DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Informe Técnico / Technical Report

Ref. No.: DSIC-II/09/04

Pages: 58

Title: Domain Independent Temporal Planning in a Planning-Graph-Based Approach

Author(s): Antonio Garrido & Eva Onaindía

Date: 13-05-2004

Keywords: temporal planning, graphplan planning, heuristics, least-commitment search

V^o B^o
Leader of research Group

Author(s)

Domain Independent Temporal Planning in a Planning-Graph-Based Approach

Antonio Garrido AGARRIDOT@DSIC.UPV.ES
Eva Onaindía ONAINDIA@DSIC.UPV.ES
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46071, Valencia, Spain

Abstract

Many planning domains have to deal with temporal features that can be expressed using durations that are associated to actions. Unfortunately, the conservative model of actions used in many existing temporal planners is not adequate for domains which require more expressive models. This paper presents a temporal planning approach that combines the principles of Graphplan and TGP and uses the information calculated in the planning graph to deal with a non-conservative model of actions that include local conditions and effects. In this approach, we propose two strategies for search. The first one is based on the Graphplan backward search. The second one is based on a least-commitment and heuristic search, and it attempts to overcome the main limitations of a chronological backtracking search when dealing with large temporal problems. This search has proved to be beneficial in the scalability of the planner and the experiments show that a planner using this new search is competitive with other state-of-the-art planners *w.r.t.* the plan quality[1].

1. Introduction

There has been an impressive growth in the use of AI Planning algorithms in the last decade (Blum & Furst, 1997; Bonet & Geffner, 2001; Long & Fox, 2002; Weld, 1999). Many problems which were unsolvable a few years ago are now easily solved by current state-of-the-art planners. However, classical planning assumes some simplifications which are not always acceptable in real world planning problems. These simplifications prevent the planners from dealing with more realistic features, such as temporal and metric capabilities, explicit management of resources, more expressive domain definition languages, problems with uncertainty, optimisation criteria, etc. (Do & Kambhampati, 2001; Fox & Long, 2001; Garrido et al., 2002; Gerevini & Serina, 2002a; Haslum & Geffner, 2001; Hoffmann, 2002; Smith & Weld, 1999). Solving these simplifications imposes challenges on current planning research in order to extend and improve the functionalities of planners.

This paper deals with three of the previous functionalities: i) planning with temporal features (actions with duration), ii) more expressive domain definition languages (non-conservative model of actions of PDDL2.1, Fox & Long, 2001), and iii) plan optimisation (makespan). On one hand, it is clear that real planning problems that deal with time cannot assume that actions have the same duration (Garrido, Onaindía, & Barber, 2001; Smith & Weld, 1999). For instance, in a logistics domain, the action `fly(plane,London,Moscow)` is

1. This article is an extended and revised version of the papers (Garrido, Fox, & Long, 2002; Garrido & Onaindía, 2003) that have been published at ECAI–2002 and IJCAI–2003, respectively.

longer than `fly(plane,London,Paris)`. Discarding the assumption that actions have the same duration implies a more complex problem because the number of possible execution times for actions is vastly increased. On the other hand, temporal planners appearing in the recent literature, such as parcPLAN, TGP, TP4 or LPG1.0 (El-Kholy & Richards, 1996; Smith & Weld, 1999; Haslum & Geffner, 2001; Gerevini & Serina, 2002a) have had some success in dealing with actions with duration. However, these planners have adopted a conservative model of actions which is a modest extension of the one used by non-temporal planners. This means that two actions cannot overlap in *any way* if they have conflicting preconditions or effects. This makes it possible to produce reasonable plans in some planning domains, but there exist other domains that require a richer model of actions and in which better quality plans can be found. The new version of PDDL (McDermott, 1998), called PDDL2.1 (Fox & Long, 2001), provides a level 3 with a more permissive (non-conservative) model of durative actions which subsumes the conservative model of actions. Level 3 includes local conditions/effects and allows actions to overlap even when their preconditions or effects refer to the same propositions, achieving a more accurate exploitation of action concurrency in which better quality (shorter makespan) plans can be found. Consequently, guaranteeing the plan that minimises the global makespan is an important issue in temporal planning, and the application of heuristics to find good plans is becoming more and more important.

In this paper, we describe our experiences with a Temporal Planning SYStem (from now on TPSYS, Garrido et al., 2001, 2002; Garrido & Onaindía, 2003) to manage the non-conservative model of durative actions provided in level 3 of PDDL2.1. This paper is based on previous works on temporal planning and tries to integrate them into one temporal planning approach based on Graphplan (Blum & Furst, 1997). First, we discuss how actions with duration can be handled in a Graphplan approach (Garrido et al., 2001). Technically, this involves extending the mutual exclusion reasoning as presented in TGP (Smith & Weld, 1999). Second, we describe the way to extend and modify the Graphplan algorithm to deal with a non-conservative model of actions (Garrido et al., 2002). Third, we propose a new search method based on least-commitment and heuristic search to improve the performance of the planning algorithm and overcome the main inefficiencies of the Graphplan backward search. These inefficiencies are especially relevant in temporal settings (Garrido & Onaindía, 2003). Hence, the main contributions of this paper are:

- An analysis of how non-conservative durative actions (level 3 of PDDL2.1) can be managed in a Graphplan-based approach. A non-conservative model of actions implies fewer constraints on the execution of actions, providing more opportunities to the planner for selecting actions and finding shorter makespan plans.

- An extension of the analysis of mutual exclusion relations to include local conditions and effects based on the work of TGP. TPSYS introduces a mutex classification into static and dynamic mutex (action/action, proposition/action and proposition/proposition, Garrido et al., 2001, 2002), which is now more precise and provides new ways of overlapping actions.

- An explanation of how a temporal planning graph can be generated, without `no-op` actions. The extension of this temporal graph now contains some subtle details due to the local conditions and effects of non-conservative actions.

- A description of two (optimal and non-optimal) search approaches and the way they obtain a temporal plan. Consequently, both approaches can solve problems of the type *"obtain a plan of makespan that is shorter than a given value $\mathcal{D}_{max}$"*.
  - The first approach guarantees the properties of completeness and optimality *w.r.t.* makespan and follows the same strategy as Graphplan, extracting a plan through the temporal graph. However, the traditional *directionality* of Graphplan is broken due to the combination of local effects and conditions of the actions.
  - The second approach performs a search stage based on least-commitment and heuristic techniques, but it is neither complete nor optimal-preserving. This approach is motivated by the inefficiencies of the first approach and solves them by means of an incremental generation of plans: the algorithm generates a relaxed plan from the temporal graph which is used as a *skeleton* of the final plan. New actions are then heuristically selected and inserted into the plan to support the unsolved (sub)goals. This way the search is better guided and its behaviour is more scalable.
- Some experimental results showing the comparison between the two search approaches and the comparison of the least-commitment approach with other state-of-the-art temporal planners.

This paper is organised as follows. In the next section, we present the motivation for dealing with a non-conservative model of actions. In section 3, we introduce the action model and terminology used throughout the rest of the paper, presenting some of the difficulties that local conditions and effects may have on planning. Section 4 introduces TPSYS, its three stage structure, and the problem formalisation. The first stage of TPSYS is presented in section 5, along with the definition of static mutex. The second stage is presented in section 6. This section describes the formalisation of dynamic mutex and the extension of the temporal planning graph. Section 7 presents the search of a temporal plan and analyses two search approaches. The former is based on Graphplan and also discusses several methods for improving the search, such as temporal memoization and heuristic search to reduce the search space. The latter is based on least-commitment and heuristic techniques; it reviews the main inefficiencies of the Graphplan backward search and proposes a new search approach to overcome these inefficiencies. Section 8 includes the experimental results of the two search approaches in some benchmark problems used in the International Planning Competitions and makes some comparisons with state-of-the-art temporal planners. In section 9, we discuss two alternatives for transforming non-conservative actions into conservative actions to be *directly* used in classical planners. Finally, section 10 presents our conclusions through related work as well as some directions for future work.

2. Motivation for a Non-Conservative Model of Actions

In real problems, the motivation for dealing with actions with duration is clear: actions usually have widely different durations (Smith & Weld, 1999). Although the motivation for dealing with a non-conservative model of actions is also clearly justified, it requires a more detailed analysis. Unlike conservative models of actions in PDDL, non-conservative

actions of PDDL2.1 allow the modelling of temporal planning domains to achieve a fuller exploitation of concurrency. PDDL2.1 extends PDDL to include not only duration of actions, but also local conditions and effects. This involves a more precise modelling of the state transitions undergone by different propositions within the durative interval of the action. In particular, the traditional preconditions of the starting point of the action need not necessarily be maintained throughout the interval. There may be *pre*conditions of the final effect of the action that can be achieved concurrently with the action rather than maintained throughout the duration of the action. Hence, it becomes necessary to distinguish invariant from non-invariant conditions depending on whether they can be affected during the interval of execution or not. Furthermore, there might be initial effects at the starting point that can be exploited by concurrent actions. All these distinctions give rise to quite sophisticated opportunities for concurrent actions in a PDDL2.1 plan.

Let us consider the conservative action `fly(plane,origin,destination)`. This action requires the proposition `at(plane,origin)` to be true before executing the action, and asserts the propositions ¬`at(plane,origin)` and `at(plane,destination)` at the end of the action. This implies that the location of the `plane` is inaccessible until the end of the action, preventing concurrent actions (for instance, those that require the `plane` not to be in the origin) from being executed in parallel with `fly(plane,origin,destination)`. As Fox and Long (2001) suggest, this may exclude many valid plans. In PDDL2.1, this situation can be avoided by simply asserting ¬`at(plane,origin)` as an initial effect and `at(plane,destination)` as a final effect. In addition, if we want to know that the `plane` is `flying` during the action `fly`, it would be enough to assert the proposition `flying(plane)` as an initial effect and ¬`flying(plane)` as a final effect. In the conservative model, the action equivalent to this `fly` action would not represent the fact of being `flying` due to the inability to express the proposition `flying(plane)` and ¬`flying(plane)` as initial and final effects, respectively. Therefore, it is impossible to work with more realistic actions that require this proposition, such as a possible `refuel-during-flight` action. Although this limitation could be overcome in a conservative model of actions by splitting each non-conservative action into three conservative actions, this makes the size of the problem larger as we will discuss in section 9.

Even though in real problems instantaneous actions are never really *instantaneous*, there are some cases in which these actions could be useful for modelling purposes. Level 3 of PDDL2.1 also allows the definition of these actions, i.e. traditional actions with no duration. Since PDDL2.1 intends to provide *physics* instead of *advice* of the planning problem, instantaneous actions could be useful in order to obtain a valid plan for different executive agents when the duration of the action is very small (or even unknown) to be considered by the planning agent. More generally, the domain engineer might choose to model the domain at a level of abstraction at which it is not interesting to capture the durations of practically instantaneous actions. That is, the engineer might choose to emphasise the durations of some actions but not of others. These modelling choices do not lead to conflict with the semantics presented by Fox and Long (2001) because, at level 3 of PDDL2.1, it is possible to express an instantaneous action as an action with barely measurable duration. This duration is epsilon, an amount so small that it makes no sense to split it. This means that non-interfering actions that take epsilon time can happen in parallel but they cannot be interleaved. This epsilon is so small that it never changes the sequence of actions in the

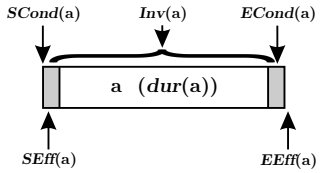


Figure 1: Components of a durative action.

plan. Epsilon has to be chosen appropriately for a given domain and problem, because it represents a discretisation of the time-line into indivisible units, the end points of which mark the points at which actions can be initiated or terminated.

3. Action Model and Terminology

Non-conservative durative actions of PDDL2.1 can require more conditions to be guaranteed for the success of the action than traditional actions of PDDL. Durative actions not only have effects that hold at their conclusions, but they also have effects that must be asserted immediately after the actions start.

