New Results about LCGP, a Least Committed GraphPlan

Michel Cayrol **Pierre Régnier** **Vincent Vidal**

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

Abstract

Planners from the family of Graphplan (Graphplan, IPP, STAN...) 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 independence, Graphplan constrains 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 fewer 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, LCGP "practically" solves more problems than planners from the family of Graphplan (Graphplan, IPP, STAN...). In most cases, these tests demonstrate the best performances of LCGP. Then, we present a domain-independent heuristic for variable and domain ordering. LCGP is thus improved using this heuristic, and compared with HSP-R, a very efficient non-optimal sequential planner, based on an heuristic backward state space search.

1 Introduction

Since some years, the development of a new family of planning systems based on the planner Graphplan (Blum and Furst 1995) leads to numerous evolutions in planning. Graphplan develops, level after level, a compact search space called a planning-graph. During this construction stage, it does not use all the informations (exclusion relations among state variables or actions) that are progressively taken into account in the other planning techniques (state space search, search in the space of partial plans). These constraints are only computed and memoized at each level as mutual exclusions in the way of CSP (Kambhampati 1999a, 1999b). The search space is easier to develop but, on the other side, its achievement does not coincide with the obtaining of a solution. A second stage (extraction stage) is necessary to try to extract a valid plan from the planning-graph and the sets of mutual exclusions.

Several techniques have been employed to improve Graphplan: reduction of the search space before the extraction stage (Fox and Long 1998; Nebel, Dimopoulos, and Koehler 1997), improvement of the domain and problem representation language (Koehler et al. 1997; Weld, Anderson, and Smith 1998; Guéré and Alami 1999; Koehler 1998), improvement of the extraction stage (Kambhampati 1999a, 1999b; Kautz and Selman 1999; Long and Fox 1999; Fox and Long 1999; Zimmerman and Kambhampati 1999).

2 Summary of our method

In all these works, the structure of the generated plans remains the same whatever the construction graph method is. A plan can indeed be represented as a minimal sequence of sets of simultaneous actions: each step of the algorithm produces a level of the planning-graph, each level being connected to a set of simultaneous actions.

In a plan of Graphplan (minimal sequence of sets of simultaneous actions), the computation of the final situation E_f – produced by the simultaneous application of the actions of a set Q from an initial situation E_i – is independent of the order employed to apply these actions. The final situation remains the same whatever the order of these actions in the sequence is, provided that Q checks a property I (Independence). This property is easy to test and I(Q) denotes that Q verifies the property I. E_f is directly computed from the initial situation E_i and from the simultaneous application of the actions of the set Q: $E_f = g(E_i, Q)$.

We have established another property A (Authorization) which is less restrictive than I and easier to verify ($\mathrm{I}(Q) \Rightarrow \mathrm{A}(Q)$). This property guarantees the existence of at least one serialization S of the actions of Q (but does not require its computation). The application of this sequence to E_i can be identically computed ($E_f = g(E_i, Q)$) but E_f can no more be considered as the result of the *simultaneous* application of the actions of Q.

Then, we have developed a Graphplan-like planner called LCGP (Least Committed Graphplan, see (Cayrol, Régnier and Vidal 2000)) which works in the same way: it incrementally constructs a stratified graph, then searches it to extract a plan. The graph that Graphplan would have built is a subgraph of the one of LCGP (cf. example § 6.4). So, goals generally appear sooner (at the same time in the worst cases). LCGP transforms then the produced plan into a Graphplan-like plan. The earlier obtaining of a solution is done to the prejudice of the optimality of these plans in the sense of Graphplan (number of levels). In practice, in classical benchmarks (Logistics, Ferry...), LCGP rapidly gives a solution when Graphplan is unable to produce a plan after a significant running time.

At first, we will formalize the structure of the plans of Graphplan (cf. § 3). Then, we will suggest that Graphplan, using the Independence criterion, over-constrains the choice of the actions into the sets of simultaneous actions (cf. § 4). We will demonstrate that we can relax this criterion by changing the structure of the produced plans. For lack of space, cf. (Vidal 2000) for details of demonstrations and algorithms.

3 Semantics and formalization of the plans of Graphplan

The most important element of a plan is the *action*, which is an instance of an *operator*. In Graphplan, operators are Strips-like operators, without negation in their preconditions. We use a first order logic language L, constructed from the vocabularies Vx, Vc, Vp that respectively denote finite disjoint sets of symbols of variables, constants and predicates. We do not use symbols of functions.

Definition 1: An *operator*, denoted by o, is a triple $\langle pr, ad, de\rangle$ where pr, ad and de denote finite sets of atomic formulas in the language L. Prec(o), Add(o) and Del(o) respectively denote the sets pr, ad and de of the operator o. O denotes the finite set of operators.

Definition 2: A *state* is a finite set of ground atomic formulas (i.e. without any symbol of variable). A ground atomic formula is also called a *proposition*. P denotes the set of all the propositions that can be constructed with the language L.

Definition 3: An *action* denoted by a is a ground instance $o\theta = \langle pr\theta, ad\theta, de\theta\rangle$ of an operator o which is obtained by applying a substitution θ defined with the language L such that $ad\theta$ and $de\theta$ are disjoint sets. Prec(a), Add(a), Del(a) respectively denote the sets $pr\theta$, $ad\theta$, $de\theta$ and represent the preconditions, adds and deletes of a.

Definition 4: A denotes the finite set of actions obtained by applying all the possible ground instanciations of the operators of O.

The main structure we are going to define, the *sequence of sets of actions*, will be used later to represent the plans of Graphplan and LCGP: it defines the order in which the sets of actions are considered from the point of view of the execution of the actions they contain.

Definition 5: A *sequence of sets of actions* is a finite and ordered series of finite sets of actions. A sequence of sets of actions S is noted $\langle Q_i\rangle_n$, with $n \in \mathbb{N}$ and $i \in [1, n]$. If $n = 0$, S is the empty sequence: $S = \langle Q_i\rangle_0 = \langle\rangle$; if $n > 0$, S can be noted $\langle Q_1, Q_2, ..., Q_n\rangle$. If the sets of actions are singletons (i.e. $Q_1 = \{a_1\}$, $Q_2 = \{a_2\}$, ..., $Q_n = \{a_n\}$), the associated sequence of sets of actions is called *sequence of actions* and will be (incorrectly) noted $\langle a_1, a_2, ..., a_n\rangle$. The set of finite sequences of finite sets of actions formed from the set of actions A is denoted by $(2^A)^*$. The set of sequences of actions formed using the set of actions A is denoted by A^*.

