d'un délai de 24 heures de calcul. Ils sont par contre tous résolus par LCGP en un temps moyen de 86 secondes. Ces problèmes sont donc plus difficiles que les autres pour les deux planificateurs, mais malgré cela, LCGP les résout tous.
- Dans tous les problèmes, LCGP est plus rapide que Graphplan.

En ce qui concerne les caractéristiques du graphe de planification et du plan-solution :

	Temps CPU (sec.)			Rapport tCPU.GP/ tCPU.LCGP	Actions		Niveaux du plan		
							GP	LCGP	
Problèmes	**STAN 3.0**	**GP**	**LCGP**		**GP**	**LCGP**		**(+)**	**(++)**
logistics.easy	,05	,67	,57	1,18	25	25	9	9	6
rocket_ext.a		14,78	,53	27,89	30	26	7	7	4
rocket_ext.b	24,34	4,19	,48	8,73	26	26	7	7	4
logistics.a	4,34	143,82	1,80	79,90	54	54	11	11	7
logistics.b	5,67	76 402,09	2,81	27 189,36	45	45	13	13	8
logistics.c	6 135,85	≥,86 400	250,54	≥,344,86		53	≥,11	13	8
logistics.d	22 105,24	plus de mém.	6,31			73		15	9
logistics.d3		≥,86 400	6,53	≥,13 231,24		72	≥,13	13	8
logistics.d1		≥,86 400	24,33	≥,3 551,17		68	≥,14	17	10
log010	,59	22,73	5,50	4,13	42	41	10	11	7
log011	218,03	5 946,83	3,38	1 759,42	48	49	11	12	7
log012	,77	6,37	2,08	3,06	38	38	8	8	5
log013	523,24	3 785,04	5,80	652,59	67	66	11	11	7
log014	1,17	8,95	6,96	1,29	71	75	10	11	7
log015		≥,86 400	6,84	≥,12 631,58		63	≥,11	12	7
log016		plus de mém.	6,05			40		16	9
log017		≥,86 400	120,25	≥,718,50		44	≥,16	17	10
log018	3,04	40,86	11,27	3,63	48	49	11	11	7
log019	5,77	15,90	4,57	3,48	46	48	11	12	7
log020		≥,86 400	509,95	≥,169,43		87	≥,14	15	9
log021	3 259,27	3 102,36	6,64	467,22	63	64	11	12	7
log022		≥,86 400	6,78	≥,12 743,36		74	≥,14	15	9
log023	232,42	≥,86 400	68,12	≥,1 268,35		61	≥,13	13	8
log024	286,80	3 445,45	6,58	523,62	64	67	12	13	8
log025	1,46	15,28	5,90	2,59	58	56	12	13	8
log026	,64	8,01	5,64	1,42	51	51	12	12	8
log027	23,15	≥,86 400	5,96	≥,14 496,64		72	≥,13	14	8
log028		≥,86 400	16,42	≥,5 261,88		77	≥,14	15	9
log029	1,19	13,37	10,22	1,31	46	46	10	11	7
log030	1,11	47,14	5,60	8,42	52	51	13	13	8
Moyenne 1 (*)	1 563,53	5 167,99	4,80	1 077,54	48,56	48,72	10,50	10,94	6,78
Moyenne 2 (**)	1 563,53	34 179,42	37,15	920,11	48,56	55,37	11,50	12,40	7,53

(+) nombre de niveaux du plan-solution après transformation par l'algorithme du § 6.3 ("à la Graphplan").
(++) nombre de niveaux du plan-solution avant transformation par l'algorithme du § 6.3.
(*) moyenne sur les problèmes résolus (cases blanches). Pour LCGP : moyenne sur les problèmes résolus par Graphplan.
(**) moyenne sur les 30 problèmes. Pour un échec, le temps pris en compte est 24h (86400 secondes).
Case gris clair : échec pour la résolution du problème correspondant.

Table 1 : Résultats dans le domaine Logistics

- Pour les 18 problèmes uniquement résolus par Graphplan, LCGP trouve une solution en moins d'étapes que Graphplan (avant transformation du plan par l'algorithme du § 6.3). La réduction de la taille (en nombre de niveaux) de l'espace de recherche est l'une des raisons du gain de performances.
- Après transformation du plan-solution par LCGP (cf. § 6.3), l'optimalité en nombre de niveaux est perdue : dans 8 des 18 problèmes résolus par Graphplan, LCGP trouve 1 niveau de plus. La différence reste cependant minime. Cette perte d'optimalité est la conséquence d'une autre des raisons de l'efficacité de LCGP : le graphe de planification dans lequel LCGP trouve un plan-solution contient plus de plans potentiels que celui dans lequel Graphplan trouve sa propre solution.
- Cette perte d'optimalité en nombre de niveaux ne se traduit pas forcément par un nombre d'actions plus important : LCGP trouve moins d'actions que Graphplan dans 2 des 8 problèmes non optimaux (cf. ci-dessus) et autant pour 1 problème. Sur les 10 problèmes restants (qui sont optimaux en nombre de niveaux pour LCGP), ce dernier trouve moins d'actions que Graphplan dans 3 cas et autant dans 6 autres cas.

