On the application of least-commitment and heuristic search in temporal planning

Antonio Garrido and **Eva Onaindia**
Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46071 Valencia, Spain
{agarridot,onaindia}@dsic.upv.es

Abstract

Graphplan planning graphs are structures widely used in modern planners. The exclusion relations calculated in the planning graph extension provide very useful information, especially in temporal planning where actions have different duration. However, Graphplan backward search has some inefficiencies that impose limitations when dealing with large temporal problems. This paper presents a new search process for temporal planning to avoid these inefficiencies. This search uses the information of a planning graph and shows beneficial in the scalability of the planner. Moreover, our experiments show that a planner with this new search is competitive with other state-of-the-art planners *w.r.t.* the plan quality.

1 Introduction

Many of the current challenges in AI planning focus on increasing the functionalities of planners to deal with more real features, such as temporal capabilities, explicit management of resources, more expressive domain definition languages, heuristic techniques and optimisation criteria, etc. [Fox and Long, 2001; Gerevini and Serina, 2002; Smith and Weld, 1999]. 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 [Fox and Long, 2001]), and iii) plan optimisation (makespan). Traditional temporal planners have adopted a conservative model of actions, where two actions cannot overlap in *any way* if they have conflicting preconditions or effects. This model is adequate in some planning domains, but there exist others that require a richer model of actions. Level 3 of PDDL2.1 used in IPC–2002 provides a model of durative actions which allows a more accurate exploitation of action concurrency to obtain shorter makespan plans.

This paper describes our experiences with a Temporal Planning SYStem (from now on TPSYS), based on Graphplan [Blum and Furst, 1997] and TGP [Smith and Weld, 1999], to manage the model of durative actions proposed in level 3 of PDDL2.1. TPSYS performs a Graphplan backward search, which guarantees the properties of completeness and optimality. However, backward search has some inefficiencies that impose important limitations in large temporal problems. In these problems, the search space is vastly increased and the performance of the algorithm degrades. We suggest a new two-stage search process to overcome these inefficiencies. First, a backward search generates an initial relaxed plan. Next, this relaxed plan is used as an outline for generating a solution plan by means of a non-complete heuristic process, where actions are only definitively allocated in the plan when they are applicable and no mutex (least-commitment). This allows to increase the scalability of search, producing non-optimal, but good quality, plans.

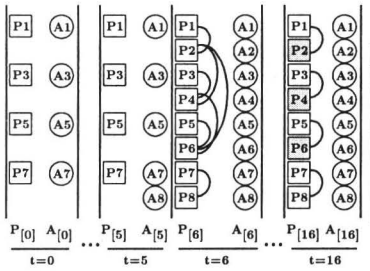
2 A Review of TPSYS

TPSYS is based on a three-stage process [Garrido *et al.*, 2002], which combines the ideas of Graphplan and TGP. This means that TPSYS incrementally extends a temporal planning graph, performs a backward search through that graph and extracts a plan.

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 two types of effects: $SEff_a$ and $EEff_a$ with the effects to be asserted at the *start* and *end* of a, respectively.

The first stage of TPSYS calculates the action-action and proposition-action 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 ($A_{[t]}$). Unlike Graphplan, levels in TPSYS represent instants of time $t \in \mathbb{R}^+$ in which propositions are present and actions can start/end. Action-action, proposition-action and proposition-proposition mutex relationships are calculated during the extension of the temporal planning graph. The third stage performs the search in a Graphplan backward way, extracting an optimal plan through the planning graph. 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.

Backward search in TPSYS preserves the same properties of completeness and optimality of Graphplan, but it en-



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)

Figure 1: Outline of the temporal planning graph for the `ferry` problem. Shaded propositions represent the non-pairwise mutex problem goals at time 16. Mutex relations between propositions are represented by thick lines.