Definition 6: We define the following functions:

- first: $(2^A)^* - \{\langle\rangle\} \rightarrow 2^A$ is defined by: first($\langle Q_1, Q_2, ..., Q_n\rangle$) = Q_1.
- rest: $(2^A)^* - \{\langle\rangle\} \rightarrow (2^A)^*$ is defined by: rest($\langle Q_1, Q_2, ..., Q_n\rangle$) = $\langle Q_2, ..., Q_n\rangle$.

Definition 7: Let $S, S' \in (2^A)^*$ be two sequences of sets of actions with $S = \langle Q_i\rangle_n$ and $S' = \langle Q'_i\rangle_m$. The *concatenation* (noted ⊕) of S and S' is defined by:

$$S \oplus S' = \langle R_i\rangle_{n+m} \text{ with } R_i = (\text{if } i \le n \text{ then } Q_i \text{ else } Q'_i).$$

⊕ is an internal composition law in $(2^A)^*$.

Definition 8: A *linearization* of a set of actions $Q \in 2^A$ with $Q = \{a_1, ..., a_n\}$ is a permutation of Q, i.e. a sequence of actions S such as there is a bijection b: $[1, n] \rightarrow Q$ where $S = \langle b(1), ..., b(n)\rangle$. The linearization of the empty set {} is the empty sequence $\langle\rangle$. The set of all the linearizations of Q is denoted by Lin(Q).

Notations: If Q is the set of actions $Q = \{a_1, ..., a_n\}$, then:

- the union of the preconditions of the elements of Q is noted Prec(Q): Prec(Q) = Prec(a_1) ∪ ... ∪ Prec(a_n),
- the union of the adds of the elements of Q is noted Add(Q): Add(Q) = Add (a_1) ∪ ... ∪ Add(a_n),
- the union of the deletes of the elements of Q is noted Del(Q): Del(Q) = Del(a_1) ∪ ... ∪ Del(a_n).

This is also available if Q is the sequence of actions $Q = \langle a_1, ..., a_n\rangle$.
If $Q = \varnothing$ or $Q = \langle\rangle$, Prec(Q) = Add(Q) = Del(Q) = ∅.

In a plan of Graphplan, a set of actions represents actions that can be executed in any order, and even in parallel, without changing the resulting state. In this paper, we will systematically use the expression *set of simultaneous actions* – and not *set of parallel actions* – to stress the fact that actions belong to the same set (that represents a same level of the planning-graph) without fixing in advance their future order of execution. The expression *set of parallel actions* will be reserved for a set of actions that can be executed in parallel.

As the majority of partial order planners (UCPOP, SNLP...), Graphplan strongly constrains the choice of actions so as to obtain the same resulting state with a parallel or sequential execution of a plan.

To achieve this result using a Strips description of actions, every action in a set must be independent with the others, i.e. their effects must not be contradictory (not any action must delete an add effect of another one) and they must not interact (not any action must delete a precondition of another one).

Definition 9: Two actions $a_1 \neq a_2 \in A$ are *independent* iff

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

An *independent set* represents a set of parallel actions: the actions of this set are independent pairwise.

Definition 10: A set of actions $Q \in 2^A$ is an *independent set* iff the actions of this set are independent pairwise, i.e.:

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

Let us notice that for two actions to be executable in parallel, another condition must be true: they must not have incompatible preconditions. Graphplan and LCGP detect and take advantage of these incompatibilities.

A sequence $\langle Q_1, ..., Q_n\rangle$ of sets of simultaneous actions partially defines the order of execution of the actions. The end of the execution of each action in Q_i must precede the beginning of the execution of each action in Q_{i+1}. This implicates that the execution of all the actions in Q_i precedes the execution of all the actions in Q_{i+1}.

Let us formalize a plan of Graphplan, by defining an application to simulate the execution of a sequence of sets of actions from an initial representation of the world. If a sequence of sets of actions cannot be applied to a state, the result will be $\perp$, the impossible state.

Definition 11: Let $\Re$: $(2^P \cup \{\perp\}) \times (2^A)^* \rightarrow (2^P \cup \{\perp\})$, defined as:
$E \Re S =$
If $S = \langle\rangle$ or $E = \perp$
then E
else If first(S) is **independent** and Prec(first(S)) $\subseteq E$
then $[(E - \mathrm{Del}(\mathrm{first}(S))) \cup \mathrm{Add}(\mathrm{first}(S))] \Re\ \mathrm{rest}(S)$
else $\perp$.

Definition 12: A sequence of sets of actions $S \in (2^A)^*$ is a *plan* for a state $E \in (2^P \cup \{\perp\})$, in relation to $\Re$, iff $E \Re S \neq \perp$.

When $E \Re S \neq \perp$, we can associate a semantics to S which is connected with the execution of actions in the real world, because we are sure (in a static world) that our prediction of the final state is correct.

Notation: when there is no ambiguity, $((...((E \Re S_1) \Re S_2) \Re ...)\Re S_n)$ will be noted $E \Re S_1 \Re S_2 \Re ... \Re S_n$.

The successive application of $\Re$ to two sequences of sets of actions and to a state gives the same result than the application of the concatenation of these two sequences to the same state.

Property 1: Let $E \in (2^P \cup \{\perp\})$ be a state and $S_1, S_2 \in (2^A)^*$ two sequences of sets of actions. Then:

$$E \Re (S_1 \oplus S_2) = E \Re S_1 \Re S_2.$$

The Theorem 1 establishes the essential property of Graphplan: *the actions of a plan of Graphplan that can be executed in parallel give the same result when they are executed sequentially, whatever the order of execution is.*

Theorem 1: Let $E \in (2^P \cup \{\perp\})$ be a state and $S \in (2^A)^* - \{\langle\rangle\}$ a sequence of sets of actions, with $S = \langle Q_1, ..., Q_n\rangle$. Then:

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

The proof of this theorem is based on the folloowing three lemmas.

Lemma 1: Let $A, A_1, ..., A_n, B_1, ..., B_n$ be sets such as $\forall\ i \in [1, n-1]$, $A_{i+1} \cap (B_1 \cup ... \cup B_i) = \varnothing$. Then:

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

The following lemma will be used to calculate the application of a sequence of actions to a state (different from $\perp$) when it contains all the preconditions of every action of the sequence and when an action never deletes the preconditions of another one which succeeds to it (immediately or not). In this particular case, the result is always different from $\perp$.

