Least-commitment temporal planning*

Antonio Garrido, Eva Onaindia

Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia (Spain)
{agarridot,onaindia}@dsic.upv.es

Abstract. We present a search process to avoid the inefficiencies of a Graphplan backward search in a temporal planning approach. Our proposal consists of a two-stage process which first extracts a relaxed plan which is next used as the basis of a heuristic search.

1 Introduction

Many of the current challenges in planning focus on increasing and improving the functionalities of the planners for dealing with more real features, such as temporal capabilities, more expressive domain definition languages, heuristic techniques and optimisation criteria, etc. [1–4]. In particular, 10 of 14 planners which participated in the last International Planning Competition (IPC–2002) offered temporal capabilities, what shows the need to push forward the capabilities of the planning systems.

This paper deals with three of the previous functionalities: i) planning with temporal features (actions with duration), ii) more expressive domain definition languages (PDDL2.1 [1]), and iii) plan optimisation (makespan). Traditional temporal planners have adopted a conservative model of actions, which means that two actions cannot overlap in *any way* if they have conflicting preconditions or effects. However, there exist planning problems that require a richer model of actions. The domain definition language used in the IPC–2002 was PDDL2.1 [1], which provides a model of durative actions. This model allows a more accurate exploitation of action concurrency in order to obtain better quality plans.

This paper describes our experiences with a Temporal Planning SYStem (from now on TPSYS [2]) to manage the model of durative actions proposed in PDDL2.1. TPSYS performs a Graphplan backward search, which guarantees the properties of completeness and optimality *w.r.t.* makespan. However, backward search has some inefficiencies that impose limitations when dealing with temporal, large problems. In order to overcome these limitations we suggest a modification in the search process which is based on a two-stage search process. First, a backward search generates an initial relaxed plan. Next, this relaxed plan is used as the basis for generating a solution plan by means of a non-complete

* This work has been partially supported by the Spanish MCyT under projects TIC2001-4936-E and DPI2001-2094-C03-03, and by the Universidad Politécnica de Valencia under projects 20010017 and 20010980.

heuristic process. The time of execution of actions is only committed when actions are applicable and no mutex holds. This allows to speedup the performance of search to be able to solve problems with more than a few actions, producing non-optimal, but good quality, plans.

2 Review of TPSYS

TPSYS is based on a three-stage process, which combines the ideas of Graphplan and TGP [2, 4, 5]. This means that TPSYS incrementally extends a *temporal* planning graph, performs a backward search through that graph and extracts a feasible, optimal plan.

In TPSYS , a temporal planning problem is specified as the tuple $\{\mathcal{I}_s, \mathcal{A}, \mathcal{F}_s, \mathcal{D}_{max}\}$, where $\mathcal{I}_s$ and $\mathcal{F}_s$ represent the initial and final situation. $\mathcal{A}$ represents the set of durative actions. Unlike *conservative* actions, durative actions present more conditions to be guaranteed for the success of the action. These conditions are $SCond_a$, $ECond_a$ and Inv_a with the conditions of a to be guaranteed at the start, end and over all the execution of a, respectively. Durative actions have also two types of effects: $SEff_a$ and $EEff_a$ with the effects to be asserted at the start and end of a, respectively. Finally, $\mathcal{D}_{max}$ stands for the maximum duration allowed by the user (time is modelled by $\mathbb{R}^+$ and their chronological order).

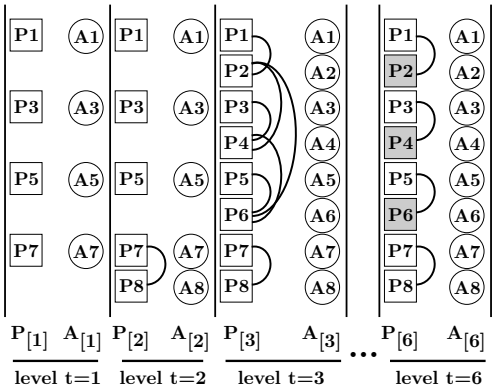
Like other Graphplan-based planners, TPSYS identifies binary mutual exclusion relations between actions and between propositions. The first stage of TPSYS calculates the action-action (AA) and proposition-action (PA) static mutex relationships. These mutex relationships are static because they only depend on the definition of the actions and they always hold.