tails the most time consuming stage. Furthermore, this search presents some inefficiencies (inherited from Graphplan) which impose important limitations in temporal problems. Let us consider a simple problem from the `ferry` domain. 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)*, with durations 1, 5 and 2, respectively. The optimal plan contains 11 actions (3x*board*, 5x*sail* and 3x*debark*) and the makespan is 34. The mutex relations are binary, so the second stage ends at time 16 and the search starts from there (see Figure 1). Since every pair of actions is mutex, only one action is planned in each level. If we assume that applicable actions are selected in each level from top to bottom in the planning graph of Figure 1, action `A2` is firstly planned at time 14. Next, actions `A7` and `A1` are planned at times 9 and 8, and so on. Because no feasible plan is found, the search backs to time 14 where all permutations of actions `A2`, `A4`, `A6` are planned under the same schema with no success. This is the first indication of inefficiency: a lot of effort is wasted trying, unsuccessfully, to plan nearly *identical* actions in problems with symmetry. As no plan is found from time 16, the planning graph is extended to time 17 and the search is re-started from scratch. This entails the second indication of inefficiency: no actions planned in previous stages of search are reused as a part of the current plan, committing similar failures in the new *one-deeper-level* search space. Although memoization helps reduce the amount of failures committed during search, the way in which it prunes one branch of search does not allow to solve the real conflict until exhausting the whole level. This is the third indication of inefficiency: if one proposition becomes unsupported, no new actions supporting it are studied but that branch is discarded and the algorithm performs a backtracking stage.

These inefficiencies have a negative influence in Graphplan-based temporal approaches. While a classical Graphplan's planning graph for this problem has 11 levels, in temporal planning no plan is found until level 34 is extended and explored. Particularly, in TPSYS the number of levels generated depends on the *dispersion* of the durations of the actions. The worst performance happens when the gcd of the durations is 1, forcing the algorithm to generate the maximum number of levels. In consequence, the previous inefficiencies and wasted search are repeated more frequently.

3 Combining Least-Commitment and Heuristics to Improve the Search Process

In this section we substitute the backward search by a two-stage search to avoid the previous inefficiencies in a temporal planning approach. First, a backward chaining stage generates an initial relaxed plan from the information of the temporal planning graph. Second, a forward chaining stage allocates the execution time of the actions in the relaxed plan.

3.1 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*) to be used as a *skeleton* of the final plan. We define a relaxed plan Π as a partially ordered set of actions in which both the problem goals and action conditions hold. It is called *relaxed* because neither mutex relationships between actions nor commitment on their start time are considered.

A relaxed plan always contains two fictitious actions with no duration called `IS` and `FS`. `IS` achieves the propositions of the initial situation, whereas `FS` requires the problem goals. Π is generated similarly to the relaxed solution plan in the FF planner [Hoffmann, 2000] with the exception that we handle durative actions (see Algorithm 1).

1: goals $\leftarrow$ problem goals {*must hold in any plan for the problem*}
2: $\Pi \leftarrow$ {`IS` $\cup$ `FS`} {*obligatory actions*}
3: **while** goals $\neq \emptyset$ **do**
4: extract g_i from goals
5: **if** g_i is not supported in Π **then**
6: $a \leftarrow \arg\min_{\forall a_i \text{ which supports } g_i}$(number of mutex which a_i imposes in Π)
7: **if** a is the only action which supports $g_i \wedge g_i$ is a condition of an obligatory action **then**
8: mark a as obligatory in Π
9: $\Pi \leftarrow \Pi \cup \{a\}$ {*no commitment on start time of a yet*}
10: goals $\leftarrow$ goals $\cup \{SCond_a \cup Inv_a \cup ECond_a\}$

Algorithm 1: Generation of an initial relaxed plan Π.

Step 6 studies the actions which support g_i and selects the action which minimises the number of mutex in Π. In problems with multiple resources, this selection tends to use as many resources as available. For instance, in a `ferry` problem with two or more ferries $f1$, $f2$,..., step 6 distributes the use of the ferries in a homogeneous way. If action *debark(c1,f1,l2)* is used for debarking the car $c1$, action *debark(c2,f2,l2)* is first selected when debarking the car $c2$ because *debark(c2,f1,l2)* is mutex with *debark(c1,f1,l2)* and imposes more mutex in Π. This selection provides the relaxed plan more information about the structure of the problem, thus increasing the quality (number of actions in parallel) of the final plan. Note that in step 6 we do not need to perform real search (or backtracking) because none of the mutexes are considered and, therefore, this strategy always leads to a solution. Moreover, we introduce the term of obligatory action

a as the only action which supports a goal which must hold in any plan for the problem. This is helpful because it means that a must be always present in Π (steps 7–8). Obviously, both IS and FS are obligatory.