Lemma 2: Let $E \in 2^P$ be a state and $S \in A^*$ a sequence of actions, with $S = \langle a_i\rangle_n$, such as: Prec(S) $\subseteq E$ and $\forall\ i \in [1, n-1]$, $\mathrm{Prec}(a_{i+1}) \cap (\mathrm{Del}(a_1) \cup ... \cup \mathrm{Del}(a_i)) = \varnothing$. Then:

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

Lemma 3: Let $E \in (2^P \cup \{\perp\})$ be a state and $Q \in 2^A$ a set of actions. Then:

$$E \Re \langle Q\rangle \neq \perp \Rightarrow \forall\ S \in \mathrm{Lin}(Q),\ E \Re \langle Q\rangle = E \Re S.$$

Now, we are going to question this property. We can remark that $E \Re \langle\{a_1, ..., a_n\}\rangle = E \Re \langle a_1, ..., a_n\rangle$ when $\forall\ i \in [1, n-1]$, $\mathrm{Del}(a_{i+1}) \cap (\mathrm{Add}(a_1) \cup ... \cup \mathrm{Add}(a_i)) = \varnothing$ and $\forall\ i \in [1, n-1]$, $\mathrm{Prec}(a_{i+1}) \cap (\mathrm{Del}(a_1) \cup ... \cup \mathrm{Del}(a_i)) = \varnothing$. In this case, we can see that $E \Re \langle a_1, ..., a_n\rangle$ can be computed without knowing the order of the actions of the sequence $\langle a_1, ..., a_n\rangle$.

4 Towards a new structure for plans

Graphplan imposes very strong conditions on the plans using the independence property to choose the actions in the sets of simultaneous actions. So, if necessary, it is always possible to execute these actions in parallel. Now, we are going to demonstrate that we can modify this property to relax a part of the constraints on simultaneous actions and nevertheless produce plans.

When we do this modification, we can no more be sure that the actions in a set of actions (actions at a same level) can be executed in parallel because they are possibly not independent. The main idea of Graphplan is preserved because these new sets of actions are used "in one piece": we always try to establish all the preconditions of all the actions in a set using the effects of the actions that belong to another set of actions (at the precedent level).

When we relax a part of the constraints on independent actions, we define a more flexible relation (no more symmetrical) between the actions: the *authorization relation*. An action a_1 authorizes an action a_2 if a_2 can be executed at the same time or after a_1. To achieve this result it is sufficient to preserve two conditions among conditions for independent actions: a_1 must not delete a precondition of a_2 (a_2 must be executable) and a_2 must not delete a fact added by a_1. This definition implies an order for the execution of two actions: a_1 authorizes a_2 means that if a_1 is executed before a_2, the add effects of a_1 will be preserved executing a_2 and the preconditions of a_2 will be preserved executing a_1. On the other hand if a_1 does not authorize a_2 and if we execute a_1 before a_2, either a_2 deletes an add effect of a_1 (so the resulting state cannot be computed by applying simultaneously a_1 and a_2), or a precondition of a_2 is deleted by a_1 (so we cannot execute a_2).

Definition 13: An action $a_1 \in A$ *authorizes* an action $a_2 \in A$ (noted $a_1 \angle a_2$) iff (1) $a_1 \neq a_2$ and (2) $\text{Add}(a_1) \cap Del(a_2) = \varnothing$ and $\text{Prec}(a_2) \cap \text{Del}(a_1) = \varnothing$. An action a_1 *forbids* an action a_2 iff the action a_1 does not authorize a_2, i.e. if not($a_1 \angle a_2$). Generally, the authorization is not an order relation.

This authorization relation leads us to a new definition of the sets that can belong to a plan. These sets will not be independent sets. We want for every set of actions to find at least one linearization of it that could be a plan. Such a linearization introduces a notion of order among actions.

Definition 14: A sequence of actions $\langle a_i \rangle_n \in 2^A$ is *authorized* iff $\forall\, i, j \in [1, n], i < j \Rightarrow a_i \angle a_j$, i.e.:

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

Definition 15: A set of actions $Q \in (2^A)^*$ is *authorized* (if not it is *forbidden*) iff one can find an authorized linearization $S \in \text{Lin}(Q)$. We will note $\text{LinA}(Q)$ the set of all the authorized linearizations of Q:

$$\text{LinA}(Q) = \{S \in \text{Lin}(Q) \mid S \text{ is an authorized linearization}\}.$$

So, a set of actions is authorized if one can find an order among the actions of the set such as no action in the set deletes either an add effect of a preceding action or a precondition of a following action.

Let us define $\mathfrak{R}^*$, a new application of a sequence of sets of actions to a state that uses the authorization relation between actions. Our planner LCGP will be based on $\mathfrak{R}^*$. With this definition, we can demonstrate a new theorem to compute the resulting state (Theorem 2). This theorem does not use all the linearizations of the independent sets of actions but only the linearizations that respect the authorization constraints among actions of the sets (authorized linearizations).

Definition 16: Let $\mathfrak{R}^*: (2^P \cup \{\perp\}) \times (2^A)^* \to (2^P \cup \{\perp\})$, defined as:

$E \,\mathfrak{R}^*\, S =$
 If $S = \langle\rangle$ or $E = \perp$
 then E
 else If first(S) is **authorized** and Prec(first(S)) $\subseteq E$
  then $[(E - \text{Del}(\text{first}(S))) \cup \text{Add}(\text{first}(S))] \,\mathfrak{R}^*\, \text{rest}(S)$
  else $\perp$.

Definition 17: A sequence of sets of actions $S \in (2^A)^*$ is a *plan* for a state $E \in (2^P \cup \{\perp\})$, in relation to $\mathfrak{R}^*$, iff $E \,\mathfrak{R}^*\, S \neq \perp$.

When $E \,\mathfrak{R}^*\, S \neq \perp$, we can associate a semantics to S (different from the semantics related to plans that are recognized using $\mathfrak{R}$). This semantics is connected with the execution of actions in the real world because we are sure (in a static world) that our prediction of the final state is correct.