The second stage extends a temporal planning graph which alternates temporal levels of propositions ($P_{[t]}$) and actions ($A_{[t]}$). Those levels are chronologically ordered by their instant of time t representing the time in which propositions are present and actions can start or end. Action-action ($AA_{[t]}$), proposition-action ($PA_{[t]}$) and proposition-proposition ($PP_{[t]}$) mutex relationships are calculated during the extension of the temporal planning graph. Mutex information provides useful information to improve search efficiency.

The third stage performs the extraction of an optimal plan, as an acyclic flow of actions, through the temporal planning graph. This stage is based on a backward Graphplan search. Since the third stage starts as soon as all the propositions in the final situation are present, non pairwise mutex, and the plan extraction is complete, TPSYS obtains the plan of optimal makespan. The properties of completeness and optimality make the search stage the most time consuming — like in most of Graphplan-based planners. In consequence, improving this stage seems to be a worth-while activity.

3 Some inefficiencies of Graphplan-based backward search

In order to motivate the need of improving the backward search in a Graphplan-based planner, we discuss the behaviour of that search in the simple plan-



Actions	Propositions
A1-board(c1,f1,l1)	P1-at(c1,l1)
A2-debark(c1,f1,l2)	P2-at(c1,l2)
A3-board(c2,f1,l1)	P3-at(c2,l1)
A4-debark(c2,f1,l2)	P4-at(c2,l2)
A5-board(c3,f1,l1)	P5-at(c3,l1)
A6-debark(c3,f1,l2)	P6-at(c3,l2)
A7-sail(f1,l1,l2)	P7-at(f1,l1)
A8-sail(f1,f2,l1)	P8-at(f1,l2)

Fig. 1. Outline of the planning graph for the `ferry` problem. Shaded propositions represent the non-pairwise mutex problem goals at level 6. Mutex relations between propositions are represented by thick lines. For simplicity, inverse actions of *board* and *debark* (for instance, *debark(c1,f1,l1)*) are not represented.

ning problem called `ferry`. The domain consists of transporting a number of cars from one location to another using a ferry which can carry only one car at a time. To keep the problem simple enough we assume three cars $c1$, $c2$ and $c3$ to be transported from location $l1$ to $l2$ by ferry $f1$. The actions are *board(?car,?ferry,?locat)*, *sail(?ferry,?locat1,?locat2)* and *debark(?car,?ferry,?locat)*.

The makespan of the feasible plan is 11, but the backward search starts from time 6 (see Fig. 1) because the mutex relations are binary. Thus, at time 6 every pair of the problem goals {`P2,P4`}, {`P2,P6`}, {`P4,P6`} are non pairwise mutex, and the planner attempts to extract one plan from here. Since every pair of actions is mutex, only one action is planned in each level. For simplicity, we assume that applicable actions are selected in each level from top to bottom in the planning graph of Fig. 1. This way, action `A2` is firstly planned at time 6. Next, actions `A7` and `A1` are planned at time 5 and 4, respectively. Then, `A4` is planned at time 3, but no feasible plan is found, and the chronological backtracking leads to plan `A6` at time 3, again with no success. After that, the chronological backtracking leads to time 6 in which firstly action `A4` and secondly action `A6` are planned with no success. This is the first indication of inefficiency in the backward search: a lot of effort is wasted trying, unsuccessfully, to plan nearly *identical* actions in problems with symmetry among goals. In this example all permutations of actions `A2`, `A4`, `A6` are planned under the same schema.

Finally, as no plan is found from time 6, the planning graph is extended to time 7 and the search is re-started from scratch. This is the second indication of inefficiency. When search is re-started from a new level, no actions planned in previous stages of search are reused as part of the current plan, committing similar failures in the new *one-deeper-level* search space. In this example, the plan for transporting at least one car could be reused from the previous stage

of search, but this is never considered. Fortunately, memoization techniques [5, 6] remember many conflicting propositions thus reducing the amount of failures committed during search. However, the way in which memoization prunes one branch of search does not allow to solve the real conflict until exhausting all the actions from the current level. This is the third indication of inefficiency. If one proposition becomes unsupported, it could be more efficient to plan a new action which achieves that proposition than discarding that branch and backtrack. In this example, if the ferry is required in $l1$ it is better to insert the action *sail(f1,l2,l1)* (A8) in the plan —extending the planning graph if necessary— than discarding the plan and backtrack. This would allow to guide the search to achieve the goals instead of continuing in a blind fashion.