One important property of Π is that if there is no mutex between overlapping actions, all actions in Π form a feasible, optimal plan. The proof of this is straightforward and relies on the complete extension of the temporal graph. The level in which the temporal graph extension ends indicates the minimal time in which the goals can be achieved by non-pairwise mutex actions. Thus, this level provides a minimal bound of the makespan for a feasible plan and, if no mutex between actions holds, the plan is not only feasible but also optimal. 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.

3.2 Planning and Allocating of Actions

This stage performs the allocation in time of actions in the relaxed plan. We use a structure called set_of_plans, with the search space formed by all the generated plans $\{\Pi_i\}$. Actions in each Π_i are divided into two disjunctive sets: $Alloc_i$ and $Relax_i$. $Alloc_i$ contains the actions which have been allocated in time and will never be removed from Π_i. $Relax_i$ contains 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 (the initial Π_i is the relaxed plan computed in the previous section). This stage finishes once $Relax_i$ gets empty, obtaining in $Alloc_i$ the actions of the plan which support all the problem goals.

```
 1: set_of_plans ← Π, generated in Algorithm 1
 2: while set_of_plans ≠ ∅ do
 3:    extract the lowest cost Πi from set_of_plans
 4:    if Alloci supports all the problem goals then
 5:       exit with success
 6:    else
 7:       a ← arg max(allocation priority)
              ∀aj ∈ Relaxi which can start at time_of_executioni
 8:       if a is mutex in Alloci then
 9:          if a is not obligatory then
10:             remove a from Relaxi
11:          else
12:             postpone start time of a in Relaxi
13:       else
14:          if a is applicable then
15:             allocate a in Alloci at time_of_executioni
16:          else
17:             insert new plans into set_of_plans to make a ap-
                plicable
18:    update time_of_executioni
```

Algorithm 2: Planning and allocating of actions.

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 (see Algorithm 2). The current time of execution in Π_i, $\texttt{time_of_execution}_i$, is initialised to 0. The algorithm always selects the plan Π_i of lowest cost from set_of_plans (step 3). If all the problem goals are supported the algorithm exits with success (step 5). Otherwise, actions from $Relax_i$ which can start at the current $\texttt{time_of_execution}_i$ are tried to be allocated (step 7). If one action a is mutex with actions in $Alloc_i$ and non-obligatory, a is removed from Π_i (step 10), 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 obligatory it is not removed but its start time is postponed (step 12). If a is not mutex and applicable, a is allocated in time (step 15). This step entails the first indication of loss of completeness. Although there exist alternative actions to a, if a can be allocated those actions are not considered in Π_i. Step 17 is a branching point in which new actions are inserted (generating new plans) to achieve unsupported conditions of a. For each action a_j supporting each condition a new plan Π_j is generated and inserted into set_of_plans 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. Step 18 moves to the next relevant $\texttt{time_of_execution}_i$ in which actions can start, extending the *TG* if necessary.

The algorithm has two important points of selection: steps 3 and 7. Step 3 selects the plan Π_i with the lowest cost from set_of_plans, where the cost is estimated as follows:

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

The cost of a plan Π_i consists of the sum of the cost due to $Alloc_i$ and $Relax_i$. $unsup(a, Alloc_i)$ is an estimation of the number of actions necessary to solve the unsupported conditions of a from the current $\texttt{time_of_execution}_i$. It is estimated through the *TG*, ignoring the delete effects of actions (as in the heuristic used in FF). $add(a, \texttt{FS})$ 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 indicates the impossibility to have simultaneously the conditions of FS and a. $duration$ represents the duration of the plan/action to take into account the makespan of Π_i. Note that $\alpha, \beta, \gamma, \theta, \lambda, \mu \geq 0$ because they have a positive impact in the cost of the plan. In opposition, $\delta \leq 0$ because it has a negative impact in the cost.