The Property 1, Lemma 1 and Lemma 2 remain true replacing $\mathfrak{R}$ by $\mathfrak{R}^*$. The Theorem 2 we achieve is close to Theorem 1 (its proof is alike): the application of every authorized linearization of sets of actions of a plan of $\mathfrak{R}^*$ always gives the same result.

Theorem 2: Let $E \in (2^P \cup \{\perp\})$ be a state and $S \in (2^A)^* - \{\langle\rangle\}$ a sequence of sets of actions, with $S = \langle Q_1, \dots, Q_n \rangle$. Then:

$$E \,\mathfrak{R}^*\, S \neq \perp \Rightarrow \forall\, S_1 \in \text{LinA}(Q_1), \dots, \forall\, S_n \in \text{LinA}(Q_n),$$
$$E \,\mathfrak{R}^*\, S = E \,\mathfrak{R}^*\, (S_1 \oplus \dots \oplus S_n).$$

5 Relations between the formalisms

The independence and authorization relations are strongly related, so the two formalisms are connected and a plan for $\mathfrak{R}$ is a plan for $\mathfrak{R}^*$:

Theorem 3: Let $E \in (2^P \cup \{\perp\})$ be a state and $S \in (2^A)^*$ a sequence of sets of actions. Then:

$$E \,\mathfrak{R}\, S \neq \perp \Rightarrow E \,\mathfrak{R}^*\, S = E \,\mathfrak{R}\, S.$$

We can also demonstrate that if a sequence of sets of actions S is not a plan for a situation E in relation to $\mathfrak{R}^*$, it is neither a plan for E in relation to $\mathfrak{R}$:

$$E \,\mathfrak{R}^*\, S = \perp \Rightarrow E \,\mathfrak{R}\, S = \perp.$$

There is another connection between the plans recognized by $\mathfrak{R}$ and the plans recognized by $\mathfrak{R}^*$: all the plans constructed using the authorized linearizations of the sets of actions of a plan recognized by $\mathfrak{R}^*$, are recognized by $\mathfrak{R}$. Moreover, the application of $\mathfrak{R}^*$ on the original plan produces the same resulting state than the application of $\mathfrak{R}$ on every plan constructed using the authorized linearizations of the sets of actions of the plan.

Theorem 4: Let $E \in (2^P \cup \{\perp\})$ be a state and $S \in (2^A)^* - \{\langle\rangle\}$ a sequence of sets of actions, with $S = \langle Q_1, \dots Q_n \rangle$. Then:

$$E \,\mathfrak{R}^*\, S \neq \perp \Rightarrow \forall\, S_1 \in \text{LinA}(Q_1), \dots, \forall\, S_n \in \text{LinA}(Q_n),$$
$$E \,\mathfrak{R}^*\, S = E \,\mathfrak{R}\, (S_1 \oplus \dots \oplus S_n).$$

This theorem is essential and gives meaning to the plans recognized by $\mathfrak{R}^*$: an elementary transformation (the search of an authorized linearization of every set of actions) produces a plan recognized by $\mathfrak{R}$ (and that Graphplan would have produced).

6 Integration of this new structure of plans in Graphplan

Now, we are going to explain the modifications we have done on Graphplan to implement this new formalism in LCGP. To sum up, one can remember that a planning-graph is a graph constituted by successive levels, each one is marked with a positive integer and is constituted by a set of actions and a set of propositions. The level 0 is an exception and only contains propositions representing facts of the initial state.

6.1 Extending the planning-graph

During this stage, the only difference between Graphplan and LCGP is about the computation of the exclusion relation between actions. In Graphplan, two actions a_1 and a_2 are mutually exclusive iff (1) they are different and (2) they are not independent (i.e. one of them forbids the other: not($a_1 \angle a_2$) **or** not($a_2 \angle a_1$)), or if a precondition of one is mutually exclusive with a precondition of the other. In LCGP, the exclusion relation between actions is thus defined:

Definition 18: Two actions a_1, $a_2 \in A$ are *mutually exclusive* iff (1) $a_1 \neq a_2$ and (2) each of them forbids the other: not($a_1 \angle a_2$) **and** not($a_2 \angle a_1$), or if a precondition of one is in mutual exclusion with a precondition of the other.

This new definition of the mutual exclusion (**or** in Graphplan, **and** in LCGP), implies that LCGP finds fewer mutually exclusive pairs of actions than Graphplan (the same number in the worst cases). Consequently, a level n of LCGP will include more actions and propositions than a level n of Graphplan (cf. example of § 6.4) because actions can sometimes be applied earlier in LCGP (given a level n, the graph of Graphplan is a subgraph of the one of LCGP). The graph of LCGP grows faster and contains, for a same number of levels, more potential plans than the graph of Graphplan (the same number in the worst cases). The extension of the graph finishes earlier too because the goals generally appear before being produced by Graphplan (at the same level in the worst cases).

6.2 Searching for a plan

After the construction stage, Graphplan tries to extract a solution from the planning-graph, using a level-by-level approach. It begins with the set of propositions constructed at the last level (that includes the goals) and inspects the different sets of actions that assert the goals. It chooses one of them (backtrack point) and searches again, at the previous level, for the sets of actions that assert the preconditions of these actions... At each level, the actions of the chosen set must be independent two by two and their preconditions must not be mutually exclusive to be in agreement with the associated semantics (parallel actions, cf. § 3). So, Graphplan tests, using the exclusion relations, that there is no pair of mutually exclusive actions.

In LCGP, mutual exclusions are not sufficient to preserve a set of actions for a plan. This set must also be authorized (cf. Definition 15), i.e. one must find a sequence of actions (authorized sequence) such as not any action deletes a precondition of a following action or an add effect of a previous action of the sequence. This condition (to check wether a set of actions is authorized) can be verified using a modified topological sort algorithm (polynomial) to test that the graph of the Definition 19 (for the considered set of actions), is a directed acyclic graph (DAG):

Definition 19: Let $Q \in 2^A$ be a set of actions, with $Q = \{a_1, ..., a_n\}$. The *authorization graph* AG(N, C) of Q is an oriented graph defined by:

- N is the set of the nodes such that for each action a_i there is only one associated node of N noted n(a_i): $N = \{\text{n}(a_1), ..., \text{n}(a_n)\}$,
- C is the set of arcs that represent the order constraints among actions: there is an arc from n(a_i) to n(a_j) iff the execution of a_i **must precede** the execution of a_j, i.e. if a_j forbids a_i:

$$\forall\ a_i \neq a_j \in Q,\ (\text{n}(a_i), \text{n}(a_j)) \in C \Leftrightarrow \text{not}(a_j \angle a_i).$$