Definition 1 (Components of a durative action in PDDL2.1) *Let* a *be a durative action which starts at time s and ends at time e, being executed throughout the interval* $[s..e]$ *(see Figure 1). The components of* a *are the following:*

- *Conditions. The three types of local conditions of a durative action are: i)* $SCond(\mathtt{a})$*, the set of conditions to be guaranteed at the start of the action (time s); ii)* $Inv(\mathtt{a})$*, the set of invariant conditions to be guaranteed over the execution of the action (time* $[s..e]$*); and iii)* $ECond(\mathtt{a})$*, the set of conditions to be guaranteed at the end of the action (time e).*

- *Duration. The duration of the action is a positive value represented by* $dur(\mathtt{a}) \in \mathcal{R}^+$.

- *Effects. The two types of effects of a durative action are: i)* $SEff(\mathtt{a}) = \{SAdd(\mathtt{a}) \cup SDel(\mathtt{a})\}$*, with the positive and negative effects respectively to be asserted at the start of the action (time s); and ii)* $EEff(\mathtt{a}) = \{EAdd(\mathtt{a}) \cup EDel(\mathtt{a})\}$*, with the positive and negative effects respectively to be asserted at the end of the action (time e).*

Durative actions involve other difficulty: there exist some effects ($SEff(\mathtt{a})$) which can be obtained before the action a ends. Hence, it might occur that an initiated action could not end because its end conditions ($ECond(\mathtt{a})$) are not satisfied in the future. Because PDDL2.1 semantics guarantees the entire execution of an action, in this case, all the start effects (and the actions which are dependent on them) should be invalidated. We call these kinds of actions *conditional actions* because their execution is provisional and it is dependent on the fulfillment of the end conditions.

Definition 2 (Conditional action) *An action* $\mathtt{a}$ *is a conditional action iff* $(SEff(\mathtt{a}) \neq \emptyset) \wedge (ECond(\mathtt{a}) \neq \emptyset)$ *holds. This way, the set of propositions* $SEff(\mathtt{a})$ *of a conditional action* $\mathtt{a}$ *only becomes valid when all propositions in* $ECond(\mathtt{a})$ *are satisfied.*

Conditional actions usually occur in domains where some effects of durative actions are achieved when actions start, required throughout the duration of execution of that action and probably deleted when the action ends. Such initial effects cannot be exploited as end effects because they do not persist beyond the end of the action. Furthermore, the successful termination of a durative action (end conditions) must be confirmed even if a goal is achieved before the end of its durative interval. This is because durative actions *promise* to terminate initiated actions in a stable state. If anything in the plan prevents this stable termination, then the plan must be considered invalid. Richer specifications might allow one to consider exogenous events (Smith & Weld, 1999) and conditions/effects which come into play at a specific point in time and must persist only over finitely bounded intervals (Do & Kambhampati, 2001). However, PDDL2.1 does not yet support this.

Definition 3 (Conditional proposition) *A proposition* $\mathtt{p}$ *is conditional iff all the actions* $\{\mathtt{a_i}\}$ *which achieve* $\mathtt{p}$ *are conditional and they have not yet ended their execution.*

Intuitively, if $\mathtt{p}$ is only achieved by conditional actions $\{\mathtt{a_i}\}$, $\mathtt{p}$ will be conditional until at least one action $\mathtt{a_i}$ ends successfully, which implies both $SCond(\mathtt{a_i})$ and $ECond(\mathtt{a_i})$ are satisfied. Once this happens $\mathtt{p}$ is valid (no longer conditional).

As we have seen in the previous section, instantaneous actions are allowed in level 3 of PDDL2.1. This does not represent a serious inconvenience because a straightforward correspondence rule can transform an instantaneous action into a durative action. This way, all the instantaneous actions present in the planning domain can be managed in the same way as durative actions.

Definition 4 (Correspondence rule $\mathcal{R}_{\mathtt{a_i} \mapsto \mathtt{a_d}}$) *The correspondence rule maps an instantaneous action* $\mathtt{a_i}$*, with* $Pre(\mathtt{a_i})$*,* $Effs(\mathtt{a_i}) = \{Add(\mathtt{a_i}) \cup Del(\mathtt{a_i})\}$ *into a durative action* $\mathtt{a_d}$ *in the following way:*

$$\begin{gathered} SCond(\mathtt{a_d}) = ECond(\mathtt{a_d}) = Inv(\mathtt{a_d}) = Pre(\mathtt{a_i}) \\ SAdd(\mathtt{a_d}) = EAdd(\mathtt{a_d}) = Add(\mathtt{a_i}) \\ SDel(\mathtt{a_d}) = EDel(\mathtt{a_d}) = Del(\mathtt{a_i}) \\ dur(\mathtt{a_d}) = 0 \end{gathered}$$

An example of the definition of durative actions of PDDL2.1 is depicted in Figure 2. This figure represents some actions of the `satellite` domain used in the IPC–2002[2]. According to Definition 1, the actions have *at start*, *over all* and *at end* conditions with the conditions to be satisfied right at the beginning of the action, during its execution and at the end of

2. More information on the domains and problems of the International Planning Competition 2002 in: `http://www.dur.ac.uk/d.p.long/IPC`

```
(:durative-action switch_on
 :parameters (?i - instrument ?s - satellite)
 :duration (= ?duration 2)
 :condition (and (at start (power_avail ?s))
                 (over all (on_board ?i ?s)))
 :effect (and (at start (not (calibrated ?i)))
              (at start (not (power_avail ?s)))
              (at end (power_on ?i))))

(:durative-action calibrate
 :parameters (?s - satellite ?i - instrument ?d - direction)
 :duration (= ?duration 5)
 :condition (and (at start (pointing ?s ?d))
                 (over all (on_board ?i ?s))
                 (over all (calibration_target ?i ?d))
                 (over all (power_on ?i)))
                 (at end (power_on ?i)))
 :effect (at end (calibrated ?i)))

(:durative-action turn_to
 :parameters (?s - satellite ?d_new - direction ?d_prev - direction)
 :duration (= ?duration 10)
 :condition (and (at start (pointing ?s ?d_prev))
                 (over all (not (= ?d_new ?d_prev))))
 :effect (and (at start (not (pointing ?s ?d_prev)))
              (at end (pointing ?s ?d_new))))

(:durative-action take_image
 :parameters (?s - satellite ?d - direction ?i - instrument ?m - mode)
 :duration (= ?duration 1)
 :condition (and (over all (calibrated ?i))
                 (over all (on_board ?i ?s))
                 (over all (supports ?i ?m))
                 (over all (power_on ?i))
                 (over all (pointing ?s ?d))
                 (at end (power_on ?i)))
 :effect (at end (have_image ?d ?m)))
```

Figure 2: Definition of some durative actions for the `satellite` domain. This domain is inspired by space-applications, which involve planning and scheduling a collection of observation tasks between multiple satellites.

the action, respectively. Analogously, the *at start* and *at end* effects are the effects to be asserted at the beginning and the end of the execution of the action.

At first glance, an extension of a Graphplan-based planner to deal directly with level 3 durative actions would seem quite easy. However, it implies important changes in the way the temporal graph is generated and in the way the search for a plan is performed. All these new requirements are presented in section 4.

4. TPSYS: a Temporal Planning SYStem

TPSYS copes with temporal planning problems under a non-conservative model of actions by combining the ideas of Graphplan (Blum & Furst, 1997) and TGP (Smith & Weld, 1999). This means that TPSYS incrementally extends a temporal planning graph, performs a backward search through that graph and extracts a feasible, optimal temporal plan.

4.1 Problem Formalisation

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 state as two sets of propositions, respectively. $\mathcal{A}$ represents the set of ground durative actions in the domain. Unlike conservative actions, durative actions present more conditions to be satisfied in order to guarantee the success of the action ($SCond(\mathtt{a})$, $Inv(\mathtt{a})$ and $ECond(\mathtt{a})$). In addition, durative actions have two types of local effects ($SEff(\mathtt{a})$ and $EEff(\mathtt{a})$). Finally, $\mathcal{D}_{max}$ stands for the maximum duration allowed by the user. Time is modelled by $\mathcal{R}^+$ and their chronological order. Although PDDL2.1 can define different metrics to be used as optimisation criteria, TPSYS always tries to minimise the makespan.

4.2 Structure of TPSYS

TPSYS consists of three consecutive stages, as indicated in Figure 3. Briefly, these three stages are:

- The first stage. This stage performs a reachability analysis to discard as much irrelevant information (propositions and actions) as possible. This simplifies the planning process because it reduces the real size of the problem. It then calculates the static mutex relations, i.e. the mutex which always hold because they only depend on the problem definition. This stage is not present in either Graphplan or TGP, and it is only executed at the beginning of the process.

- The second stage. This stage incrementally extends a temporal planning graph, alternating proposition and action levels. All the information about action/action, proposition/action and proposition/proposition dynamic mutex is propagated through the temporal graph. The extension continues until all the problem goals are non-pairwise mutex and the actions which support them have successfully terminated.

- The third stage. This stage extracts a temporal plan, as an acyclic flow of actions, through the temporal planning graph. TPSYS implements two different approaches for the search. In the search based on Graphplan (see section 7.1), the second and

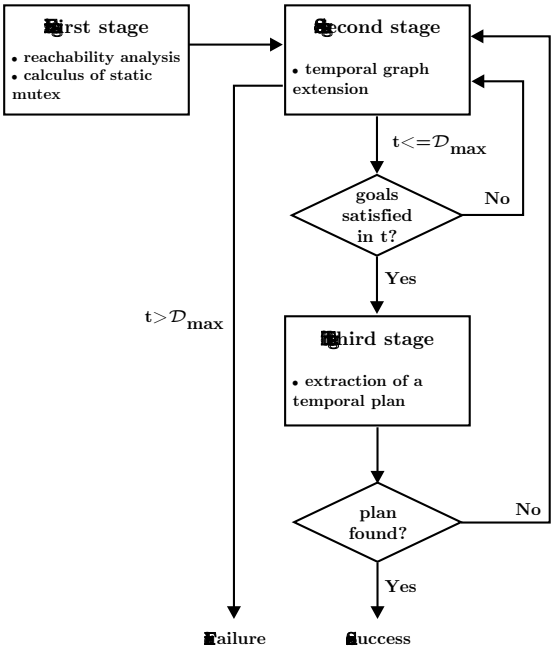


Figure 3: Structure of TPSYS.

third stages are executed in an interleaved way until either a plan is found (success) or the makespan of the plan exceeds the value $\mathcal{D}_{max}$ (failure). In contrast, in the search based on least-commitment and heuristic search (see section 7.2), the second and third stages are not executed in an interleaved way.

5. First Stage. Calculus and Analysis of Static Mutex

The first stage of TPSYS performs two important tasks: i) the operator instantiation from the domain operators, and ii) the calculus of the static mutex, which always hold. The more permissive model of actions of PDDL2.1 makes the calculus of mutex significantly more expensive. Local conditions and effects allow the simultaneous execution of actions, even when those actions refer to the same propositions. For instance, two actions with interfering propositions will never be executed in parallel in a conservative model. However, in a non-conservative model, those actions may not start or end together, but they can still be executed in parallel. Therefore, the additional cost in the calculus of mutex requires an exhaustive analysis (and filtering) of the actions to be instantiated in order to discard any irrelevant action in the planning process.

5.1 Reachability analysis

Complex domains include many operators which are necessary in real planning problems. However, such operators are not always reachable or useful in all the problem instances

for that domain. For instance, let us suppose the operator `fly(?p,?c1,?c2)` that requires that both city `?c1` and `?c2` have an airport for the plane `?p` to fly to and from. Obviously, if `city1` does not have an airport, any action `fly` with `city1` as origin or destination will never be part of a feasible plan, and the only consequence of that action will be to degrade the planner's performance. Although detecting such a situation seems a trivial task, its generalisation to domain-independent planning is not easy. In addition, other situations that arise indirectly are more difficult to detect. If no plane `?p` can be used in `city1`, any action for loading (`load(?package,?p,city1)`) or unloading (`unload(?package,?p,city1)`) will be irrelevant, no matter what the values of `?package` and `?p` are. Furthermore, there are many actions that are applicable but have no influence while calculating the plan. For instance, if the problem goal consists of transporting `package1`, any action applicable to the rest of packages is irrelevant and its analysis would degrade the planner's performance.