Pour conclure sur l'étude de ce domaine, on peut dire qu'il se prête particulièrement bien à l'utilisation de LCGP. La perte d'optimalité (en nombre de niveaux) est très largement compensée par le gain de temps. De plus, cette perte demeure minime et n'est pas corrélée avec l'optimalité en nombre d'actions. Comme le nombre d'étapes d'un plan n'est pas forcément lié à un gain de temps lors de son exécution (un plan-solution de n niveaux peut très bien, suivant la durée et l'ordre des actions, être exécuté plus rapidement qu'un plan-solution possédant moins de niveaux) et que cet aspect temporel n'est abordé ni dans Graphplan, ni dans LCGP, aucune conclusion ne peut être tirée à propos des avantages de l'optimalité[1].

7.2 Le domaine du Ferry

Ce domaine permet également d'exprimer des problèmes de transport, mais beaucoup plus simples que ceux du domaine Logistics. La version classique est très limitée : deux rives d'une voie d'eau que l'on peut traverser avec un ferry. Celui-ci doit faire passer toutes les autos de la première rive à la deuxième. Une différence essentielle par rapport au domaine Logistics est le fait que le ferry ne peut transporter qu'une auto à la fois ; les plans-solutions sont donc forcément linéaires et la difficulté des problèmes augmente très rapidement avec le nombre d'autos à cause d'une très forte combinatoire.

Un planificateur "général" comme Graphplan ne permet pas de prendre en compte certaines symétries du problème. L'ordre de traversée des voitures importe peu pourvu qu'elles passent toutes mais Graphplan travaille comme si cet ordre était important. Ce défaut peut être amélioré par un module de traitement de la symétrie statique des domaines (symétries entre l'état initial et le but) comme dans le planificateur STAN [FL99]. Ceci permet par exemple, lorsque la traversée en premier de la voiture 1 ne donne aucun résultat, de ne pas essayer de faire traverser en premier les autres autos, puisqu'elles sont toutes semblables pour ce problème.

Contrairement au domaine précédent, la Table 2 montre que LCGP n'est pas toujours le plus performant :

- De 1 à 5 autos, LCGP est plus rapide que Graphplan mais semble perdre son avantage avec l'augmentation du nombre d'autos : pour 1 et 2 autos il est 2 fois plus rapide ; 1,5 fois plus rapide pour 3 autos ; puis 1,67 fois pour 4 autos et enfin 1,36 fois pour 5 autos... Ces problèmes sont cependant résolus rapidement par les deux planificateurs et les différences ne doivent pas être interprétées de manière trop catégorique.
- Cette décroissance se confirme pour 6 autos : Graphplan est légèrement plus rapide que LCGP, malgré un nombre plus important de backtracks. Ceci vient du fait que ce dernier rajoute un test de vérification des ensembles autorisés qui est assez coûteux (cf. § 6.2).
- Les résultats s'inversent ensuite brusquement à partir de 7 autos : LCGP est 1,95 fois plus rapide que Graphplan, et 6,21 fois plus rapide pour 8 autos. On peut conjecturer que cette tendance se confirmerait (bien que nous n'ayons pas pu le vérifier). On peut en effet remarquer que le nombre de backtracks pour

Nb autos	Temps CPU (sec.)		Rapport tCPU.GP/ tCPU.LCGP	Backtracks		Actions		Niveaux du plan		
								GP	LCGP	
	GP	LCGP		GP	LCGP	GP	LCGP		(+)	(++)
1	,02	,01	2,00	6	3	3	3	3	3	2
2	,04	,02	2,00	43	13	7	7	7	7	4
3	,06	,04	1,50	626	81	11	11	11	11	6
4	,15	,09	1,67	4 052	968	15	15	15	15	8
5	,53	,39	1,36	20 931	8 482	19	19	19	19	10
6	3,31	3,35	,99	125 195	74 667	23	23	23	23	12
7	62,58	32,07	1,95	2 182 083	664 636	27	27	27	27	14
8	3 456,40	556,86	6,21	97 030 122	9 951 906	31	31	31	31	16

(+) nombre de niveaux du plan-solution après transformation par l'algorithme du § 6.3 ("à la Graphplan")
(++) nombre de niveaux du plan-solution avant transformation par l'algorithme du § 6.3

Table 2 : Résultats dans le domaine du Ferry

[1] LCGP, avant la transformation du plan-solution, reste optimal en nombre de niveaux au sens de Res*.

Graphplan augmente beaucoup plus vite que pour LCGP : pour 6 autos, Graphplan fait 1,7 fois plus de backtracks que LCGP, pour 7 autos il en fait 3,3 fois plus et pour 8 autos il en fait presque 10 fois plus. LCGP perd alors moins de temps à vérifier l'autorisation que Graphplan n'en perd à effectuer des backtracks.