Indeed, we can demonstrate that:

Theorem 5: Let $Q \in 2^A$ be a set of actions and AG(N, C) the authorization graph of Q. Then:

$$\text{AG has no cycle} \Leftrightarrow Q \text{ is authorized.}$$

6.3 Return of the plan

The plan that LCGP returns is not recognized by $\mathfrak{R}$ (which recognizes plans of Graphplan) but by $\mathfrak{R}^*$. A plan of LCGP can be transformed into a plan recognized by Graphplan, by using a modified version of the polynomial algorithm of (Régnier and Fade 1991), revised and formalized by (Bäckström 1998, p.119) who demonstrates that it finds the optimal reordering in number of levels of the plan (i.e. in number of sets of independent actions).
As for the search of an authorized sequence of a set of actions, this stage will be decomposed in two parts. At first, we build a graph that represents the constraints of the plan (i.e. order relations and independence relations among actions), and then we use a modified topological sort algorithm on this graph to find the sequence of sets of actions corresponding to the plan-solution.

Definition 20: Let $E \in 2^P$ be a state and $S \in (2^A)^*$ a sequence of sets of actions, with $S = \langle Q_1, ..., Q_n\rangle$, such as $E\ \mathfrak{R}^*\ S \neq \perp$. The *partial order graph* POG(N, C) of S is an oriented graph defined by:

- N is the set of the nodes such that for each action $a \in Q_i$, $\forall\ i \in [1, n]$, there is only one associated node of N noted n(a),
- C is the set of arcs that represent the constraints among actions: there is an arc from n(a_i) to n(a_j) iff the execution of a_i **must precede** the execution of a_j, i.e.:

$(\text{n}(a_i), \text{n}(a_j)) \in C \Leftrightarrow$
$(a_i \neq a_j \in Q_k$ with $k \in [1, n]$ and not$(a_j \angle a_i))$
or
$(a_i \in Q_k$ and $a_j \in Q_p$ and $1 \leq k < p \leq n$ and
(not$(a_j \angle a_i)$ or not$(a_i \angle a_j)$ or Add$(a_i) \cap$ Prec$(a_j) \neq \emptyset$))

The only difference with the PRF algorithm in (Bäckström, 1998) is that we must take into account the fact that actions in a same set can be not independent (in that case, they are authorized because $E\ \mathfrak{R}^*\ S \neq \perp$). So, we must order these actions in the same way we do to check wether a set of actions is authorized (cf. § 6.2).

6.4 Example

The following example illustrates the difference between Graphplan and LCGP. The set of propositions is $P = \{a, b, c, d\}$ and the set of actions is $\{A, B, C\}$, with:

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

The initial state of the problem is $I = \{a\}$, and the goal is $G = \{d\}$. Figure 1, the planning-graph of Graphplan.

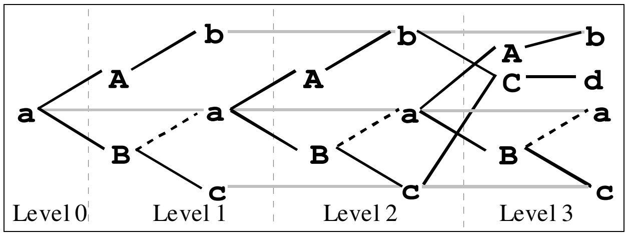

Figure 1: The planning-graph of Graphplan

The actions A and B are mutually exclusive, because B deletes a (precondition of A). At the level 1, the pairs of mutually exclusive propositions are $\{a, c\}$ and $\{b, c\}$. So, the action C cannot be used at the level 2 to produce the goal. At this level, b and c does not remain mutually exclusive, because the no−op of b and the action B are independent. The action C can be applied at the level 3. The produced plan is $\langle A, B, C\rangle$. Figure 2, the planning-graph of LCGP.

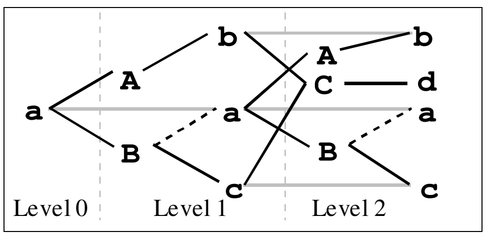

Figure 2: The planning-graph of LCGP

The main difference is that A and B are not mutually exclusive, because A authorizes B ($A \angle B$). Thus, at the level 1, the propositions b and c are not mutually exclusive, and the action C can be applied at the level 2. The produced plan is $\langle\{A, B\}, \{C\}\rangle$, that is not recognized by Res: $\{A, B\}$ is not an independent but an authorized set of actions. Using the § 6.3 algorithm, we obtain the same plan than Graphplan: $\langle A, B, C\rangle$.

7 Empirical evaluation

Here are the results of the tests we performed with our own implementation of Graphplan (we will call it GP). GP and LCGP share most of their code: differences between the two planners are minimal (cf. § 6). The common part includes well-known improvements of Graphplan: EBL/DDB techniques from (Kambhampati 1999a) and a graph construction inspired by (Long and Fox 1999). GP and LCGP are implemented in Allegro Common Lisp 5.0, and all the tests have been performed with a Pentium-II 450Mhz machine with 256Mb of RAM, running Debian GNU/Linux 2.0.

7.1 Comparison between Graphplan-based planners

We compared four planners based on Graphplan in the Logistics domain: IPP v4.0, STAN v3.0, GP and LCGP. IPP and STAN are highly optimized planners implemented in C for IPP, and in C++ for STAN. We used the 30 problems given in the BLACKBOX distribution (Kautz and Selman 1999).

One of the particularities of the Logistics domain is that plans can contain a lot of parallel actions: Graphplan finds many independent actions, so there is fewer constraints (in relation to the number of actions) than in other domains, like blocks-world domain with one arm. However, numerous constraints found by Graphplan can be relaxed by LCGP to become authorization constraints. For example, in Graphplan, the two actions "load a package in an airplane at place A" and "fly this airplane from place A to place B" are not independent: one precondition of the first action (the airplane must be at place A) is deleted by the second action. In LCGP, the first action authorizes the second so they can appear simultaneously in an authorized set. The results of these tests are shown in Table 1.