Furthermore, when dealing with temporal planning problems where actions have different duration the influence of these inefficiencies is much more significant. In that case, the number of levels generated in the temporal planning graph is higher, and all the previous inefficiencies and wasted search is repeated more frequently. In particular, if we model the actions *board(?car,?ferry,?locat)*, *sail(?ferry,?locat1,?locat2)* and *debark(?car,?ferry,?locat)* with durations 1, 5 and 2, respectively, the backward search for this example starts after extending 16 levels but no plan is found until level 34 is extended and explored. This indicates how these inefficiencies are more outstanding in temporal approaches based on Graphplan backward search.

4 Combining least-commitment and heuristics to improve the search process

In this section we describe how the search process can be improved by combining least-commitment and heuristic techniques. Our new approach consists of two stages. First, a backward chaining stage generates an initial relaxed plan from the information of the temporal planning graph. Actions, which can be mutex, in this relaxed plan support all the problem goals but with no commitment on their start time. Second, a forward chaining stage allocates the execution time of the actions in the relaxed plan. While allocating actions, mutex between actions break the initial plan *relaxation*, and subsequently new actions must be planned to solve these mutexes.

4.1 Generation of an initial relaxed plan

This stage generates an initial relaxed plan from the information of the temporal planning graph (from now on TG). We define a relaxed plan Π as a partially ordered set of actions in which both the problem goals and action preconditions are satisfied. It is called *relaxed* because no mutex relationships are considered during its generation.

Plan Π is generated in a similar way to the relaxed plan solution in FF [7] with the exception that we handle durative actions (see Algorithm 1). Starting from the level in which the TG extension finishes, we simply insert into Π the

actions with the earliest start time of execution which support the preconditions of FS (final fictitious action in a relaxed plan). It is important to note that actions are not committed in time yet. Next, start, invariant and end conditions of the inserted actions are supported in the same way, and so on. The process finishes once all the (sub)goals are supported by actions in Π.

1: goals $\leftarrow \mathcal{F}_s$
2: $\Pi \leftarrow \{\texttt{IS} \cup \texttt{FS}\}$ {*initial and final fictitious actions*}
3: **while** goals $\neq \emptyset$ **do**
4: extract g_i from goals
5: **if** g_i is not supported in Π **then**
6: $a \leftarrow \arg\min_{\forall a_i \text{ which supports } g_i}$(earliest start time from TG)
7: **if** a is the only action which supports g_i **then**
8: mark a as obligatory in Π
9: $\Pi \leftarrow \Pi \cup \{a\}$
10: goals $\leftarrow$ goals $\cup \{SCond_a \cup Inv_a \cup ECond_a\}$

Algorithm 1: Generation of an initial relaxed plan Π.

Note that in step 6 we do not need to perform real search —or backtracking— because none of the mutexes are considered at this point and, therefore, this strategy always leads to a solution.

One important property of Π is that if there is no mutex between actions whose execution overlaps, all actions in Π form a feasible, optimal plan. Unfortunately, this is not a very common situation and mutex relations break the plan *relaxation*. This entails to postpone the allocation of actions, and/or to plan new actions to solve the unsupported (sub)goals.

4.2 Planning and allocating of actions

This stage allocates actions in time in the relaxed plan. This is done through a search in a plan space (called `set_of_plans`) formed by all the generated plans $\{\Pi_i\}$. Actions in each Π_i are divided into two disjunctive sets: $Alloc_i$ and $Relax_i$. $Alloc_i$ is formed by the actions which have been allocated in time and will never be removed from Π_i. $Relax_i$ is formed by the actions which have not been allocated yet, and so they can be removed from Π_i. Initially, $Alloc_i$ is empty and $Relax_i$ contains all the actions in Π_i. This stage finishes once $Relax_i$ gets empty, obtaining in $Alloc_i$ the actions of the plan.