In step 7, the action a with the maximal allocation priority is selected to be studied at $\texttt{time_of_execution}_i$. This priority is estimated as follows:

$$
\begin{aligned}
prio(a) &= \rho \cdot unsup(a, Alloc_i) + \sigma \cdot succ(a, Relax_i) + \\
&\quad \tau \cdot meetsucc(a, Relax_i) + \psi \cdot duration(a)
\end{aligned}
$$

The allocation priority of action a depends on several local factors. $succ(a, Relax_i)$ is the number of direct successor

(dependent) 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 are directly supported by a, the more important a is in the plan. Similarly, the *meeting* successor actions indicate the number of successor actions which can be immediately executed without mutex. $unsup$ and $duration$ are defined as above. Coefficients $\sigma, \tau \geq 0$ because they imply to select an 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, but according to our experimental analysis they are general enough for most temporal planning problems. We can also implement different heuristic methods by setting the values of the coefficients, which in our case are the same for all the domains. Although the values of the coefficients can influence the search, their precise value is not as relevant as in other heuristic approaches based on local search as LPG [Gerevini and Serina, 2002]. For instance, the static cost of inserting an action in LPG is based on the number of unsupported conditions, what cannot represent the real complexity of solving each condition. On the contrary, the coefficient $unsup$ deals with that complexity because it estimates the real cost to support it in the *TG*.

It is important to note that plans are incrementally generated without discarding allocated actions and with no redundancy due to symmetry. Although Algorithm 2 explores the complete space of actions to make actions applicable, the allocation priority imposes an order of execution and discards the rest of feasible orderings (second indication of loss of completeness) . For instance, in the `ferry` problem of section 2, when studying the actions to *debark* {`A2`, `A4`, `A6`} the algorithm allocates one action and postpones the others. This avoids the complete exploration of all the permutations of `A2`, `A4` and `A6`, preventing the planner from the generation of symmetric plans.

4 Experimental Results

We have implemented the previous search on top of TPSYS, conducting several experiments[1]. The two first experiments compare the new search *vs.* the Graphplan backward search. In the **first experiment**, we studied the impact of the new search in problems with a high degree of symmetry, such as the `ferry` and `gripper`, focussing on the planner scalability. Figures 2-a,b show the results for some problems of these domains. Although it is not surprising that the least-commitment search (`tpsys-LC`) is faster than Graphplan backward search (`tpsys-GP`), it is worthy to mention that `tpsys-LC` scales up much better than `tpsys-GP`, especially in the `gripper` domain. Moreover, `tpsys-LC` found optimal plans without backtracking for all the problems.

The **second experiment** deals with some problems of the simple-time track of the last IPC–2002[2]. Although both approaches have difficulties to solve all the problems (see Figures 2-c,d,e,f), the least-commitment search can solve more problems. On one hand, backward search difficulties rise as a result of the redundancy in the complete, blind search process. On the other hand, least-commitment search difficulties rise as a result of: i) the heuristic, *greedy* approach, which can lead to the wrong path in the search, and ii) the non-complete preserving search. The `zenotravel` and `satellite` domains show the highest speedups. It is important to note that the best improvement is produced in the `satellite` domain, which is the only domain in the competition that exploits the end conditions of durative actions.

The third and fourth experiments are aimed at evaluating the quality of the plans generated by the new approach. We are mainly interested in the makespan of the plans, but we also consider the number of actions as an additional indication of the plan quality. The **third experiment** uses the plans generated in the experiment 1 (Figures 2-a,b) and compares them with the plans generated by LPG[3] [Gerevini and Serina, 2002] and Mips[4] [Edelkamp, 2002]. We have chosen these two planners as they handle temporal features and showed distinguished performance in the last IPC. To date our interest has not focussed on code optimisation, so in this experiment we will not compare the execution time of the planners[5] but only their plan quality. Table 1 shows the comparison between the three planners taking into account the makespan and the number of actions of each plan. We have not included the results for the `ferry` domain because the three planners generate the same fully sequential plans. However, plans for the `gripper` domain can have actions in parallel (the *gripper* is a resource with capacity for two balls) so that the planner can exploit a better concurrency. In order to simplify the resulting plans the actions have been assigned duration 1. Hence, the optimal makespan is $2n - 1$ and the number of actions in an optimal plan is $3n - 1$, where n is the number of balls in the problem. Surprisingly, both LPG or Mips generate sequential plans as can be easily noticed from the fact that the makespan comes directly from the number of actions in the plan. `tpsys-LC` is the only of the three planners which generates optimal plans for all the problems, highly exploiting the action parallelism.