Among the three planners based on Graphplan (which use the independence relation), STAN is the most efficient. Two reasons can explain this result: STAN has the EBL/DDB capacities described in (Kambhampati 1999b), and it preserves only the actions that are relevant for each problem thanks to its pre-planning type analysis tools (Fox and Long 1998). Then comes GP, which solves fewer problems than STAN but significantly more than IPP. GP is faster than IPP except on 2 problems. This can be explained by the EBL/DDB capabilities of GP.

Our planner, LCGP, solves all the problems with extremely good performances compared to the other planners. STAN is however faster than LCGP in 9 problems, but there is no doubt about the possible efficiency of LCGP if it had the same features as STAN (C++ implementation and pre-planning analysis tools). In most of the problems, the planning-graph construction takes almost all the time: the search time is then negligible. Only a few problems (*log.c*, *log017*, *log020*, *log023*) take relatively more time due to the hardness of search in the second stage. The improvement is evident: LCGP runs on the average 1800 times faster than GP on the problems solved by both planners. This result can be explained by the reduction of the search space (cf. § 6, the number of levels needed to solve the problems).

None of these planners produces systematically optimal solutions (in number of actions), but their plans contain approximately the same number of actions. LCGP is not optimal in number of levels (in the sense of Graphplan, with the independence relation: LCGP is optimal in number of levels with the authorization relation), but the size of the plan does not seem to be degraded: even more, LCGP sometimes gives the best solution (cf. *log010*, *log013*, *log025*...).

7.2 An efficient heuristic for variable and domain orderings

The planning-graph of Graphplan and the dynamic constraint satisfaction problem are closely connected as demonstrated in (Kambhampati 1999b). A proposition p

	CPU time (sec.)				Ratio TimeGP/ TimeLCGP	Actions				Levels		
										GP	LCGP	
Problems	**IPP**	**STAN**	**GP**	**LCGP**		**IPP**	**STAN**	**GP**	**LCGP**		**(+)**	**(++)**
log.easy	0.06	0.05	0,41	0,32	1.29	25	25	25	25	9	9	6
rocket.a	23.09		16,29	0,49	33.05	30		30	28	7	7	4
rocket.b	34.40	24.34	5,85	0,40	14.62	26	26	26	26	7	7	4
log.a	2,174.07	4.34	164,01	1,23	133.56	54	54	54	54	11	11	7
log.b	5,820.92	5.67	76 402,09	1,78	43,043.43	45	44	45	45	13	13	8
log.c		6,135.85	≥86,400	227,69	≥379		52		53	≥11	13	8
log.d		22,105.24		3,89			68		73		15	9
log.d3		≥86,400	≥86,400	4,46	≥19,355				72	≥13	13	8
log.d1			≥86,400	23,88	≥3,619				68	≥14	17	10
log010	1,861.77	0.59	28,53	3,28	8.70	43	43	42	41	10	11	7
log011		218.03	8 635,85	2,18	3,965.04		48	48	49	11	11	7
log012	74.40	0.77	6,61	1,17	5.64	38	38	38	38	8	8	5
log013		523.24	4 526,88	3,70	1,222.82		67	67	66	11	11	7
log014	122.65	1.17	6,04	5,00	1.21	70	71	70	75	10	11	7
log015		≥86,400	≥86,400	4,26	≥20,267				61	≥11	13	7
log016				4,13					40		16	9
log017			≥86,400	101,07	≥855				44	≥16	17	10
log018		3.04	40,74	5,58	7.31		48	52	50	11	11	7
log019		5.77	14,46	2,60	5.56		47	46	50	11	12	7
log020			≥86,400	578,01	≥149				87	≥14	15	9
log021		3,259.27	2 594,30	4,20	617.25		63	63	66	11	12	7
log022			≥86,400	4,18	≥20,690				74	≥14	15	9
log023		232.42	≥86,400	90,50	≥955		61		61	≥13	13	8
log024		286.80	3 349,58	3,96	846.71		64	64	67	12	13	8
log025		1.46	12,62	3,38	3.74		57	58	56	12	13	8
log026		0.64	4,85	3,10	1.56		51	50	50	12	12	8
log027		23.15	≥86,400	3,41	≥25,345		71		72	≥13	14	8
log028			≥86,400	11,61	≥7,445				78	≥14	14	9
log029	3,558.05	1.19	8,27	5,96	1.39	46	49	46	46	10	11	7
log030		1.11	49,54	3,06	16.21		52	52	52	13	13	8
Mean (*)	1,518.82	1,563.53	5,325.94	2.85	1,866.28	41.89	52.33	48,67	49.11	10,50	10,89	6.78
Mean (**)	-	-	≥34,280.96	36.95	≥927.82	-	-	-	55.57	≥11.50	12,37	7.53

(+) number of levels of the plan after transformation by the algorithm of § 6.3
(++) number of levels of the plan before transformation by the algorithm of § 6.3
(*) mean of solved problems (white cells). For LCGP : mean of problems solved by Graphplan.
(**) mean of the 30 problems.
grey cell: failure in the resolution of the corresponding problem.

Table 1: Comparison between Graphplan-based planners in the Logistics domain

at a level n in the planning-graph corresponds to a single variable p_n in the dynamic CSP framework; and the set of actions D that establish this proposition at this level n in the planning-graph corresponds to the domain D_{p_n} of the variable p_n in the dynamic CSP framework.

Two orderings have a great influence on Graphplan's search. On one hand, is the ordering on variables during search, also known as dynamic variable ordering (DVO). (Kambhampati 1999b) reports limited improvements in performance, using the following heuristic: choose first the goal with the least establishers. This heuristic has a limited effect too when allying DVO and forward checking (we select then the goal that has the least *remaining* establishers, after pruning values from domains by forward checking). On the other hand, is the ordering on the values of the domains. This ordering can also be considered dynamically during search (see sticky values and folding the domain in (Kambhampati 1999b)).

We describe here a simple domain-independent heuristic for DVO and for static domain ordering (domains are ordered before the search stage), that gives good results with LCGP. This heuristic is very efficient (see Table 2) in several domains (Ferry, Gripper, Blocks-world, Logistics...), but leads to bad results in the Tower of Hanoi domain.