Reachability analysis techniques to discard irrelevant information in planning processes are not new and have been studied in recent literature (Fox & Long, 1998; Nebel, Dimopoulos, & Köehler, 1997). One of the most outstanding works has been presented by Nebel et al. (1997) in the IPP planner. This work combines a backward and a forward search process and discards information which is likely to be unnecessary in the plan. Nebel et al. use non-admissible heuristics to remove as much information (facts and operators) as possible. However, in some cases this leads to discarding information that is necessary for the planning process. Detecting the information that is strictly necessary for a plan tends to be as difficult as generating the entire plan. Therefore, we use an intermediate approach to discard irrelevant information in an efficient, solution-preserving way (i.e. our reachability analysis does not remove information that is necessary for a feasible plan).

Figure 4 describes the algorithm for the reachability analysis performed in the first stage of TPSYS. The algorithm does not use any estimation or heuristic function as in (Nebel et al., 1997). However, there is a specific distinction between the information (propositions and actions) that is reachable from the initial state and information that is reachable from the final state. The algorithm consists of two simple loops: i) forward from $\mathcal{I}_s$ (steps 1–9), and ii) backward from $\mathcal{F}_s$ (steps 10–21). Finally, the algorithm marks the propositions and actions reachable in both directions (steps 22–24) as globally reachable discarding all the information that is irrelevant to the planning process.

The reachability analysis performed in the algorithm of Figure 4 reduces the amount of irrelevant information in the problem, solving the problems introduced above. Although this algorithm does not remove all the irrelevant information, the advantages are important: the actions that will not be part of a feasible plan are removed, as well as the static, artificial propositions which are only necessary for the instantiation of the actions (Fox & Long, 1998). For instance, the fact that there is an airport in a city, a link between two cities, etc. is useful for checking whether an action can be instantiated, but not during the planning process. This reachability analysis can also detect many unsolvable problems when the problem goals are not reachable in both directions. This analysis also provides an initial estimation of the cost (in terms of actions or time) to achieve either a proposition or an action. This information can be very valuable for the heuristic estimations used in modern planning approaches.

1. *//Reachability analysis from* $\mathcal{I}_s$
2. $\texttt{reach\_props\_IS} \leftarrow \mathcal{I}_s$
3. $\texttt{reach\_acts\_IS} \leftarrow \emptyset$
4. **repeat**
5. **forall** $\texttt{a}_\texttt{i} \mid \{SCond(\texttt{a}_\texttt{i}) \cup Inv(\texttt{a}_\texttt{i})\} \subseteq \texttt{reach\_props\_IS}$
6. **if** $\texttt{a}_\texttt{i} \notin \texttt{reach\_acts\_IS}$ **then**
7. $\texttt{reach\_acts\_IS} \leftarrow \texttt{reach\_acts\_IS} \cup \{\texttt{a}_\texttt{i}\}$
8. $\texttt{reach\_props\_IS} \leftarrow \texttt{reach\_props\_IS} \cup \{SAdd(\texttt{a}_\texttt{i}) \cup EAdd(\texttt{a}_\texttt{i})\}$
9. **until** reach_acts_IS are not modified in the iteration
10. *//Reachability analysis from* $\mathcal{F}_s$
11. $\texttt{reach\_props\_FS} \leftarrow \mathcal{F}_s$
12. $\texttt{reach\_acts\_FS} \leftarrow \emptyset$
13. $\texttt{goals} \leftarrow \mathcal{F}_s$
14. **while** $\texttt{goals} \neq \emptyset$
15. **forall** $\texttt{g}_\texttt{i} \in \texttt{goals}$
16. $\texttt{goals} \leftarrow \texttt{goals} \setminus \{\texttt{g}_\texttt{i}\}$
17. **forall** $\texttt{a}_\texttt{i} \mid (\texttt{g}_\texttt{i} \in \{SAdd(\texttt{a}_\texttt{i}) \cup EAdd(\texttt{a}_\texttt{i})\} \wedge \texttt{a}_\texttt{i} \in \texttt{reach\_acts\_IS})$
18. $\texttt{reach\_props\_FS} \leftarrow \texttt{reach\_props\_FS} \cup \{\texttt{g}_\texttt{i}\}$
19. **if** $\texttt{a}_\texttt{i} \notin \texttt{reach\_acts\_FS}$ **then**
20. $\texttt{reach\_acts\_FS} \leftarrow \texttt{reach\_acts\_FS} \cup \{\texttt{a}_\texttt{i}\}$
21. $\texttt{goals} \leftarrow \texttt{goals} \cup \{SCond(\texttt{a}_\texttt{i}) \cup Inv(\texttt{a}_\texttt{i}) \cup ECond(\texttt{a}_\texttt{i})\}$
22. *//Global reachability analysis (both directions)*
23. $\texttt{reach\_props} \leftarrow \texttt{reach\_props\_IS} \cap \texttt{reach\_props\_FS}$
24. $\texttt{reach\_acts} \leftarrow \texttt{reach\_acts\_IS} \cap \texttt{reach\_acts\_FS}$

Figure 4: Algorithm for the reachability analysis.

5.2 Definition of static mutex in a non-conservative model of actions

Most Graphplan-based planners do not establish any difference between the nature of the mutex relations, calculating the same mutex repetitively during the generation of the planning graph (Halsey, 2002). However, in most problems, there is a set of mutex that always hold. For instance, it is clear that `fly(plane,city1,city2)` and `fly(plane,city2,city1)` using the same `plane` will be always mutex and they will never be executed in parallel. TGP first introduced a classification of mutex as *eternal* and *conditional* mutex in a Graphplan approach. In the same way, TPSYS performs a classification of mutex as static and dynamic mutex. Static mutex never change and are time-independent. These mutex are static because they only depend on the operators of the planning domain. Therefore, it is not necessary to postpone their calculus to the extension of the temporal planning graph. This also allows us to speed up the rest of the stages in the planning process.

When handling non-conservative actions, the mutex calculus must distinguish between different ways of overlapping actions. The conflicts which prevent actions from being executed in parallel are due to the action synchronisation and the conditions/effects which are contradictory (Fox & Long, 2001; Long & Fox, 2001). Non-conservative actions have more conditions and effects than conservative actions, so it is obvious that new types of mutex must be introduced.

Definition 5 (Static action/action mutex (*AA mutex*)) *Non-conservative actions have four types of static mutex, (see Table 1):*

1. *Case 1 ($AA_{start-start}$) represents the situation when two actions cannot start together because: i) initial effects are contradictory, or ii) initial effects and conditions are contradictory.*

2. *Case 2 ($AA_{end-end}$) represents the situation when two actions cannot end together because: i) final effects are contradictory, or ii) end effects and conditions are contradictory*[3]*.*

3. *Case 3 ($AA_{end-start}$) represents the situation when one action cannot end when another action starts because the final effects of the former are contradictory with the conditions/effects of the latter. This mutex (which does not appear in Graphplan) might seem to be a stronger requirement than is really necessary, but it takes into account the fact that simultaneity can never be relied upon in the real world —it cannot be guaranteed that the action requiring the at start condition will definitely happen after the achievement of that condition at execution time*[4]*.*

4. *Case 4 ($AA_{during-during}$) represents the situation when one action cannot start/end during the execution of another action because the effects of the former are contradictory with the invariant conditions of the latter.*

3. Clearly, when two actions have the same duration, $AA_{end-end}$ mutex implies $AA_{start-start}$ mutex, and vice versa.
4. In order to simplify the calculus of this case, TPSYS takes the correctness-preserving assumption of including an epsilon ($\epsilon > 0$) between the action which ends and the action which starts, thus solving the problem of simultaneity. The value of ϵ is so small that it does not negatively affect the soundness of the algorithm.

Case	Condition for the mutex	Type of mutex	Relation
1	$(SAdd(\mathtt{a}) \cap SDel(\mathtt{b}) \neq \emptyset) \vee (SAdd(\mathtt{b}) \cap SDel(\mathtt{a}) \neq \emptyset)$ $((SAdd(\mathtt{a}) \cup SDel(\mathtt{a})) \cap (SCond(\mathtt{b}) \cup Inv(\mathtt{b})) \neq \emptyset)$ $((SAdd(\mathtt{b}) \cup SDel(\mathtt{b})) \cap (SCond(\mathtt{a}) \cup Inv(\mathtt{a})) \neq \emptyset)$	$AA_{start-start}$	
2	$(EAdd(\mathtt{a}) \cap EDel(\mathtt{b}) \neq \emptyset) \vee (EAdd(\mathtt{b}) \cap EDel(\mathtt{a}) \neq \emptyset)$ $((EAdd(\mathtt{a}) \cup EDel(\mathtt{a})) \cap (ECond(\mathtt{b}) \cup Inv(\mathtt{b})) \neq \emptyset)$ $((EAdd(\mathtt{b}) \cup EDel(\mathtt{b})) \cap (ECond(\mathtt{a}) \cup Inv(\mathtt{a})) \neq \emptyset)$	$AA_{end-end}$	
3	$((EAdd(\mathtt{a}) \cup EDel(\mathtt{a})) \cap (SCond(\mathtt{b}) \cup Inv(\mathtt{b})) \neq \emptyset)$ $((EAdd(\mathtt{b}) \cup EDel(\mathtt{b})) \cap (SCond(\mathtt{a}) \cup Inv(\mathtt{a})) \neq \emptyset)$ $(EAdd(\mathtt{a}) \cap SDel(\mathtt{b}) \neq \emptyset) \vee (EDel(\mathtt{a}) \cap SAdd(\mathtt{b}) \neq \emptyset)$ $(EAdd(\mathtt{b}) \cap SDel(\mathtt{a}) \neq \emptyset) \vee (EDel(\mathtt{b}) \cap SAdd(\mathtt{a}) \neq \emptyset)$	$AA_{end-start}$	
4	$(Inv(\mathtt{a}) \cap SDel(\mathtt{b}) \neq \emptyset) \vee (Inv(\mathtt{b}) \cap SDel(\mathtt{a}) \neq \emptyset)$ $(Inv(\mathtt{a}) \cap EDel(\mathtt{b}) \neq \emptyset) \vee (Inv(\mathtt{a}) \cap EDel(\mathtt{a}) \neq \emptyset)$	$AA_{during-during}$	

Table 1: Conditions for the AA static mutex relationships between actions a and b.

Definition 6 (Static proposition/action mutex (*PA mutex*)) *One proposition* $\mathtt{p}$ *is statically PA mutex with action* $\mathtt{a}$ *iff* $\mathtt{p} \in \{SDel(\mathtt{a}) \cup EDel(\mathtt{a})\}$.

As an example of the analysis of static mutex calculated at this stage, we will assume a simple example defined on the satellite domain of Figure 2. Let phenomenon1, phenomenon2 and phenomenon3 be three phenomena which represent the directions for the satellite to point to. Let satellite1 be the only satellite, initially pointing to phenomenon0 and with instrument1. turn_to(satellite1, phenomenon1, phenomenon0) is $AA_{start-start}$ with calibrate(satellite1, instrument1, phenomenon0) due to the contradiction in proposition pointing. take_image(satellite1, phenomenon1, instrument1, thermograph0) and switch_on(instrument1, satellite1) are $AA_{end-end}$ due to the proposition power_on. Finally, the actions take_image(satellite1, phenomenon1, instrument1, thermograph0) and turn_to(satellite1, phenomenon0, phenomenon1) are $AA_{during-during}$ because the invariant condition pointing is deleted (it does not point to phenomenon1). These three situations prevent the actions from starting, ending and being executed together.

6. Second Stage. Extension of the Temporal Planning Graph

The second stage of TPSYS extends the temporal planning graph. Unlike Graphplan, the extension of the temporal planning graph is not as regular and symmetric: actions have different durations and this breaks the original symmetry of the planning graph. Now, the levels are not equidistant and the effects can extend through several levels. Therefore, the actions which are mutex with an action a might be mutex at not only time t, but also at all the levels between t and $t+dur(\mathtt{a})$. This requires some changes in the extension of the planning graph; those changes are analysed in the following sections.

6.1 The temporal planning graph

The temporal planning graph of TPSYS consists of a directed, layered graph which alternates proposition levels ($P_{[t]}$) and action levels ($A_{[t]}$) with the propositions and actions,

respectively, which are present in time t. Each level represents a real time stamp (action durations are real values) instead of representing a simple planning step as in Graphplan. Consequently, time is explicitly stored in the levels of the graph which are chronologically ordered by their value.