In the **fourth experiment**, we have used the plans generated in the last IPC to perform a comparison between `tpsys-LC` and some of the state-of-the-art planners which participated in the simple-time track. We have analysed

[1]The experiments were run on a Pentium IV 2 GHz with 512 Mb.

[2]More information on the domains, problems and results of the international planning competition (IPC–2002) in: `http://www.dur.ac.uk/d.p.long/IPC`

[3]LPG is based on a non-deterministic local search and, consequently, it would not be fair to work with the plan from only one execution. In our experiments with LPG, we have run each problem ten times and extracted the average values. We have used the LPG1.0 version as provided in: `http://prometeo.ing.unibs.it/lpg`

[4]We have used the Mips version as provided in: `http://www.informatik.uni-freiburg.de/~mmips`

[5]LPG and Mips are clearly faster than `tpsys-LC`. For instance, they get to solve some simple problems before `tpsys-LC` finishes its 1st and 2nd stages. Currently, these stages represent an important bottleneck in our implementation and we think they could be drastically reduced.

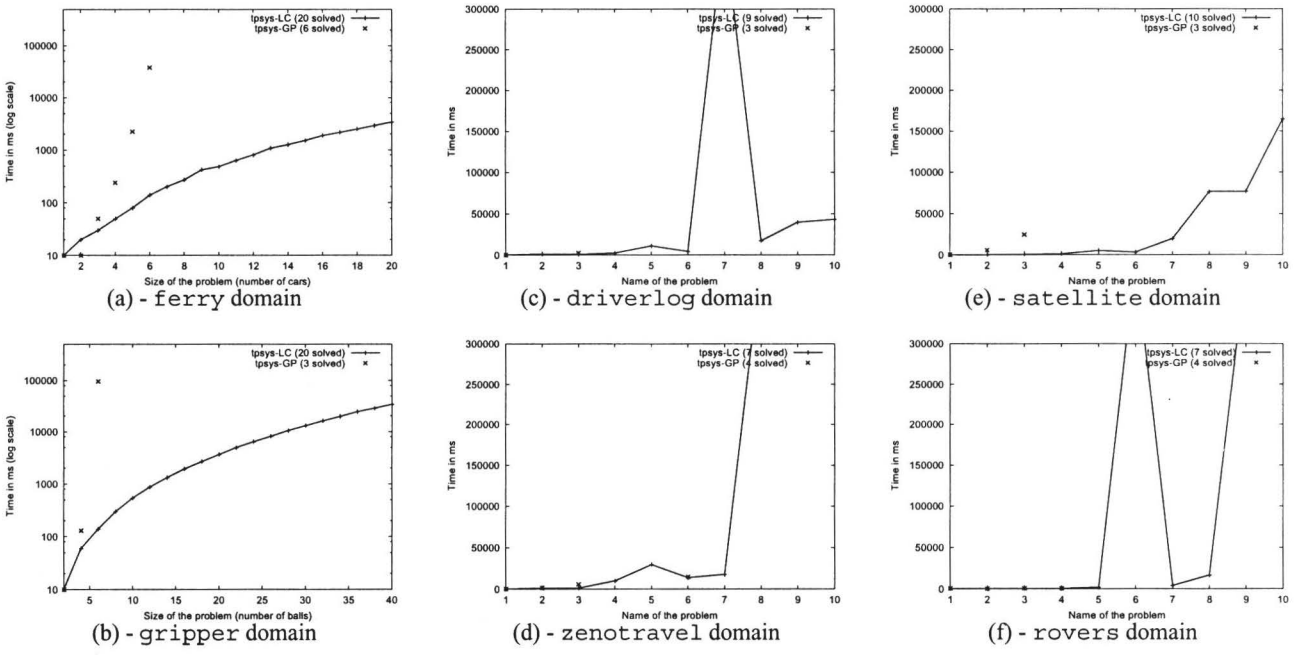


Figure 2: Comparison of the least-commitment search *vs.* the Graphplan backward search. Tests were censored after 300 s.