The idea is to move forward in time, simulating the real execution of Π_i, progressively taking care of the actions which can start their execution. The current time of execution in Π_i, represented by $\texttt{time\_of\_execution}_i$, is initialised to 0. Algorithm 2 describes the behaviour of this stage. The algorithm always selects the plan Π_i with lowest cost from `set_of_plans` trying to satisfy all the problem goals (step 3). If the goals are supported the algorithm exits with success (step 5). Otherwise, available actions from $Relax_i$ are tried to be allocated in

```
1:  set_of_plans ← Π, generated in Algorithm 1
2:  while set_of_plans ≠ ∅ do
3:    extract the lowest cost Π_i from set_of_plans
4:    if Alloc_i supports all the problem goals then
5:      exit with success
6:    else
7:      if ∀a_j ∈ Alloc_i | ∃g_j ∈ ECond(a_j) which is unsolved then
8:        insert new plans into set_of_plans to satisfy g_j
9:      else
10:       a ← arg max(allocation priority)
                ∀a_j∈Relax_i available at time_of_execution_i
11:       if a is mutex in Alloc_i then
12:         if a is not obligatory then
13:           remove a from Π_i
14:       else
15:         if a is applicable then
16:           allocate a in Alloc_i to be started at time_of_execution_i
17:         else
18:           insert new plans into set_of_plans to make a applicable
19:   update time_of_execution_i
```

Algorithm 2: Planning and allocating of actions.

the current $\texttt{time\_of\_execution}_i$. If action a is non-obligatory and mutex with actions in $Alloc_i$, a is removed from Π_i (step 13), delaying the fulfillment of its goals to a future time of execution. The reason to remove a non-obligatory action is that it could be a bad choice for the plan. If a is not mutex and applicable, a is allocated in time (step 16). Branching points are in steps 8 and 18, in which it is necessary to insert new actions to achieve unsolved propositions. For each action a_j which supports one of these propositions, a new plan Π_j is generated with a_j marked as obligatory in Π_j. It is important to note that a_j is not allocated in time, but it is inserted with its earliest start time of execution extracted from the *TG*. This is part of the least-commitment technique performed in the allocation of actions when they are inserted into plans.

The algorithm has two important points of selection (steps 3 and 10). In step 3, the plan Π_i with the lowest cost from `set_of_plans` is selected. This cost is estimated as follows:

$$
\begin{aligned}
cost(\Pi_i) &= cost(Alloc_i) + cost(Relax_i), \text{ where} \\
cost(Alloc_i) &= \alpha \cdot unsup(\texttt{FS}, Alloc_i) + \beta \cdot duration(\Pi_i), \text{ and} \\
cost(Relax_i) &= \sum_{\forall a \in Relax_i} \gamma \cdot unsup(a, Alloc_i) + \delta \cdot add(a, \texttt{FS}) + \theta \cdot del(a, \texttt{FS}) + \\
&\quad \lambda \cdot PA\_mutex(a, \texttt{FS}) + \mu \cdot duration(a)
\end{aligned}
$$

The cost of a plan Π_i is the sum of the cost due to $Alloc_i$ and $Relax_i$. $unsup(action, Alloc_i)$ is an estimation of the number of actions necessary to

achieve the conditions of *action* in $Alloc_i$ from the current $\texttt{time\_of\_execution}_i$. $add(a, \texttt{FS})$ is the number of conditions of FS which a supports, whereas $del(a, \texttt{FS})$ is the number of conditions of FS which a deletes. $PA\_mutex(a, \texttt{FS})$ is the number of mutex between conditions of FS and action a. This mutex information is extracted from the TG and it indicates the impossibility to have simultaneously the conditions of FS and a. *duration* represents the duration of the plan/action. Note that $\alpha, \beta, \gamma, \theta, \lambda, \mu \geq 0$ because they have a positive impact in the cost of the plan, whereas $\delta \leq 0$ because it has a negative impact in the cost of the plan.

In step 10, the available action a with the maximal allocation priority in $Relax_i$ is selected to be studied at time $\texttt{time\_of\_execution}_i$. This priority is estimated as follows:

$$prio(a) = \rho \cdot unsup(a, Alloc_i) + \sigma \cdot succ(a, Relax_i) + \tau \cdot meetsucc(a, Relax_i) + \psi \cdot duration(a)$$

The allocation priority of action a depends on several factors. $succ(a, Relax_i)$ is the number of direct successor actions of a in $Relax_i$. $meetsucc(a, Relax_i)$ is the number of direct successor actions of a which can start as soon as a ends, i.e. which meet a. These two values indicate the importance of an action for the successor actions in $Relax_i$. Intuitively, the more successor actions a has, the more important a is in the plan. Similarly, the *meeting* successor actions indicate the number of successor actions which can be executed without mutex. *unsup* and *duration* are defined as above. Coefficients $\sigma, \tau \geq 0$ as they are used to select the next appropriate action to be allocated. However, $\rho, \psi \leq 0$ because they indicate that action a is not promising enough to be allocated yet.