Non-conservative actions that start at one level t may require and insert propositions at that level. In order to simplify the extension of the temporal planning graph, we have split each level into two parts: *end*-part and *start*-part. The *end*-part (*start*-part) analyses all the actions that end (start) at that time and all their conditions/effects. Actions that end (start) at an action level $A_{[t]}$ are stored in $A_{[t]end}$ ($A_{[t]start}$). Analogously, propositions achieved as final (initial) effects are stored in $P_{[t]end}$ ($P_{[t]start}$).

6.2 Formalisation of dynamic mutex

The dynamic mutex to be calculated in time t in the temporal planning graph are the action/action mutex ($AA_{[t]}$), the proposition/action mutex ($PA_{[t]}$) and the proposition/proposition mutex ($PP_{[t]}$). We use the notation $AA_{[t]}$, $PA_{[t]}$ and $PP_{[t]}$ for the dynamic mutex which are time-dependent and can disappear through the extension of the graph, whereas AA and PA are used for the static mutex which always hold. On one hand, the mutex to be calculated in the *end*-part are: $AA_{[t]end-end}$ (actions that are mutex ending in t), $PA_{[t]end-end}$ (propositions that are mutex with actions ending in t), and $PP_{[t]end-end}$ (propositions that are mutex in t after the ending of their supporting actions). On the other hand, the mutex to be calculated in the *start*-part are: $AA_{[t]start-start}$ (actions that are mutex starting in t), $AA_{[t]end-start}$ (mutex between actions which end and start in t), $PA_{[t]start-start}$ (propositions that are mutex with actions starting in t), $PP_{[t]end-start}$ (propositions that are mutex in t and are generated by actions which end/start in t), and $PP_{[t]start-start}$ (propositions that are mutex in t after the starting of their supporting actions). Although this classification of mutex might seem excessive, the main reason for breaking down these mutex relationships into *end*-part and *start*-part is to make their calculus simpler, as can be seen in the following definitions:

Definition 7 (Dynamic $AA_{[t]end-end}$ mutex) *Two actions* $\mathtt{a}, \mathtt{b}$ *are end-end mutex in* t *if one of the following holds: i)* $\mathtt{a}, \mathtt{b}$ *are* $AA_{end-end}$*, ii)* $ECond(\mathtt{a})$,$ECond(\mathtt{b})$ *are* $PP_{[t]end-end}$*, or iii)* $\mathtt{a}, \mathtt{b}$ *are* $AA_{[t-min(dur(\mathtt{a}),dur(\mathtt{b}))]start-start}$*.*

Definition 8 (Dynamic $PA_{[t]end-end}$ mutex) *Let* $\mathtt{p}$, $\mathtt{a}$ *be a proposition and an action, respectively. For each action* $\mathtt{b_i}$ *supporting* $\mathtt{p}$ *in* t*, let* $\Upsilon_{i[t]}$ *be the condition under which* $\mathtt{b_i}$ *is mutex with the persistence of* $\mathtt{p}$ *in* t*, i.e.* $\Upsilon_{i[t]} = [(\mathtt{p}, \mathtt{b_i}$ *are* $PA) \vee (\mathtt{p}$,$ECond(\mathtt{b_i})$ *are* $PP_{[t]end-end})]$*. Proposition* $\mathtt{p}$ *and action* $\mathtt{a}$ *are end-end mutex in* t *if the following condition holds:* $\bigwedge_i[\Upsilon_{i[t]} \wedge (\mathtt{a}, \mathtt{b_i}$ *are* $AA_{[t]end-end})]$*.*

Definition 9 (Dynamic $PP_{[t]end-end}$ mutex) *Let* $\mathtt{p}, \mathtt{q}$ *be two propositions and* $\{\mathtt{a_i}\}, \{\mathtt{b_j}\}$ *be two sets of actions that support* $\mathtt{p}$ *and* $\mathtt{q}$ *in* t*, respectively. Propositions* $\mathtt{p}, \mathtt{q}$ *are end-end mutex in* t *if the following conditions hold: i)* $\forall \mathtt{b_j} : \mathtt{p}, \mathtt{b_j}$ *are* $PA_{[t]end-end}$*, and ii)* $\forall \mathtt{a_i} : \mathtt{q}, \mathtt{a_i}$ *are* $PA_{[t]end-end}$*.*

Definition 10 (Dynamic $AA_{[t]start-start}$ mutex) *Two actions* $\mathtt{a}, \mathtt{b}$ *are start-start mutex in* t *if one of the following holds: i)* $\mathtt{a}, \mathtt{b}$ *are* $AA_{start-start}$*, or ii)* $SCond(\mathtt{a})$,$SCond(\mathtt{b})$ *are* $PP_{[t]start-start}$*.*

Definition 11 (Dynamic $AA_{[t]end-start}$ mutex) *Two actions* $\mathtt{a}$ *(ending in* t*) and* $\mathtt{b}$ *(starting in* t*) are end-start mutex in* t *if one of the following conditions holds: i)* $\mathtt{a}, \mathtt{b}$ *are* $AA_{end-start}$*, or ii)* $ECond(\mathtt{a})$,$SCond(\mathtt{b})$ *are* $PP_{[t]end-end}$*.*

Definition 12 (Dynamic $PA_{[t]start-start}$ mutex) *Let* $\mathtt{p}$*,* $\mathtt{a}$ *be a proposition and an action, respectively. For each action* $\mathtt{b_i}$ *supporting* $\mathtt{p}$ *in* t*, let* $\Psi_{i[t]}$ *be the condition under which* $\mathtt{b_i}$ *is mutex with the persistence of* $\mathtt{p}$ *in* t*, i.e.* $\Psi_{i[t]} = [(\mathtt{p}, \mathtt{b_i}$ *are* $PA) \vee (\mathtt{p}$,$SCond(\mathtt{b_i})$ *are* $PP_{[t]start-start})]$*. Proposition* $\mathtt{p}$ *and action* $\mathtt{a}$ *are start-start mutex in* t *if the following condition holds:* $\bigwedge_i[\Psi_{i[t]} \wedge (\mathtt{a}, \mathtt{b_i}$ *are* $AA_{[t]start-start})]$*.*

Definition 13 (Dynamic $PP_{[t]end-start}$ mutex) *Let* $\mathtt{p}$ *be a proposition firstly supported in* t *by the set of actions* $\{\mathtt{a_i}\}$ *which end in* t*. Analogously, let* $\mathtt{q}$ *be another proposition firstly supported in* t *by the set of actions* $\{\mathtt{b_j}\}$ *which start in* t*. Propositions* $\mathtt{p}, \mathtt{q}$ *are end-start mutex in* t *if the following condition holds:* $\forall \mathtt{a_i}, \mathtt{b_j} : \mathtt{a_i}, \mathtt{b_j}$ *are* $AA_{[t]end-start}$*.*

Definition 14 (Dynamic $PP_{[t]start-start}$ mutex) *Let* $\mathtt{p}, \mathtt{q}$ *be two propositions and* $\{\mathtt{a_i}\}$*,* $\{\mathtt{b_j}\}$ *be two sets of actions which support* $\mathtt{p}$ *and* $\mathtt{q}$ *in* t*, respectively. Propositions* $\mathtt{p}, \mathtt{q}$ *are start-start mutex in* t *if the following conditions hold: i)* $\forall \mathtt{b_j} : \mathtt{p}, \mathtt{b_j}$ *are* $PA_{[t]start-start}$*, and* $\forall \mathtt{a_i} : \mathtt{q}, \mathtt{a_i}$ *are* $PA_{[t]start-start}$*.*

Intuitively, $AA_{[t]}$ mutex indicates the impossibility of two actions ending, starting or abutting together at the same time t. $PA_{[t]}$ mutex indicates the impossibility of having a proposition and an action starting or ending at time t. $PP_{[t]}$ mutex indicates the impossibility of having two propositions together at time t. This calculus of the mutex relationships obtains the same mutex as Graphplan. Thus, it provides very useful information for improving the search process by avoiding combination of actions, propositions and propositions/actions which cannot be satisfied simultaneously, thus reducing the search space.

As can be observed in the previous definitions, the calculus of the mutex relationships in the *end*-part and *start*-part are nearly identical. The only difference consists of recovering and storing the information in different structures. However, in some cases the structures can be the same, reducing the storage requirements. For example, we only keep one structure $PP_{[t]}$ to store the information $PP_{[t]end-end}$, $PP_{[t]end-start}$ and $PP_{[t]start-start}$.

6.3 Extension of the temporal planning graph

The temporal planning graph is incrementally generated by means of the same forward chaining process of Graphplan. Specifically, the process consists of generating all the actions $\{\mathtt{a_i}\}$ in action level $A_{[t]}$ of the graph as soon as their start and invariant conditions are non-pairwise mutex in the proposition level $P_{[t]}$, generating their start and end effects in the proposition levels $P_{[t]}$ and $P_{[t+dur(\mathtt{a_i})]}$, respectively.

An important point to be considered when dealing with non-conservative actions in a Graphplan-based approach is the condition to finish the extension of the temporal graph. In Graphplan, this condition holds once all the propositions of the final state are non-pairwise mutex. Non-conservative actions may assert start effects which might satisfy goals in the final state *before* these actions end. In the case of conditional actions (see Definition 2),

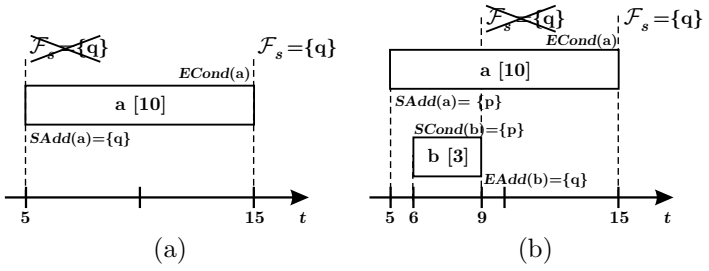


Figure 5: Two situations where conditional actions/propositions make it necessary to extend the termination criterion of the temporal graph.

however, this implies that the temporal graph extension might end in a level in which it is impossible to find a feasible plan because one of the propositions in the final state is still conditional (see Definition 3).

Figure 5 shows two examples that require extending the termination criterion of the temporal graph. Let us assume that in both cases $\mathcal{F}_s = \{\mathtt{q}\}$ and that all the final conditions ($ECond$) hold when required. In Figure 5-a, the goal is satisfied at time $t = 5$ by the start effect of a. However, the extension of the temporal graph cannot end until $t = 15$ (when a ends). This situation prevents proposition q from being valid until the termination of a. The situation becomes more complex when more actions and propositions are involved. In Figure 5-b, the goal is satisfied at time $t = 9$ when b ends, but the extension of the temporal graph cannot end until $t = 15$. In this case, proposition p, which is required by action b, is achieved at $t = 5$, but it is conditional until $t = 15$ when a ends. This information must be propagated through the graph to all the actions/propositions dependent on p, and it makes the effect q of action b invalid until $t = 15$, when the graph extension can finally terminate.

In order to avoid an incomplete extension of the temporal graph which prevents the planner from finding a feasible plan, some information on the instant of validity of the propositions must be propagated. The strategy for this propagation follows the same *disjunctive* mechanism as Graphplan. This mechanism calculates the instant of time when propositions are valid and, therefore, the temporal graph extension can terminate.

Definition 15 (Instant of validity of a proposition) *Let* $\{\mathtt{a_i}\}$ *be the set of actions that support proposition* p *and* $t_{est}(\mathtt{a_i})$ *the earliest start time of each action* $\mathtt{a_i}$*. The instant of validity of a proposition* p*,* $Inst_v(\mathtt{p})$*, is given by the value* $\min(\alpha_i)$*, where:*

$$\alpha_i = \begin{cases} 0 & \text{if } \mathtt{p} \in \mathcal{I}_s \\ \max(Inst_v(SCond(\mathtt{a_i}) \cup Inv(\mathtt{a_i})), Inst_v(ECond(\mathtt{a_i})), t_{est}(\mathtt{a_i}) + dur(\mathtt{a_i})) & \text{if } \mathtt{p} \notin \mathcal{I}_s \end{cases}$$

and the value of the expression $Inst_v(\{\mathtt{p_i}\})$ on a set of propositions $\{\mathtt{p_i}\}$ is defined as $\max(Inst_v(\mathtt{p_i}))$.