Problem	tpsys-LC	LPG	Mips
gripper-4	7 (11)	13 (13)	15 (15)
gripper-8	15 (23)	25 (25)	31 (31)
gripper-12	23 (35)	45 (45)	47 (47)
gripper-16	31 (47)	53 (53)	63 (63)
gripper-20	39 (59)	63 (63)	79 (79)
gripper-24	47 (71)	77 (77)	95 (95)
gripper-28	55 (83)	89 (89)	111 (111)
gripper-32	63 (95)	105 (105)	127 (127)
gripper-36	71 (107)	117 (117)	143 (143)
gripper-40	79 (119)	127 (127)	159 (159)

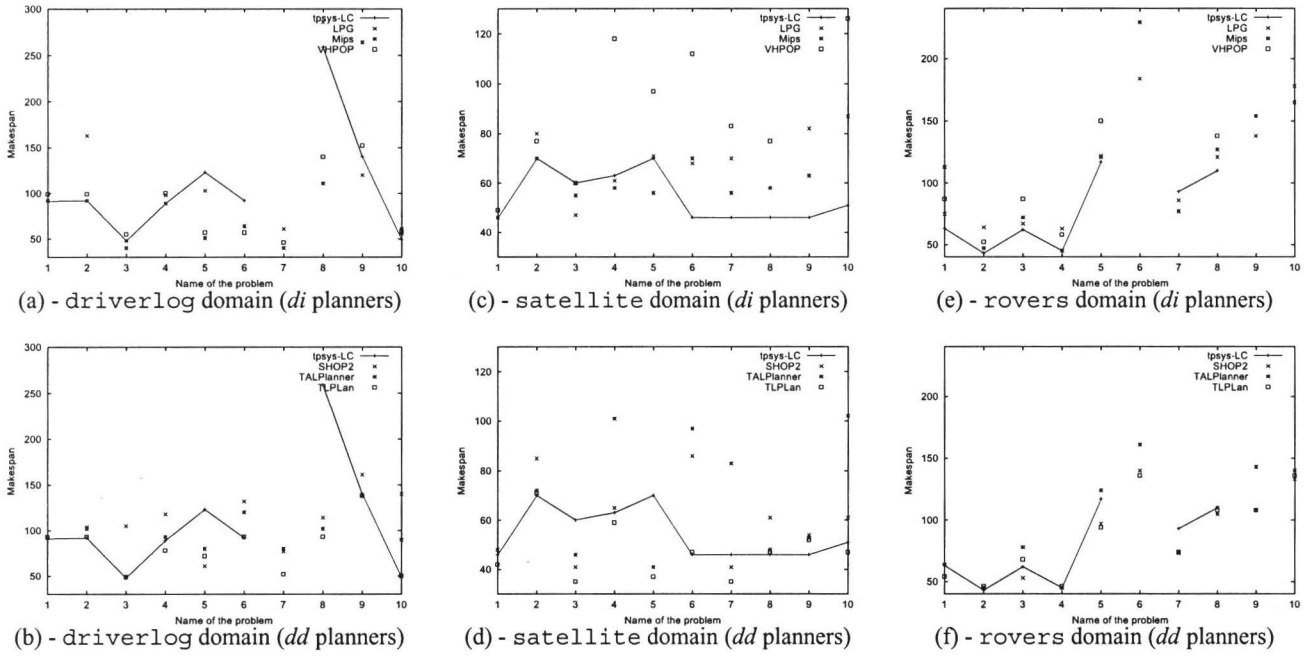
Table 1: Comparison of the makespan (number of actions are in brackets) of the plans for the gripper domain obtained by tpsys-LC, LPG and Mips planners.

the plans generated by the domain-independent (*di*) planners LPG, Mips and VHPOP, and the domain-dependent (*dd*) planners SHOP2, TALPlanner and TLPlan. For LPG and Mips we have run the problems again and for the rest of the planners we have used the results of the competition. Although Sapa, TP4 and IxTeT also participated in the competition we have not found enough results for the simple-time track to be considered in our comparison. Figure 3 shows the results of this comparison (for lack of space we only include the domains with more problems solved). In the driverlog domain, the plans of tpsys-LC are generally longer than the rest of planners with the exception of LPG and SHOP2. In the satellite domain, tpsys-LC behaves in average better than the rest of the planners, with the only exception of TLPlan. In the rovers domain, tpsys-LC generates again plans of very good quality, which are only improved by SHOP2 and TLPlan. Further, we have noticed that tpsys-LC generates plans with nearly the same (or even fewer) actions than the rest of the planners. These experimental results show that the new approach significantly improves the scalability of the Graphplan backward search.