The idea is the following: for DVO, we select first the proposition whose starting level[1] is the highest; for domain ordering, we select first the action whose starting level is the lowest. Indeed, to minimize the search space when attempting to satisfy a set of propositions, we must consider first the most constrained propositions: the ones that appear in a high level are the most likely to have still mutexes between their establishers (because mutual exclusions between propositions and actions tend to

[1]By starting level of a proposition (or action), we mean the number of the first level in which this proposition (or action) appears.

disappear when the planning-graph grows). On the other hand, we choose first the establishers that appear the earliest in the planning-graph because their preconditions are more likely to be no more mutually exclusive.

The usual strategy for static domain ordering consists in privileging the choice of no-ops. Using another strategy

Problems	CPU time (sec.)		Ratio TimeLCGP/	Expanded nodes (*)		Actions		Levels	
	LCGP	+DVO	TimeLCGP+DV(	LCGP	+DVO	LCGP	DVO	(+)	(++)
ferry6	3,05	0,30	10,17	10 512	654	23	23	23	12
ferry8	387,51	2,51	154,39	995 339	4 958	31	31	31	16
gripper6	1,45	0,39	3,74	3 962	528	17	17	11	6
gripper8	165,81	8,02	20,68	409 019	9 489	23	23	15	8
bw-large-a	3,42	2,49	1,37	171	87	12	12	12	12
bw-large-b	257,65	19,13	13,47	22 359	6 866	18	18	18	18
log.c	227,69	1,81	125,80	443 054	272	53	62	13	8
log020	578,01	8,38	68,97	764 804	4 199	87	93	15	9
hanoi5	8,41	10,48	0,80	3 885	10 326	32	32	32	21

(+) number of levels of the plan after transformation by the algorithm of § 6.3

(++) number of levels of the plan before transformation by the algorithm of § 6.3

(*) expanded nodes for LCGP corresponds to the number of calls of the function Find-Plan (see (Kambhampati 1999b)) modified as stated in § 6.2. Find−Plan is called one time for each set of propositions to be establis

Table 2: Benefits of the DVO heuristic

Problems	Total time (sec.)		Search time (sec.)		Expanded nodes		Actions	
	HSP-R	LCGP	HSP-R	LCGP	HSP-R	LCGP	HSP-R	LCGP
log.easy	0,04	0,32	0.009	0.001	50	7	27	25
rocket.a	0,04	0,34	0.017	0.015	59	36	28	26
rocket.b	0,04	0,36	0.018	0.005	60	7	29	28
log.a	0,09	1,02	0.051	0.005	191	11	67	65
log.b	0,08	1,38	0.038	0.094	137	272	51	51
log.c	0,14	1,81	0.084	0.157	236	272	69	62
log.d	0,43	3,60	0.251	0.009	280	13	81	75
log.d3	0,58	3,81	0.354	0.007	317	9	82	78
log.d1	0,29	3,57	0.147	0.141	219	177	77	75
log010	0,38	3,28	0.194	0.004	179	8	46	43
log011	0,21	1,89	0.096	0.009	156	13	54	55
log012	0,17	1,14	0.072	0.003	142	6	41	40
log013	0,50	3,35	0.295	0.007	268	8	74	74
log014	0,57	4,38	0.310	0.006	263	8	82	71
log015	0,43	3,90	0.222	0.477	222	406	69	65
log016	0,13	6,17	0.040	4.483	103	10 965	44	45
log017	0,13	2,61	0.046	0.750	118	3 067	48	43
log018	0,87	5,52	0.431	0.004	227	8	56	50
log019	0,29	2,62	0.127	0.004	146	8	50	52
log020	0,63	8,38	0.378	3.861	340	4 199	99	93
log021	0,56	4,30	0.330	0.006	276	8	69	67
log022	0,47	3,85	0.290	0.072	323	101	87	86
log023	0,35	3,77	0.177	0.586	197	415	70	65
log024	0,41	3,17	0.228	0.006	232	9	73	68
log025	0,36	3,14	0.187	0.006	206	9	67	64
log026	0,30	3,08	0.124	0.008	154	15	52	53
log027	0,41	3,30	0.235	0.011	260	15	76	76
log028	1,03	8,13	0.673	0.103	399	121	88	83
log029	0,79	5,94	0.434	0.006	248	8	50	48
log030	0,34	3,07	0.160	0.005	180	9	52	52
Mean	0.37	3,37	0.201	0,36	206,27	673,67	61.93	59,27

Grey cells: problems in which LCGP makes more backtracks than HSP-R and in which LCGP's search time is higher than HSP-R's one.

Table 3: Comparison LCGP+DVO and HSP-R in the Logistics domain

can lead to a degradation in the quality of the solution in number of actions of the plan (cf. Logistics domain, Table 2). However, for an efficiency purpose, we will employ our heuristic in what follows.

7.3 Comparison with HSP-R

Thanks to our DVO heuristic, LCGP is more competitive with HSP-R (Bonet and Geffner 1999), which actually seems to be faster than Graphplan-based planners. We compare LCGP and HSP-R on the Logistics domain, in which HSP-R runs very fast (see Table 3).

If we compare the total running time, we see that HSP-R is about 10 times faster than LCGP. But LCGP is implemented in Common Lisp + CLOS, and HSP-R in C which is certainly faster than Lisp. Furthermore, we have not included the compilation time of the problems for HSP-R, which took about 32 seconds (around 1 second per problem).

Most of the running time of LCGP is spent building the planning graph. Indeed, if we consider the only search time, LCGP is faster than HSP-R in 23 of the 30 problems, which correlates exactly with the number of expanded nodes. Furthermore, LCGP expands less than 15 nodes in 19 of the problems while HSP-R needs around 200 nodes on these problems.

If we now look at the quality of the solution, we see that whereas LCGP finds more actions using the DVO heuristic, it finds generally shorter plans than HSP-R.

7.4 Ferry and Gripper domains

Performances of LCGP in these domains are not as good as in the Logistics domain, but are around 8 times better than with GP (see Table 4 and Table 5). It is amazing to see that in the Ferry domain, whose problems have linear solutions, planning-graphs produced by LCGP are almost 2 times shorter than those of GP. Indeed, in LCGP, the actions "embark a car on side A" and "sail from side A to side B" can belong to the same authorized set, so as "debark a car at place B" and "sail from place B to place A".