Figure 6 describes the algorithm for the extension of the temporal graph in TPSYS. The algorithm starts at time $t = 0$ (step 1) and incrementally generates new proposition and

1. $t \leftarrow 0$
2. $A_{[0]end} \leftarrow \emptyset$
3. $P_{[0]end} \leftarrow \mathcal{I}_s$
4. **while** $((\mathcal{F}_s \notin PP_{[t]end-end} \vee Inst_v(\mathcal{F}_s) > t) \wedge t \leq \mathcal{D}_{\max})$
5. *//end-part of the level*
6. **forall** $\mathtt{a_i}$ that can end in $A_{[t]end}$
7. $A_{[t]end} \leftarrow A_{[t]end} \cup \{\mathtt{a_i}\}$
8. $P_{[t]end} \leftarrow P_{[t]end} \cup EAdd(\mathtt{a_i})$
9. calculate mutex $AA_{[t]end-end}$, $PA_{[t]end-end}$ and $PP_{[t]end-end}$
10. *//start-part of the level*
11. **forall** $\mathtt{b_j}$ that can start in $A_{[t]start}$
12. $A_{[t]start} \leftarrow A_{[t]start} \cup \{\mathtt{b_j}\}$
13. $P_{[t]start} \leftarrow P_{[t]start} \cup SAdd(\mathtt{b_j})$
14. calculate mutex $AA_{[t]start-start}$, $AA_{[t]end-start}$, $PA_{[t]start-start}$, $PP_{[t]end-start}$ and $PP_{[t]start-start}$
15. $t \leftarrow$ next level in the temporal graph

Figure 6: Algorithm for the temporal graph extension.

action levels: *end*-part (steps 5–9) and *start*-part (steps 10–14). Intuitively, the algorithm generates all the levels where actions can end/start and mutex can disappear, i.e. levels which may be relevant in order to achieve a feasible plan. Note that the algorithm does not need to consider `no-op` actions through the extension because of the richer calculus of the mutex. However, this does not entail any inconvenience for the persistence of the propositions/actions which implicitly persist in time: if one proposition/action is present at time t, it will be present at any time $t' > t$.

As can be observed, the extension terminates at time t when all propositions in $\mathcal{F}_s$ are not dynamically mutex and valid. This termination condition guarantees that a feasible plan will never be shorter than t. Although this termination condition is necessary for finding a plan, it is still not sufficient as also occurs in Graphplan.

Lemma 1 (The extension of the temporal graph is complete) *The algorithm for the temporal graph extension generates a complete planning graph, where all the relevant levels in which actions can end/start are present. This way, if the algorithm terminates at a level t, there is not an intermediate level $t' \mid 0 < t' < t$ where an action that is relevant for finding an optimal plan can start/end.*

Proof The proof is direct by the definition of the algorithm and the incremental generation of the graph. Given an action level $A_{[t]}$, the algorithm generates all the actions $\{\mathtt{a_i}\}$ that can start at that level. Consequently, all the levels $A_{[t+dur(\mathtt{a_i})]}$ and $P_{[t+dur(\mathtt{a_i})]}$ are also generated. This guarantees the completeness of every level, and incrementally, of all the levels present in the graph.

Let us suppose that a proposition level $P_{[t'']}$, which is not generated in the graph, is included between two consecutive levels $P_{[t]}$ and $P_{[t']}$ present in the graph (obviously $t < t'' < t'$). The level $P_{[t'']}$ is not relevant in the temporal graph because it does not store

relevant information: no relevant actions end/start at time t'' and, therefore, no mutex can disappear (level t'' would keep the same mutex information as level t). Consequently, level t'' will not be used when finding a feasible plan.

6.4 An example of a temporal planning graph

In this section, we will illustrate the extension of the temporal planning graph for a problem of the `satellite` domain defined in Figure 2. Let us assume the duration 10, 5 and 1 for actions of type "`turn_to`", "`calibrate`" and "`take_image`", respectively. In the initial state $\mathcal{I}_s$ there exists one satellite `satellite1` pointing to `phenomenon0` with an instrument `instrument1` which is switched on and ready to be calibrated. The problem goals ($\mathcal{F}_s$) consist of taking images of `phenomenon1`, `phenomenon2` and `phenomenon3`. Figure 7 shows a part of the temporal planning graph, whose extension terminates at time $t = 27$.

The actions in the graph start when their start and invariant conditions are present and they are non-pairwise mutex. Although the conditions for actions of type "`take_image`" are present at $t = 10$, they do not start until $t = 15$ (when their conditions are non-pairwise mutex). In consequence, all the problem goals of $\mathcal{F}_s$ are independently achieved at $t = 16$. However, TPSYS continues the extension until $t = 27$ when the problem goals are non-pairwise mutex and valid. This indicates to us that there is no feasible plan shorter than 27 and it acts as a lower bound of the makespan of the plan.

This example also illustrates the increase in the complexity of the temporal planning graph *w.r.t.* the number of levels. The temporal graph has 16 levels, whereas the equivalent classical Graphplan planning graph (with no duration on actions) has only 5 levels. Now, the degree of concurrency between actions is higher and this forces the algorithm to explicitly generate more levels. In addition, the number of levels depends on the duration of the actions and their *dispersion*[5], and it can completely change when the duration of any action changes.

7. Third Stage. Search of a Temporal Plan

The third stage of TPSYS performs the search of a temporal plan from the temporal graph generated in the second stage. In TPSYS we have analysed two different approaches to perform this search:

- The search approach based on Graphplan. In this approach, the third stage follows the same strategy as Graphplan. However, the *right-to-left* directionality of Graphplan or TGP search might be broken due to the fact that durative actions provide two alternatives to support the goals (final and initial effects) and they may require final conditions. The search is based on an iterative deepening and chronological backtracking that extracts the plan through the levels of the planning graph. This means that the second and third stage are executed in an interleaved way until a plan is found or the value $\mathcal{D}_{max}$ is exceeded.

5. The number of levels of the temporal graph grows when the actions have widely different durations. Specifically, the worst case happens when the greatest common divisor of the durations is 1. If the gcd = 1, the algorithm must generate the maximum number of levels, thus increasing the complexity of the second stage.

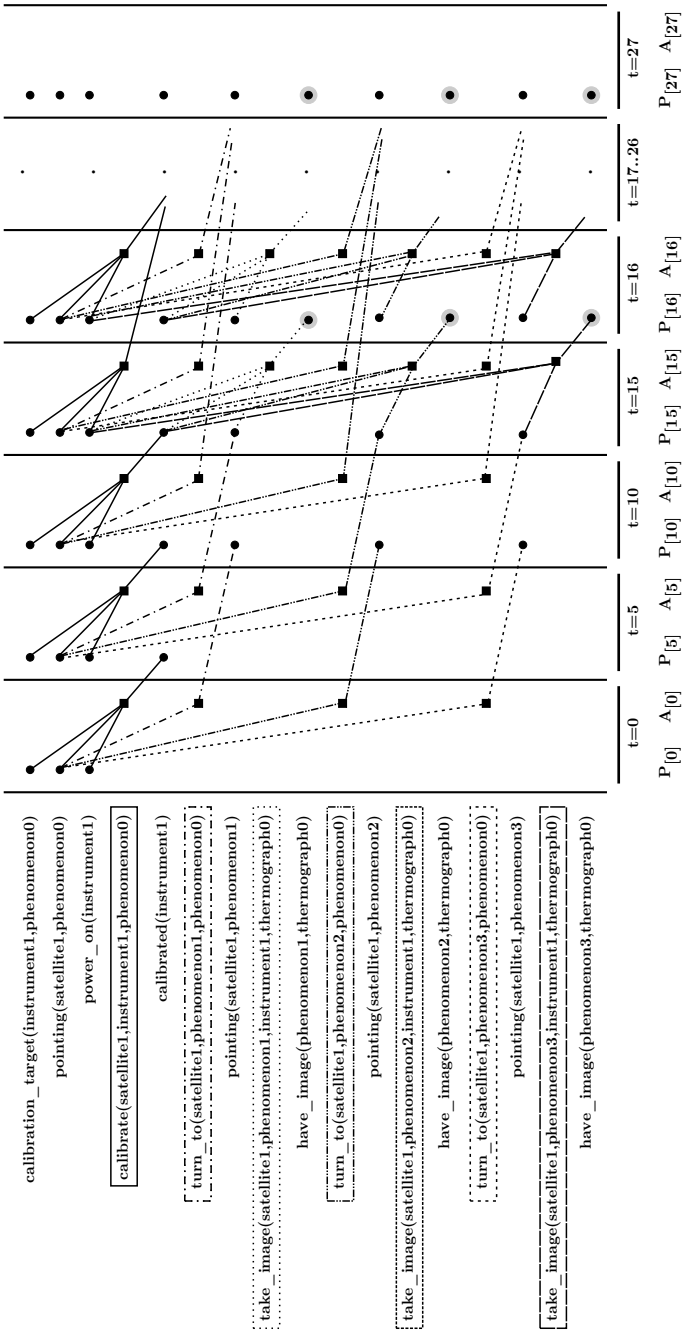


Figure 7: Part of the temporal planning graph for the example of the `satellite` domain. Although the real graph of TPSYS is more complex (it contains 18 actions and 13 propositions), only the most relevant information is represented. For simplicity, the graph does not show the explicit separation of each level into *end*-part and *start*-part.

- The search approach based on least-commitment and heuristic techniques. In this approach, the third stage is divided into two new stages. First, a relaxed plan is generated in a backward chaining way. Second, this relaxed plan is used as a *skeleton* to generate the temporal plan and as the basis for calculating heuristic estimations in a forward chaining way. Now, the original second and third stages are no longer executed in an interleaved way. Consequently, the third stage extends the temporal graph as much as necessary until finding a plan or exceeding the value $\mathcal{D}_{max}$.

7.1 Search based on Graphplan

The Graphplan search consists of a *simple* regressive process, planning the necessary actions to support the problem (sub)goals. However, when handling actions with local conditions/effects, there are different ways of achieving the goals, not only by the *at end* effects but also by the *at start* effects. Consequently, planning such an action commits a plan to satisfying the start, invariant and end conditions. This makes the regressive process *trickier*, as shown in the following example.

Let t be the instant of time when proposition p must be satisfied during the extraction of a plan. Let us suppose that conditional action a achieves p as a start effect ($\mathtt{p} \in SAdd(\mathtt{a})$) at time t. The usual strategy after planning a is to satisfy all its initial and final conditions. Initial conditions must occur at time t, but final conditions must occur at time $t' = t + dur(\mathtt{a})$. Obviously, $t' > t$ and this forces the search algorithm to revisit a previously visited instant of time t'. This situation is new in a Graphplan approach and involves a *right-to-left* and *left-to-right* strategy until all the problem goals are supported.