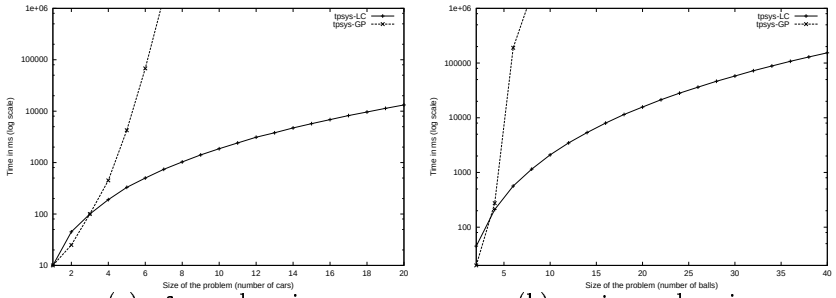
The previous evaluation functions can have many more heuristic factors. However, according to our experiments these factors are general enough for most temporal planning problems. Moreover, an important point is that the precise value of each coefficient is not so relevant as in other heuristic approaches based on local search [3].

5 Preliminary results

We have implemented the previous search process in TPSYS, conducting two experiments[1]. We wanted to compare the performance of the new search process *vs.* the backward Graphplan search. Obviously, the quality of the plans will be different, but our interest focusses on the capability of the planner to solve more complex problems. In the first experiment, we have studied the impact of the new search process in problems with a high degree of symmetry among goals. Therefore, we have used two well-known planning domains with symmetry among goals, which are the `ferry` and the `gripper` domain. Fig. 2 shows the results for some problems of these domains and demonstrates that the least-commitment search (`tpsys-LC`) scales up much better than Graphplan backward

[1] All the experiments were run on a Celeron 400MHz with 256 Mb

search (`tpsys-GP`), even in the simpler problems. Moreover, every plan found by both approaches is optimal.



(a) - `ferry` domain (b) - `gripper` domain

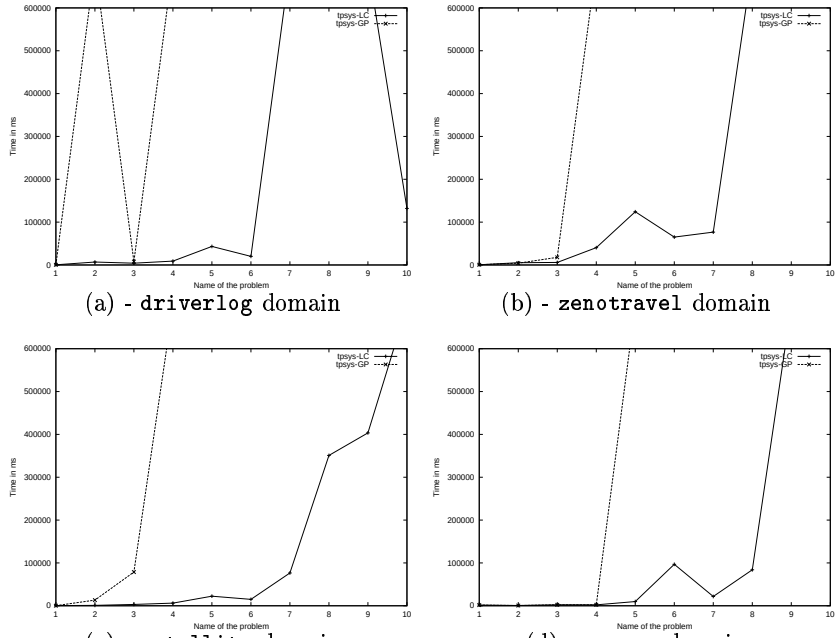
Fig. 2. Impact of the redundancy among goals in the two search processes.

The second experiment deals with more general problems. In particular, we tested the performance of both approaches in some domains of the last IPC–2002 competition. These domains represent transportation problems, space-applications problems, etc. Fig. 3 shows the results of these comparisons.

Both search approaches have difficulties to solve all the problems, but least-commitment search can solve more problems and faster. Backward search difficulties rise as a result of the redundancy in the search process. Least-commitment search difficulties rise as a result of the heuristic, *greedy* approach, which can lead to the wrong path in the search. The `satellite`, `rovers` and `zenotravel` domains show the highest speedups. In the `driverlog` domain many of the paths to achieve the problem goals interact negatively, and the greedy approach gets stuck in the wrong path as shown in Fig 3-a.