5 Conclusions through Related Work

Least-commitment techniques have been widely used in planning, relying on the consistency of temporal constraints (IxTeT and HSTS), postponing the assignment of values to variables or the order of execution of actions, etc. [Weld, 1994]. We use a least-commitment approach to overcome the limitations of the Graphplan backward search detected in previous works [Fox and Long, 1999; Zimmerman and Kambhampati, 1999]. Our approach basically postpones the allocation in time of actions until they become not mutex and applicable. Thus, the algorithm generates a relaxed plan similarly to FF, to be used as a *skeleton* of the plan. Next, it allocates actions in time according to their mutex relations and to several local heuristic criteria in the line of LPG. Our critical difference with LPG relies on several points. First, the planning graph is temporal and moves chronologically in time instead of planning steps. Second, the *TG* is not only used to extract heuristic information but also to generate a relaxed plan. Third, this plan is repaired in a forward chaining direction, and unlike LPG the allocated actions are never removed. Fourth, the heuristics exploit better the structure of the plan and the real importance of each action in the plan. Finally, tpsys-LC uses a more precise model of action mutex which implies fewer constraints on the execution of actions and a larger search space.

This paper contributes in the way in which least-



(a) - driverlog domain (*di* planners) (c) - satellite domain (*di* planners) (e) - rovers domain (*di* planners)

(b) - driverlog domain (*dd* planners) (d) - satellite domain (*dd* planners) (f) - rovers domain (*dd* planners)

Figure 3: Comparison of the quality of the plans obtained by tpsys-LC and some state-of-the-art planners.

commitment can be applied in a Graphplan-based temporal (or classical) approach, substituting the backward search by the new two-stage search. The new search combines the information calculated in the planning graph with a heuristic, greedy search process, and increases the planner scalability. The advantage is that the planner can exploit a high level of action concurrency, what leads to plans highly competitive with other state-of-the-art planners under a deterministic approach. Although in temporal planning the most important criterion of quality is the makespan, the experimental results show that the new search generates plans in which the number of actions is quite good (even in problems of IPC–1998 and IPC–2000). The main disadvantage of this approach is that it is not complete preserving. However, although completeness and optimality are desired properties, guaranteeing them entails a huge complexity, preventing planners from producing plans with more than a few actions. The algorithm has still some limitations and our future work is to refine the heuristic functions to improve the quality of the plans, the performance and to avoid some of the inconveniences of the greedy search.

Acknowledgments

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

References

[Blum and Furst, 1997] A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[Edelkamp, 2002] S. Edelkamp. Mixed propositional and numeric planning in the model checking integrated planning system. In *Proc. Workshop on Planning for Temporal Domains (AIPS-2002)*, pages 47–55, 2002.

[Fox and Long, 1999] M. Fox and D. Long. The detection and exploitation of symmetry in planning domains. In *Proc. IJCAI-99*, pages 956–961, 1999.

[Fox and Long, 2001] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK, 2001.

[Garrido *et al.*, 2002] A. Garrido, M. Fox, and D. Long. A temporal planning system for durative actions of PDDL2.1. In *Proc. ECAI-2002*, pages 586–590, 2002.

[Gerevini and Serina, 2002] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs with action costs. In *Proc. AIPS-2002*, pages 281–290. 2002.

[Hoffmann, 2000] J. Hoffmann. 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.

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

[Weld, 1994] D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):93–123, 1994.

[Zimmerman and Kambhampati, 1999] T. Zimmerman and S. Kambhampati. Exploiting symmetry in the planning-graph via explanation-guided search. In *Proc. AAAI-99*, 1999.