7.1.1 Extraction of a temporal plan

The algorithm for extracting a temporal plan is shown in Figure 8. TPSYS uses two basic structures indexed by an instant of time t: $GoalsToSatisfy_{[t]}$ and $Plan_{[t]}$. $GoalsToSatisfy$, which is initialised with the propositions in $\mathcal{F}_s$ (step 2), stores the goals to be satisfied at each level. $Plan$ stores the actions committed at each level and is initialised empty (step 3). The search process is repeated while $GoalsToSatisfy_{[t]}$ is not empty. Step 12 extracts a proposition p to be satisfied at time t. Proposition p might already be supported in $Plan$ because actions in $Plan$ are planned at different levels and not always in a *right-to-left* order (step 13). If p is not supported, step 14 selects the action to be planned (backtracking point). All the actions supporting p at time t must be considered in order to guarantee the completeness of the algorithm. This includes any action a, starting at s and ending at e, with local effects supporting p ($\mathtt{p} \in \{SAdd(\mathtt{a}) \cup EAdd(\mathtt{a})\}$). Step 15 checks the compatibility of a with the actions in $Plan$, thus guaranteeing the correctness of the plan. This requires making sure that a is not conflicting with already planned actions (AA and PA static mutex and $AA_{[s]end-start}$, $AA_{[s]start-start}$, $AA_{[e]end-end}$ and $AA_{[e]end-start}$ dynamic mutex). In addition, it is necessary to make sure that a is not mutex with the goals to be satisfied in $GoalsToSatisfy_{[s]}$ ($PA_{[s]start-start}$ dynamic mutex), and $GoalsToSatisfy_{[e]}$ ($PA_{[e]end-end}$ dynamic mutex). If a is compatible, then the structures $Plan_{[s]}$ and $GoalsToSatisfy_{[s/e]}$ are updated (steps 16–18). Finally, when $GoalsToSatisfy$ gets empty, $Plan$ contains all the committed actions of the plan with optimal makespan. Otherwise, it is necessary to

1. $t \leftarrow$ time when the temporal graph extension finished at the second stage
2. $GoalsToSatisfy_{[t]} \leftarrow \mathcal{F}_s$
3. $Plan \leftarrow \emptyset$
4. **while** $(t \geq 0)$
5. **if** $t = 0$
6. **if** $GoalsToSatisfy_{[t]} \subseteq \mathcal{I}_s$
7. success *// Plan found*
8. **else**
9. failure *// Backtracking*
10. **else**
11. **while** $GoalsToSatisfy_{[t]} \neq \emptyset$
12. extract p from $GoalsToSatisfy_{[t]}$
13. **if** p is not supported in $Plan$
14. select $\langle$a,$s, e\rangle$ which supports p in t *// if* $\nexists$a *then backtracking*
15. **if** a is compatible in $Plan$
16. $Plan_{[s]} \leftarrow Plan_{[s]} \cup \{\mathtt{a}\}$
17. $GoalsToSatisfy_{[s]} \leftarrow GoalsToSatisfy_{[s]} \cup SCond(\mathtt{a}) \cup Inv(\mathtt{a})$
18. $GoalsToSatisfy_{[e]} \leftarrow GoalsToSatisfy_{[e]} \cup ECond(\mathtt{a}) \cup Inv(\mathtt{a})$
19. **else**
20. goto step 14 and select another action
21. $t \leftarrow e$ *// the time when* a *ends (this breaks the* Graphplan *directionality)*

Figure 8: Algorithm for the extraction of a temporal plan.

extend a new level of the temporal planning graph (second stage) and repeat the search process from the new level in the same interleaved way as Graphplan.

7.1.2 Properties of the search process

The algorithm for the extraction of a temporal plan in TPSYS has properties which are identical to Graphplan because both are based on the same chronological backtracking search. The non-conservative model of actions that TPSYS handles does not entail any inconvenience to guarantee the properties of correctness, completeness and optimality.

Correctness In TPSYS, the correctness of the algorithm is based on two important points: i) all the problem goals are supported and all the local (initial, invariant and final) conditions are supported when they are required; and ii) all the actions must be executed with no conflicts in the plan.

The previous points are always guaranteed in TPSYS by the definition of the search algorithm itself. First, the algorithm does not terminate until all the problem (sub)goals are supported ($GoalsToSatisfy_{[t]} = \emptyset$ in every level t). Second, when an action is planned, the algorithm guarantees its executability without conflicts by checking the compatibility of that action with the rest of actions in the plan.

Completeness The search strategy based on chronological backtracking used in the algorithm explores all the alternatives to support each goal. Consequently, if there exists a plan which solves a problem, the search algorithm always finds it.

Lemma 2 (The search algorithm is complete) *If there exists a plan with makespan t and the algorithm performs the search from level t, the plan is found.*

Proof The proof is trivial and relies on the completeness of the extension of the temporal planning graph (see Lemma 1) and on the chronological backtracking of the search process. Given a plan with makespan t, the completeness of the temporal graph guarantees that the actions of that plan start/end in the levels of the temporal graph of size t. In addition to this, the search algorithm considers all the actions supporting each goal (backtracking point). Therefore, all the feasible alternatives for finding a plan are considered from time t before generating a new level with time $t' > t$.

Optimality The search algorithm is optimal *w.r.t.* makespan. This can be guaranteed due to the chronological extension of the temporal graph and the complete search performed from each level.

Theorem 1 (The search algorithm is optimal) *The first plan the algorithm extracts from the temporal planning graph is a plan of optimal duration.*

Proof By contradiction, let $\mathcal{P}_t$ be the first plan (of makespan t) the algorithm extracts. We will assume this plan is not optimal, so there exists an alternative plan $\mathcal{P'}_{t'}$ (of makespan $t' < t$) which is optimal but has not been found. This implies one of the following cases: i) the level t' has not been generated in the second stage and consequently no search has been performed from that level, or ii) the level t' has been generated but the search algorithm has not found the plan $\mathcal{P'}_{t'}$. The first case is false by Lemma 1 which claims the completeness of the temporal graph extension where all the relevant levels for a plan are present in the graph. The second case is also false by Lemma 2, which claims the completeness of the search algorithm that explores all the feasible actions that can end at level t'. This contradicts the initial choice of the existence of plan $\mathcal{P'}_{t'}$ and, consequently, the first plan $\mathcal{P}_t$ extracted by the algorithm is a plan of optimal makespan.

7.1.3 An example of the extraction of a temporal plan

In this section, we will illustrate the extraction of a temporal plan for the problem defined in section 6.4 on the `satellite` domain (see Figure 2). Let us assume that the problem goals ($\mathcal{F}_s$) consist of taking images of `phenomenon1`, `phenomenon2` and `phenomenon3`. The search starts from time $t = 27$ when the temporal graph extension has terminated (see Figure 7) and performs the following steps:

1. Initially, $Plan$ is empty and $GoalsToSatisfy_{[27]} = \mathcal{F}_s$
2. The actions that support the goals are the actions of type "`take_image`" that start at $t = 26$. When one of these actions is planned, the action is inserted into $Plan$ and its conditions into $GoalsToSatisfy$. In these actions, the invariant conditions must be held throughout the action execution and final conditions must be satisfied at the end of the action execution. This makes it necessary to support the final condition `power_on`(`instrument1`) at the level in which the action ends. In this case, it is not necessary to plan new actions because `power_on`(`instrument1`) is present in the initial

state. Therefore, the algorithm tries to satisfy the rest of the conditions of the action, moving to the levels where they are required. As can be observed, this search follows the same chronological backtracking process as Graphplan; i.e. planning one action, inserting its conditions as goals to be satisfied and so on.

3. Finally, the search algorithm proves the impossibility of finding a plan of makespan 27, extending the level $t = 28$ and repeating the search. This process is repeated, interleaving the extension of a new level and the search from it until level $t = 38$. At this level, the algorithm finds the optimal plan shown in Figure 9.

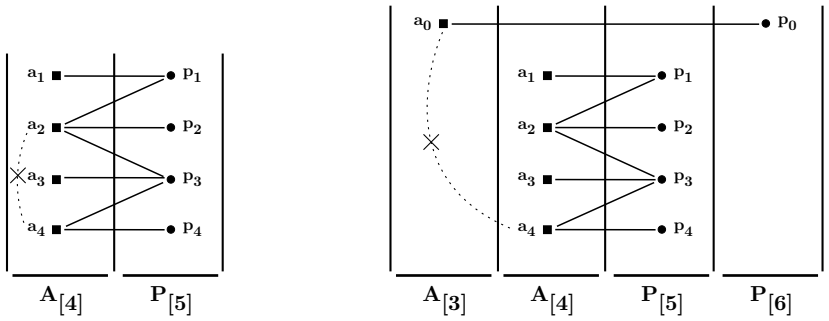
```
0.001:  calibrate(satellite1,instrument1,phenomenon0) [5]
5.006:  turn_to(satellite1,phenomenon1,phenomenon0) [10]
15.016: take_image(satellite1,phenomenon1,instrument1,thermograph0) [1]
16.017: turn_to(satellite1,phenomenon2,phenomenon1) [10]
26.027: take_image(satellite1,phenomenon2,instrument1,thermograph0) [1]
27.028: turn_to(satellite1,phenomenon3,phenomenon2) [10]
37.038: take_image(satellite1,phenomenon3,instrument1,thermograph0) [1]
```

Figure 9: Optimal temporal plan with the solution to the `satellite` problem. The time stamps for the starting point of the actions include a small value which is necessary to avoid the synchronisation problems in the validation of PDDL2.1 plans (Fox & Long, 2001; Long & Fox, 2001).

7.1.4 Memoization in temporal planning

The completeness of the chronological backtracking strategy used in the third stage of TPSYS has a serious inconvenience when the same goals are considered at the same levels of the graph in successive iterations during the search. For instance, after one failed stage of search from level t, the algorithm resumes the search from a new level $t' > t$ where it repeats part of the same search (and failures) performed at level t. Moreover, this inconvenience can also occur at the internal levels of the temporal graph. This situation, which vastly degrades the performance of the chronological backtracking, also appears in Graphplan. However, the impact is greater in a temporal planning approach where the planning graph contains more levels: the same unfruitful search is repeated more times.

Graphplan copes with this problem by means of a memoization technique during the search (Blum & Furst, 1997). If a set of goals $\{\mathtt{p_i}\}$ to be satisfied at level t cannot be solved, Graphplan memoizes $\{\mathtt{p_i}\}$ as unsolvable at level t. This memoization prevents the algorithm from repeating the same failures in the future. Kambhampati (1999, 2000) efficiently extends Graphplan memoization by means of CSP techniques, including Explanation-Based Learning (EBL) and Dependency-Directed Backtracking Capabilities (DDB). The EBL technique extracts information on the real origin of the conflicts, whereas the DDB technique propagates information on the failure to avoid an entire chronological backtracking process. Kambhampati's experiments using these techniques show spectacular improvements in the performance of the search, maintaining the properties of completeness and



(a) - Memoization in classical planning (b) - Memoization in temporal planning

Figure 10: Memoization in classical planning *vs.* memoization in temporal planning. Dotted lines between actions represent mutex between them. For simplicity, only the relevant levels for the example are represented.

optimality. In consequence, the application of EBL and DDB techniques seems promising in a temporal setting based on Graphplan. However, the application of such techniques in a temporal setting is not as direct as in classical planning, as can be seen in Figure 10.

Figure 10 illustrates the differences between memoization in classical planning (Graphplan & EBL+DDB) and memoization in temporal planning (TPSYS). In Figure 10-a, the set of propositions to satisfy at level $P_{[5]}$ is $\{\mathtt{p_1}, \mathtt{p_2}, \mathtt{p_3}, \mathtt{p_4}\}$. In the example, there is no feasible combination of actions $\{\mathtt{a_1}, \mathtt{a_2}, \mathtt{a_3}, \mathtt{a_4}\}$ at level $A_{[4]}$ supporting $P_{[5]}$ because $\mathtt{a_2}$ and $\mathtt{a_4}$ are mutex. Consequently, Graphplan memoizes $\{\mathtt{p_1}, \mathtt{p_2}, \mathtt{p_3}, \mathtt{p_4}\}$ as an unsolvable goal set in $P_{[5]}$. This memoization is sound, but there clearly exists a more precise goal subset ($\{\mathtt{p_2}, \mathtt{p_4}\}$) which is unsolvable. Kambhampati's EBL technique detects that neither $\mathtt{p_1}$ nor $\mathtt{p_3}$ takes part in the resolution of the rest of goals and the minimal goal subset which is memoized at level $P_{[5]}$ is $\{\mathtt{p_2}, \mathtt{p_4}\}$. The fact of memoizing the minimal goal subset increments the opportunities to discard bad choices in future searches (i.e. any combination of goals containing $\{\mathtt{p_2}, \mathtt{p_4}\}$ will be unsolvable at $P_{[5]}$). Moreover, the DDB technique allows us a better guidance of the chronological backtracking, thus avoiding the entire exploration of the search space. In Figure 10-a, propositions $\{\mathtt{p_1}, \mathtt{p_2}, \mathtt{p_3}, \mathtt{p_4}\}$ can be solved *iff* $\{\mathtt{p_2}, \mathtt{p_4}\}$ are solved, and any assignment of actions to propositions $\mathtt{p_1}$ and/or $\mathtt{p_3}$ is useless. The DDB technique avoids these assignments which are not successful when solving the real conflict.