6 Conclusions through related work

Graphplan has become a popular approach to planning, so many attempts to improve its functionalities and to overcome its main inefficiencies have been studied. In particular, several works have been done to reduce the redundancy in the search space. The work developed by Fox & Long in TIM [8] allows to extract information about the structure of the planning domain to eliminate irrelevant propositions and actions, thus reducing the number of actions to be considered during search. Moreover, detection and exploitation of symmetry have shown impressive results on the performance of the Graphplan-based planners [9, 10]. On the other hand, Kambhampati [6] has analysed the similarities of Graphplan and dynamic CSPs to provide several strategies, such as explanation based learning (EBL) and dependency directed backtracking (DDB), very useful for backward



(a) - `driverlog` domain (b) - `zenotravel` domain

(c) - `satellite` domain (d) - `rovers` domain

Fig. 3. Times of execution for some problems of the IPC–2002 (all tests were censored after 600 seconds).

search. In particular, EBL/DDB techniques and learning —memoizing— from failures significantly improves the backward search, reducing the amount of unsuccessful search. With some modifications [10], these last techniques also avoid the whole *re-play* of the search when re-starting from scratch from a new level.

Graphplan has also been augmented with least-commitment techniques in Cayrol's work [11]. In this case, the definition of mutex relationships is relaxed, which allows to achieve the problem goals earlier in the planning graph. Although this entails a more compact search space, it is necessary a posterior transformation of the plan to guarantee its executability. Our application of least-commitment to temporal planning is totally different to Cayrol's because we do not use a backward search through a planning graph. We generate a relaxed plan, similarly to FF [7], to be used as the basis of the solution plan. Next, the planner refines and allocates actions in time according to their mutex relations and to several local heuristic criteria in the line of LPG [3].

This paper contributes in the way in which least-commitment can be applied in a temporal approach. The techniques presented can be used in any Graphplan-based approach for planning, substituting the backward stage by the new two-stage search. An advantage of this kind of search is that it combines the information calculated in the planning graph with a heuristic, greedy process

of search. The process of search is guided by two estimation functions which use local features to select the next action or plan to be considered. The main disadvantage of this approach is that it is not complete preserving. However, although completeness and optimality are desired properties, guaranteeing them entails an exhaustive search, thus preventing planners from producing plans with more than a few actions. In particular, none of the planners with temporal features which participated in the last AIPS Planning Competition can guarantee optimal plans in reasonable times of execution. In fact, the outstanding planners in that competition are based on local search and hill-climbing techniques, which are not complete preserving. Our future work follows that line of research to improve the overall performance of the planner and the quality of the plans generated by a proper use of estimation heuristic functions to guide the search.

References

1. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK (2001)
2. Garrido, A., Fox, M., Long, D.: A temporal planning system for durative actions of PDDL2.1. In Harmelen, F.V., ed.: Proc. European Conf. on AI (ECAI-2002), Amsterdam, IOS Press (2002) 586–590
3. Gerevini, A., Serina, I.: LPG: a planner based on local search for planning graphs with action costs. In: Proc. 6th Int. Conf. on AI Planning and Scheduling (AIPS-2002). (2002) 281–290
4. Smith, D., Weld, D.: Temporal planning with mutual exclusion reasoning. In: Proc. 16th Int. Joint Conf. on AI (IJCAI-99), Stockholm, Sweden (1999) 326–337
5. Blum, A., Furst, M.: Fast planning through planning graph analysis. Artificial Intelligence **90** (1997) 281–300
6. Kambhampati, S.: Planning graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP techniques in graphplan. Journal of Artificial Intelligence Research **12** (2000) 1–34
7. Hoffmann, J.: A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In: Proc. 12th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, North Carolina, USA (2000)
8. Fox, M., Long, D.: The automatic inference of state invariants in TIM. Journal of AI Research **9** (1998) 367–421
9. Fox, M., Long, D.: The detection and exploitation of symmetry in planning domains. In Kaufmann, M., ed.: Proc. 15th International Joint Conference on AI. (1999) 956–961
10. Zimmerman, T., Kambhampati, S.: Exploiting symmetry in the planning-graph via explanation-guided search. In: Proc. Nat. Conf. on Artificial Intelligence. (1999)
11. Cayrol, M., Reginer, P., Vidal, V.: Least commitment in graphplan. Artificial Intelligence **130** (2001) 85–118