Dans ce domaine, LCGP n'est donc pas toujours le plus rapide. Ceci vient en partie du temps nécessaire au calcul de l'autorisation des ensembles d'actions. Graphplan est plus performant sur un problème qui reste facile : environ 3 secondes et aux alentours de 100.000 backtracks. La difficulté à résoudre les problèmes augmente ensuite plus vite pour Graphplan que pour LCGP. La perte de temps de LCGP sur des problèmes faciles est négligeable par rapport à ce qui est gagné sur des problèmes plus difficiles.

8 Conclusion

Aucune des améliorations précédemment apportées à Graphplan n'a jamais modifié la structure du graphe de planification, si ce n'est pour augmenter l'expressivité du langage de description (effets conditionnels...) ou prendre en compte l'incertain. Dans Graphplan, la structure de ce graphe est basée sur le concept d'indépendance entre actions qui permet de générer des plans-solutions comportant des actions exécutables en parallèle. Nous avons montré ici que cette condition d'indépendance pouvait être avantageusement remplacée par une condition moins restrictive d'autorisation entre actions. L'espace de recherche qui est alors développé par l'algorithme LCGP que nous avons implémenté devient plus compact (moins de niveaux que Graphplan, chacun étant plus dense) et entraîne dans certains domaines une telle amélioration des performances que la perte de la propriété d'optimalité qui en résulte (toute relative par ailleurs, cf. § 7.1) nous semble négligeable. Le nombre d'actions des plans ne semble pas être lié à l'optimalité en nombre d'étapes, si bien que LCGP retourne parfois des plans-solutions parallèles comportant moins d'actions que Graphplan.

Références

[BAC98] Bäckström C., *Computational aspects of reordering plans*. JAIR 9, pp. 99-137, 1998.

[BF95] Blum A., Furst M., *Fast planning through planning graphs analysis*. Actes de IJCAI, pp. 1636-1642, 1995.

[BF97] Blum A., Furst M., *Fast planning through planning graphs analysis*. Artificial Intelligence 90, pp. 281-300, 1997.

[BK96] Bacchus F., Kabanza F., *Using temporal logic to control search in a forward chaining planner*. Actes de EWSP, pp. 141-153, 1996.

[BW94] Barrett A., Weld D., P*artial-order planning: evaluating possible efficiency gains*. Artificial Intelligence 67, pp. 71-112, 1994.

[FAR95] Farreny H., *Recherche heuristiquement ordonnée dans les graphes d'états*. Editions MASSON, 1995.

[FL98] Fox M., Long D., *The automatic inference of state invariants in TIM*. JAIR 9, pp. 367-421, 1998.

[FL99] Fox M., Long D., *The detection and exploitation of symmetry in planning problems*. Actes de IJCAI, pp. 956-961, 1999.

[GA99] Guéré E., Alami R., *A possibilistic planner that deals with non-determinism and contingency*. Actes de IJCAI, pp. 996-1001, 1999.

[KAM97] Kambhampati S., *Refinement planning as a unifying framework for plan synthesis*. AI Magazine, pp. 67-97, 1997.

[KAM99] Kambhampati S., *Improving Graphplan's search with EBL & DDB techniques*. Actes de IJCAI, pp. 982-987, 1999.

[KNHD97] Koehler J., Nebel B., Hoffmann J., Dimopoulos Y., *Extending planning graphs to an ADL subset*. Actes de ECP, pp. 273-285, 1997.

[KOE98] Koehler J., *Planning under resources constraints*. Actes de ECAI, pp. 489-493, 1998.

[KS99] Kautz H., Selman B., *Unifying SAT-based and Graph-based Planning*. Actes de IJCAI, pp. 318-325, 1999.

[LF99] Long D., Fox M., *The efficient implementation of the plan-graph in STAN*. JAIR 10, pp. 87-115, 1999.

[MBD94] Minton S., Bresina J., Drummond M., *Total order and partial order planning: a comparative analysis*. Artificial Intelligence, pp. 71-111, 1994.

[NDK97] Nebel B., Dimopoulos Y., Koehler J., *Ignoring irrelevant facts and operators in plan generation*. Actes de ECP, pp. 338-350, 1997.

[RF91] Régnier P., Fade B., *Complete determination of parallel actions and temporal optimization in linear plans of actions*. Actes de EWSP, pp. 100-111, 1991.

[VB94] Veloso M., Blythe J., *Linkability: examining causal link commitments in partial-order planning*. Actes de AIPS, pp. 170-175, 1994.

[VCR99] Vidal V., Cayrol M., Régnier P., *Contribution à l'amélioration de Graphplan : formalisation, critique, modification : LCGP*. Rapport IRIT 99/12-R, 1999.

[VR99] Vidal V., Régnier P., *Total order planning is better than we thought*. Actes de AAAI, 1999.

[WAS98] Weld D., Anderson C., Smith D., *Extending Graphplan to handle uncertainty and sensing actions*. Actes de AAAI, pp. 897-904, 1998.

[WEL94] Weld D., A*n introduction to least commitment planning*. AI Magazine, pp. 27-61, 1994.

[ZK99] Zimmerman T., Kambhampati S., *Exploiting symmetry in the planning-graph via explanation-guided search*. Actes de AAAI, 1999.