In Figure 10-b, actions $\{\mathtt{a_2}, \mathtt{a_4}\}$ are no longer mutex. Consequently, the set of propositions $\{\mathtt{p_1}, \mathtt{p_2}, \mathtt{p_3}, \mathtt{p_4}\}$ is now solvable at $P_{[5]}$. However, when dealing with actions with duration that can be executed throughout several levels, an action can conflict with actions planned at other levels. This happens in Figure 10-b, where action $\mathtt{a_4}$ (duration 1) is mutex with action $\mathtt{a_0}$ (duration 3). In this case, propositions $\{\mathtt{p_1}, \mathtt{p_2}, \mathtt{p_3}, \mathtt{p_4}\}$ are again unsolvable at $P_{[5]}$, but they cannot be memoized because the origin of the conflict depends on action $\mathtt{a_0}$ which is already present in the plan. This indicates an important difference between memoization in classical and temporal planning: in temporal planning, the possibility for memoization may depend on actions that are already planned. In the example, the mem-

oization of $\{\mathtt{p_1}, \mathtt{p_2}, \mathtt{p_3}, \mathtt{p_4}\}$ at $P_{[5]}$ could be wrong because it depends on the presence of $\mathtt{a_0}$ in the plan, and the propositions will become feasible if $\mathtt{a_0}$ is not in the plan.

TPSYS extends the memoization based on EBL+DDB introduced by Kambhampati in order to be applicable in a temporal planning setting. This requires some modifications in the algorithm presented in Figure 8. Now, the algorithm is divided into two new algorithms: *Extract_Temporal_Plan* (Figure 11) and *Satisfy_Goals* (Figure 12), which are recursively invoked. The search starts with the first algorithm and it is invoked at each level of the planning graph where there are goals to satisfy. The second algorithm is invoked to satisfy such goals, assigning one action to each goal. Once the set of goals to be satisfied at one level gets empty, *Extract_Temporal_Plan* is invoked again to continue the search from another level.

1. **Extract_Temporal_Plan** $\rightarrow$ {boolean}
2. $t \leftarrow$ longest level t' of the temporal graph where $GoalsToSatisfy_{[t']} \neq \emptyset$
3. **if** $\exists M \in Memo_{[t]} \mid M \subseteq GoalsToSatisfy_{[t]}$
4. failure and return $\{TRUE\}$ *// goals already memoized as unsolvable at level t*
5. **else**
6. Invoke Satisfy_Goals($GoalsToSatisfy_{[t]}, t, \emptyset$) $\rightarrow \{\mathtt{CC'}, \mathtt{is\_memo\_possible'}\}$
7. **if** Satisfy_Goals fails
8. **if** $\mathtt{is\_memo\_possible'} = TRUE$
9. $Memo_{[t]} \leftarrow Memo_{[t]} \cup \{\mathtt{CC'}\}$ *// memoization of the conflict set*
10. failure and return $\{\mathtt{is\_memo\_possible'}\}$
11. **else**
12. success and return $\{\mathtt{is\_memo\_possible'}\}$

Figure 11: Algorithm for the extraction of a temporal plan using memoization.

Extract_Temporal_Plan returns a boolean value which indicates whether or not the memoization of the goals is possible. If the goals to be satisfied at level t are memoized as unsolvable in $Memo_{[t]}$, the algorithm ends with failure (step 4). This condition preserves the properties of completeness and optimality, because that search will not lead to a feasible plan. If the goals are not memoized, step 6 invokes *Satisfy_Goals*. If *Satisfy_Goals* fails, returning a conflict set with the propositions which cannot be solved and the memoization is possible, step 9 memoizes the conflict set in $Memo_{[t]}$, and *Extract_Temporal_Plan* ends with failure (step 10). If *Satisfy_Goals* is successful, *Extract_Temporal_Plan* ends successfully (step 12).

Satisfy_Goals requires as input the goals to be satisfied (GTS) at level t and the set of actions (A) planned to solve the goals in GTS. Now, it is necessary to establish a difference between the actions present in $Plan$ and the actions in A to solve goals at the current level t. The output is the proposition conflict set (as a subset of GTS) and a boolean value which indicates whether the memoization is possible at level t. The algorithm includes the structure CC to store the conflict set and the flag is_memo_possible. *Satisfy_Goals* checks whether GTS can be solved at $t = 0$, terminating with success or failure (steps 2–6). When GTS gets empty, the search is done at another level by invoking *Extract_Temporal_Plan* (steps 7–12). Step 14 extracts the proposition p to be satisfied and, if it is not already satisfied in $Plan$ (steps 15–19), step 21 extracts an action $\mathtt{a_i}$ to support it. If $\mathtt{a_i}$ is incompatible

1. **Satisfy_Goals**(GTS:goals,t:level,A:actions} $\rightarrow$ {conflict set, boolean}
2. **if** $t = 0$
3. **if** $GTS \subseteq \mathcal{I}_s$
4. success and return $\{\emptyset, FALSE\}$ *// plan found*
5. **else**
6. failure and return $\{GTS, TRUE\}$ *// backtracking and recursion*
7. **else if** $GTS = \emptyset$
8. Invoke Extract_Temporal_Plan $\rightarrow$ {$\texttt{is\_memo\_possible}'$} *// search in another level*
9. **if** Extract_Temporal_Plan fails
10. failure and return $\{\emptyset, \texttt{is\_memo\_possible}'\}$
11. **else**
12. success and return $\{\emptyset, FALSE\}$
13. **else**
14. extract $\texttt{p}$ from GTS
15. **if** $\texttt{p}$ is not satisfied in $Plan$
16. $\texttt{CC} \leftarrow \{\texttt{p}\}$; $\texttt{is\_memo\_possible} \leftarrow TRUE$
17. $A_p \leftarrow \{\langle \texttt{a}_\texttt{i}, s_i, e_i \rangle\}$ with actions supporting $\texttt{p}$
18. **if** $A_p = \emptyset$
19. failure and return $\{\texttt{CC}, \texttt{is\_memo\_possible}\}$
20. **else**
21. extract $\langle \texttt{a}_\texttt{i}, s_i, e_i \rangle$ from A_p
22. **if** $\texttt{a}_\texttt{i}$ is incompatible with any action $\texttt{b} \in A$
23. $\texttt{p}_\texttt{b} \leftarrow$ proposition which forced the choice of action $\texttt{b}$; $\texttt{CC} \leftarrow \texttt{CC} \cup \{\texttt{p}_\texttt{b}\}$
24. goto step 18 and select another action
25. **else if** $\texttt{a}_\texttt{i}$ is incompatible with any action $\texttt{c} \in Plan$
26. $\texttt{is\_memo\_possible} \leftarrow FALSE$ *// memoization at this level is no longer possible*
27. goto step 18 and select another action
28. **else**
29. $Plan_{[s_i]} \leftarrow Plan_{[s_i]} \cup \{\texttt{a}_\texttt{i}\}$ *// action* $\texttt{a}_\texttt{i}$ *can be planned*
30. $GoalsToSatisfy_{[s_i]} \leftarrow GoalsToSatisfy_{[s_i]} \cup SCond(\texttt{a}_\texttt{i}) \cup Inv(\texttt{a}_\texttt{i})$
31. $GoalsToSatisfy_{[e_i]} \leftarrow GoalsToSatisfy_{[e_i]} \cup ECond(\texttt{a}_\texttt{i}) \cup Inv(\texttt{a}_\texttt{i})$
32. $A \leftarrow A \cup \{\texttt{a}_\texttt{i}\}$ *//* $\texttt{a}_\texttt{i}$ *planned at this level*
33. Invoke Satisfy_Goals$(GTS, t, A) \rightarrow \{\texttt{CC}', \texttt{is_memo_possible}'\}$
34. $\texttt{is\_memo\_possible} \leftarrow \texttt{is\_memo\_possible} \wedge \texttt{is\_memo\_possible}'$
35. **if** Satisfy_Goals fails
36. **if** $\texttt{p} \in \texttt{CC}'$
37. $\texttt{CC} \leftarrow \texttt{CC} \cup \texttt{CC}'$ *// union of the conflict sets*
38. goto step 18 and select another action
39. **else**
40. failure and return $\{\texttt{CC}', \texttt{is\_memo\_possible}\}$ *// no chronological backtracking*

Figure 12: Algorithm for the satisfaction of goals using memoization.

with any action in A, the conflict set CC is updated with the proposition which made that action be selected (steps 22–24). This detects the origin of the conflict that makes the goals unsolvable, as the technique EBL proposes. The set CC is incrementally updated with all the conflicting propositions (though the memoization in the current level is still possible). However, if one selected action is incompatible with actions in $Plan$, the memoization at the current level is no longer possible because it depends on the actions planned in other levels (steps 25–27). Otherwise, if $\mathtt{a_i}$ can be planned, the structures $Plan$, $GoalsToSatisfy$ and A are updated in steps 29–32, and *Satisfy_Goals* is invoked with the rest of goals to satisfy in GTS (step 33). Steps 34–40 are based on the DDB technique to avoid having to explore the entire search space. If proposition p is present in CC′, it means that the resolution of p is in conflict with the resolution of another proposition in GTS. In this case, step 37 joins the conflict sets CC and CC′. If $\mathtt{p} \notin \mathtt{CC}'$, then p does not take part in the conflict set and any new re-assignment to p can be omitted, breaking the chronological backtracking and improving the search performance.

In a temporal planning approach, the temporal memoization reduces the search space and improves the behaviour of the chronological backtracking, especially in the situations where the goals are unsolvable as a result of actions planned at other levels. In chronological backtracking, all the possible assignments of actions to propositions are studied (complete exploration). In contrast, the temporal memoization and the application of the DDB technique *forbids* any new assignment until backtracking to the level at which the real conflict arises. However, the mutex between actions of different levels can prevent the algorithm from doing a significant number of memoizations. This makes the conditions to memoize a conflict set stricter than in Graphplan. Therefore, the benefits of temporal memoization in TPSYS are not as spectacular as the memoization in Graphplan, as will be shown in section 8.1.

7.1.5 Heuristic search in temporal planning

The Graphplan search is similar to a CSP resolution process (Kambhampati, Lambrecht, & Parker, 1997; Kambhampati, 2000). Consequently, the application of CSP heuristics to a Graphplan approach seems to be a straightforward result. Kambhampati applied heuristics for variable (propositions) and value (actions to satisfy propositions) orderings in Graphplan (Kambhampati, 2000; Kambhampati & Sanchez Nigenda, 2000). Although his experimental results showed some improvements in the performance, there still exists an important drawback: a complete exploration of all the levels of the graph must be done before finding a plan. Therefore, the precise ordering of actions/propositions is only relevant at the last level of the graph and does not greatly reduce the search space. This demonstrates the difficulty of finding domain-independent admissible heuristics for Graphplan (Nguyen, Kambhampati, & Nigenda, 2002). Consequently, it seems more promising to focus on non-admissible heuristics that achieve *reasonable quality* solutions in most domains.

In TPSYS we have applied two types of heuristics to reduce the search space (see Figure 13). The first one reduces the number of levels to be explored in the temporal graph, thus reducing the search depth. The second one reduces the number of actions to be explored while supporting each proposition, thus reducing the branching factor.

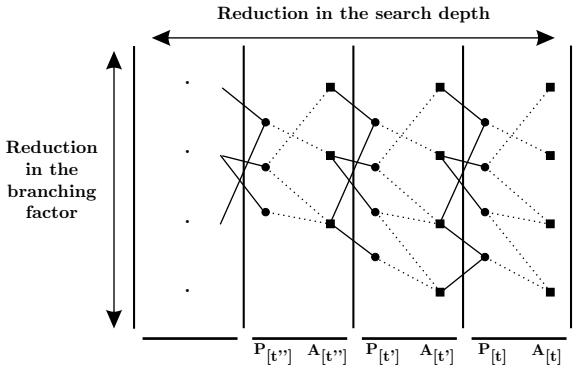


Figure 13: The search space reduction can be tackled by: i) reducing the search depth, and ii) reducing the branching factor.