7.5 Blocks-world domain

We used the Prodigy version of this domain, with 6 operators and one arm. As there is no parallelism at all to exploit, even for LCGP, the planning-graphs built by GP and LCGP are exactly the same; so the search stage is performed in exactly the same way. We could however expect LCGP to be slower than GP, because of the need to recognize the authorized sets (cf. § 6.3). But as there is no parallelism, a set of actions considered during search

Subgoals	CPU time (sec.)		Ratio TimeGP/ TimeLCGP	Expanded nodes		Actions		Levels		
								GP	LCGP	
	GP	LCGP		GP	LCGP	GP	LCGP		(+)	(++)
1	0,02	0,01	1,54	4	3	3	3	3	3	2
2	0,03	0,02	1,32	11	5	7	7	7	7	4
3	0,05	0,04	1,50	91	13	11	11	11	11	6
4	0,15	0,06	2,55	506	51	15	15	15	15	8
5	0,52	0,12	4,46	2 007	198	19	19	19	19	10
6	1,86	0,30	6,18	6 500	654	23	23	23	23	12
7	6,23	0,90	6,95	18 478	1 888	27	27	27	27	14
8	18,95	2,51	7,55	48 111	4 958	31	31	31	31	16
9	55,17	7,69	7,18	117 884	12 199	35	35	35	35	18
10	152,63	21,05	7,25	276 770	28 695	39	39	39	39	20
11	421,15	55,54	7,58	630 371	65 415	43	43	43	43	22
12	1 117,30	147,05	7,60	1 404 787	145 869	47	47	47	47	24

(+) number of levels of the plan after transformation by the algorithm of § 6.3
(++) number of levels of the plan before transformation by the algorithm of § 6.3

Table 4: Comparison in the Ferry domain

Subgoals	CPU time (sec.)		Ratio TimeGP/ TimeLCGP	Expanded nodes		Actions		Levels		
								GP	LCGP	
	GP	LCGP		GP	LCGP	GP	LCGP		(+)	(++)
2	0,03	0,03	1,08	4	3	5	5	3	3	2
4	0,14	0,06	2,25	299	20	11	11	7	7	4
6	3,05	0,39	7,86	6 750	528	17	17	11	11	6
8	65,09	8,02	8,12	97 633	9 489	23	23	15	15	8
10	905,94	112,82	8,03	928 124	86 076	29	29	19	19	10
12	###	1 203,65	8,36	6 818 442	585 934	35	35	23	23	12

(+) number of levels of the plan after transformation by the algorithm of § 6.3
(++) number of levels of the plan before transformation by the algorithm of § 6.3

Table 5: Comparison in the Gripper domain

contains only one "real" action (all the others are no-ops). It is not useful to perform the authorization test on a set of actions containing less than three "real" actions, because:

- one no-op always authorizes another no-op;
- if an action does not authorize a no-op, then the no-op does not authorize the action, so they are mutually exclusive (and vice versa);
- two actions that do not authorize themselves are mutually exclusive.

Thus the test of authorization of a set of actions is performed on the set of the "real" actions of this set, if they are at least three. This explains why LCGP and GP perform exactly the same in this domain (for example 19.13 secs. in the problem *bw-large-b*).

8 Conclusion

None of the earlier improvements of Graphplan never modified the structure of the planning-graph, with the except of the modifications used to improve the expressiveness of the description language (conditional effects, quantification...) or to take into account uncertainty. In Graphplan, the structure of the graph is based on the concept of independence between actions, that allows the generation of plans with parallel actions.

In this paper, we demonstrate that this condition can advantageously be replaced by a less restrictive one: the authorization between actions. The search space which is then developed by LCGP becomes more compact (fewer levels than Graphplan), which tremendously speeds up the search time in some domains. The loss of optimality in the sense of Graphplan (in number of levels) does not appear to be significant, compared to the gain in efficiency. Furthermore, the optimality in number of actions is not related to the optimality in number of levels (when parallelism is possible), so LCGP can give better solutions (in number of actions) than Graphplan.

We also propose a domain-independent heuristic for variable and domain orderings that greatly improves LCGP, but can degrade the quality of the plan. On the Logistics domain, LCGP becomes competitive with HSP-R, a very efficient heuristic search planner.

References

Bäckström C. 1998. Computational aspects of reordering plans. In *Journal of Artificial Intelligence Research* 9:99–137.

Blum A. and Furst M. 1995. Fast planning through planning-graphs analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

Bonet B. and Geffner H. 1999. Planning as heuristic search: new results. In *Proceedings of the Fifth European Conference on Planning (ECP'99).*

Cayrol M.; Régnier P. and Vidal V. 2000. LCGP : une amélioration de Graphplan par relâchement de contraintes entre actions simultanées. To appear in *Actes du Douzième Congrès de Reconnaissance des Formes et Intelligence Artificielle (RFIA'2000).*

Fox M. and Long D. 1998. The automatic inference of state invariants in TIM. In *Journal of Artificial Intelligence Research* 9:367–421.

Fox M. and Long D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, 956–961.

Guéré E. and Alami R. 1999. A possibilistic planner that deals with non-determinism and contingency. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, 996–1001.

Kambhampati S. 1999a. Improving Graphplan's search with EBL & DDB techniques. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, 982–987.

Kambhampati S. 1999b. Planning-graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP Techniques in Graphplan. To appear in *Journal of Artificial Intelligence Research.*

Kautz H. and Selman B. 1999. Unifying SAT-based and Graph-based Planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, 318–325.

Koehler J. 1998. Planning under resources constraints. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI'98)*, 489–493.

Koehler J.; Nebel B.; Hoffmann J. and Dimopoulos Y. 1997. Extending planning-graphs to an ADL subset. In *Proceedings of the Fourth European Conference on Planning (ECP'97),* 273–285.

Long D. and Fox M. 1999. The efficient implementation of the plan-graph in STAN. In *Journal of Artificial Intelligence Research* 10:87–115.

Nebel B.; Dimopoulos Y. and Koehler J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proceedings of the Fourth European Conference on Planning (ECP'97)*, 338–350.

Régnier P. and Fade B. 1991. Complete determination of parallel actions and temporal optimization in linear plans of actions. In *Proceedings of the European Workshop on Planning (EWSP'91)*, 100–111.

Vidal V. 2000. Contribution à la planification par compilation de plans. Rapport IRIT 00/03–R, Université Paul Sabatier, Toulouse, France.

Weld D.; Anderson C. and Smith D. 1998. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, 897–904.

Zimmerman T. and Kambhampati S. 1999. Exploiting symmetry in the planning-graph via explanation-guided search. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, 605–611.