Reduction in the search depth The search stage in TPSYS is done through the temporal graph, so any reduction in the size of that graph will speed up the search. We have implemented three methods to avoid the exploration of levels which might be irrelevant in the search.

Method 1 (Reduction in the number of levels generated in the 2nd stage) *Each level $A_{[t]start}$ of the temporal graph generates all the actions $\{\mathtt{a_i}\}$ that can start at time t. This involves generating as many levels $t' = t + dur(\mathtt{a_i})$ as actions with different durations in $A_{[t]start}$. However, there are some levels which do not include new actions or propositions and, consequently, they might be irrelevant in the graph. If one level t does not contain any new action $\mathtt{a_i}$, this method will generate only one level with the time $t' = t + \max(dur(\mathtt{a_i})), \forall \mathtt{a_i} \in A_{[t]start}$ instead of generating all the intermediate levels $t' = t + dur(\mathtt{a_i})$.*

Method 2 (Reduction in the number of levels explored in the 3rd stage) *When all the goals to satisfy at level t are the result of planning `no-op` actions to keep the persistence, this method postpones the satisfaction of these goals to a previous level. This postponing mechanism holds until a level where one action starts or ends is achieved and, therefore, its initial, invariant and final conditions must also be satisfied.*

Method 3 (Skipping levels in the 3rd stage after a failed search) *After one failed search from level t, the algorithm extends a new level in the temporal graph. Instead of resuming the search from that level, this method continues extending the temporal graph (without performing any search) until achieving one level $t' \geq t + \max(dur(\mathtt{a_i})), \forall \mathtt{a_i}$ that supports $\mathcal{F}_s$, where the search is resumed again.*

Intuitively, Method 1 avoids the generation of all the reachable levels. This reduces the number of levels in the graph, especially when the greatest common divisor of the duration of the actions is 1. The benefits of this method are twofold: i) the temporal graph extension

requires less effort, and ii) the search space in the extraction of a plan is smaller. Method 2 tends to tilt the achievement of the (sub)goals to the extreme levels (start and ending) of the action execution. This reduces the overhead when studying the actions that support the goals and avoids an exhaustive analysis of all the levels of the graph, thus making the search less dependent on the number of the levels in the graph. Method 3 prevents the algorithm from searching from the immediate level(s) after a failed search, by skipping and ignoring them. This increases the opportunities for the actions that support the problem goals to be executed without conflicts. This method also increases the number of levels between two consecutive search stages, thus reducing the negative effect due to a high number of levels in the graph. The main drawback of these three methods is, however, that they are neither complete nor optimal-preserving.

Reduction in the branching factor The complete search in TPSYS analyses all the combinations of actions that satisfy the goals at each level. For instance, at a level with 3 goals that are supported by 2 actions each, the number of combinations to study is 2^3, which involves a high branching factor per level. We have implemented two types of heuristics that discard actions in the goal satisfaction. Heuristics H1–H3 only rely on the definition of the action to decide whether the action will be used to satisfy each proposition. In contrast, heuristics H4–H5 are based on the set of propositions (current state) to be satisfied at each level in order to determine whether the action will be used to satisfy each proposition.

Heuristic 1 (min-duration) *This heuristic selects an action* $\mathtt{a}$ *to satisfy a proposition* $\mathtt{p}$ *iff* $dur(\mathtt{a}) = \min(\mathtt{dur}(\mathtt{a_i})), \forall \mathtt{a_i}$ *that supports* $\mathtt{p}$.

Heuristic 2 (min-EET) *Let* $EET(\mathtt{a})$ *be the earliest end time of* $\mathtt{a}$ *in the temporal graph. This heuristic selects an action* $\mathtt{a}$ *to satisfy proposition* $\mathtt{p}$ *iff* $EET(\mathtt{a}) = \min(EET(\mathtt{a_i})), \forall \mathtt{a_i}$ *that supports* $\mathtt{p}$.

Heuristic 3 (max-conds-IS) *Let* $num\_conds\_IS(\mathtt{a})$ *be the number of conditions of* $\mathtt{a}$ *that are present in the initial state. This heuristic selects an action* $\mathtt{a}$ *to satisfy one proposition* $\mathtt{p}$ *iff* $num\_conds\_IS(\mathtt{a}) = \max(num\_conds\_IS(\mathtt{a_i})), \forall \mathtt{a_i}$ *that supports* $\mathtt{p}$.

Previous heuristics select an action according to its duration, its *earliest end time* or the number of conditions that are present in the initial state, respectively. This requires a selection criterion based on invariable information, that does not rely on the level and state (set of propositions to be satisfied). Therefore, if the set of actions $\{\mathtt{a}, \mathtt{b}\}$ is selected to satisfy proposition $\mathtt{p}$, those actions will always be used (in any level of the graph) to satisfy $\mathtt{p}$. This allows us to take the decision prior to the search stage (for instance, in the first stage). However, if a wrong selection for an action is made, that decision could lead to the loss of feasible plans. In order to overcome this limitation, the following two heuristics rely on the state's information and estimation from a relaxed plan, avoiding the fixed selection of an action for a proposition. The idea of a relaxed plan consists of ignoring the negative effects of actions as in most of the current planners.

Heuristic 4 (min-duration-plan) *Let GTS be the set of propositions to be satisfied at one level and* $dur\_plan(\mathtt{a}, GTS)$ *be the duration of the relaxed plan of shortest duration*

that supports the state (set of propositions) after the application of a *to support GTS. This heuristic selects an action* a *to satisfy a proposition* p$\in GTS$ *iff* $dur\_plan(\mathtt{a}, GTS) = \min(dur\_plan(\mathtt{a_i}, GTS)), \forall \mathtt{a_i}$ *that supports* p.

Heuristic 5 (min-acts-plan) *Let GTS be the set of propositions to be satisfied at one level and* $num\_acts\_plan(\mathtt{a}, GTS)$ *be the number of actions of the relaxed plan with the lowest number of actions that supports the state (set of propositions) after the application of* a *to support GTS. This heuristic selects an action* a *to satisfy a proposition* p$\in GTS$ *iff* $num\_acts\_plan(\mathtt{a}, GTS) = \min(num\_acts\_plan(\mathtt{a_i}, GTS)), \forall \mathtt{a_i}$ *that supports* p.

The two previous heuristics tend to *find* the plan that simplifies the complexity (duration or number of actions) in order to satisfy any state. This allows a different selection of an action depending on the level to be applied. These heuristics are regressive (Bonet & Geffner, 1999), because they use the best relaxed plan as an estimation to reach the current state from the initial state. The main advantage of these heuristics is a more precise estimation than heuristics H1–H3, but their calculus is more time consuming and cannot be done before search.

In addition to the previous heuristics, we have included a heuristic that randomly selects the action to be used to satisfy a proposition. This heuristic makes the decision very quickly (as heuristics H1–H3) and provides a different selection of actions (as heuristics H4–H5). The main drawback is that the decision does not take into account any kind of information and it is non-deterministic.

Heuristic 6 (random) *This heuristic selects an action* a *to satisfy a proposition* p *with a given probability.*

7.2 Search based on least-commitment and heuristic techniques

The Graphplan backward search based on chronological backtracking has some inefficiencies that impose serious limitations when dealing with large problems (Fox & Long, 1999a; Zimmerman & Kambhampati, 1999), which are more relevant in temporal problems (Garrido & Onaindía, 2003). In this section, we present a new search process for temporal planning to overcome these inefficiencies. This search uses the information of a planning graph to improve the scalability of the planner and is competitive with other state-of-the-art planners *w.r.t.* the plan quality.

7.2.1 Motivation. Main inefficiencies of the Graphplan backward search

Backward search in TPSYS preserves the same properties of completeness and optimality as Graphplan, but it entails the most costly stage of TPSYS. Additionally, this search presents some inefficiencies —inherited from Graphplan— which impose severe limitations on its performance, thus reducing the scalability of the planner when solving large temporal problems.

One of the most noticeable inefficiencies appears in problems with a high degree of symmetry since the Graphplan search makes a lot of unsuccessful attempts to plan nearly identical actions. Let us consider the problem defined in section 6.4 on the `satellite` domain. The optimal plan for this problem contains seven actions (1×`calibrate`, 3×`turn_to`

Propositions	Actions
P1-calibration_target(instrument1,phenomenon0)	A1-calibrate(satellite1,instrument1,phenomenon0)
P2-pointing(satellite1,phenomenon0)	A2-turn_to(satellite1,phenomenon1,phenomenon0)
P3-power_on(instrument1)	A3-turn_to(satellite1,phenomenon2,phenomenon0)
P4-calibrated(instrument1)	A4-turn_to(satellite1,phenomenon3,phenomenon0)
P5-pointing(satellite1,phenomenon1)	A5-take_image(satellite1,phenomenon1,instrument1,thermograph0)
P6-pointing(satellite1,phenomenon2)	A6-turn_to(satellite1,phenomenon2,phenomenon1)
P7-pointing(satellite1,phenomenon3)	A7-take_image(satellite1,phenomenon2,instrument1,thermograph0)
P8-have_image(phenomenon1,thermograph0)	A8-turn_to(satellite1,phenomenon3,phenomenon2)
P9-have_image(phenomenon2,thermograph0)	A9-take_image(satellite1,phenomenon3,instrument1,thermograph0)
P10-have_image(phenomenon3,thermograph0)	

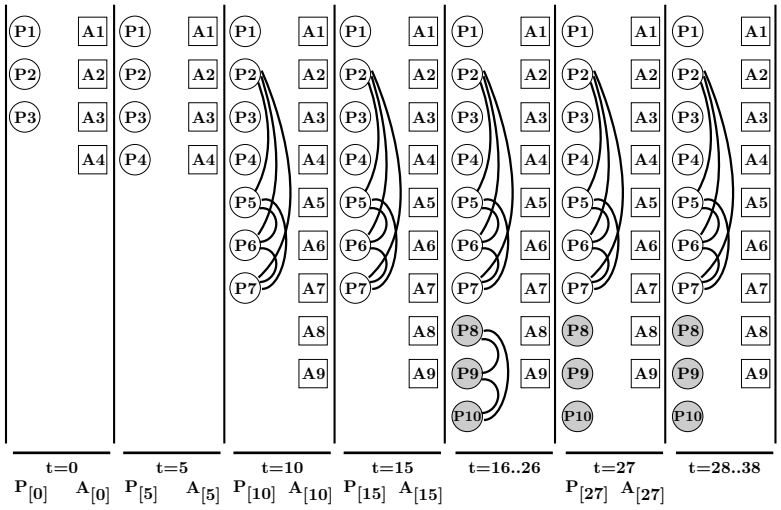


Figure 14: Outline of the temporal planning graph for the `satellite` problem (based on the temporal graph of Figure 7). Shaded propositions represent the problem goals, non-pairwise mutex at time 27. Mutex relations between propositions are represented by thick lines. For simplicity, *inverse* actions of `turn_to` (for instance, `turn_to(satellite1,phenomenon0,phenomenon1)` or `turn_to(satellite1,phenomenon0,phenomenon2)`) are not represented.

and 3×`take_image`) and has a makespan of 38 time units (see outline of the temporal planning graph in Figure 14). During the second stage, the planning graph is extended until time 27, when all the problem goals are present and non-pairwise mutex. At this point, the third stage starts the plan extraction. In this problem all possible pairs of actions are mutex and, consequently, only one action can be planned at each level (for simplicity we will assume actions are selected in increasing order starting with `A1`). The first selected action is `A5` at time 26 to satisfy `P8`. Next, TPSYS plans actions `A2` and `A1` to satisfy the conditions of `A5`. Afterwards, TPSYS attempts to plan `A7` for goal `P9`, but no feasible plan is found. The backtracking process then attempts to plan `A9` for goal `P10` but, again, unsuccessfully. Therefore, because no feasible plan can be found when `A5` is planned at level 26,

the search process backtracks to time 26 and attempts all possible permutations of actions `A5`, `A7` and `A9`. This is the **first** indication of inefficiency: a lot of effort is wasted planning nearly identical actions. It is clearly irrelevant the order in which the phenomenon images are taken in this problem. Furthermore, regardless of the order of the phenomenon images, it is impossible to find a feasible plan of duration 27.

Since no plan is found at level 27, the planning graph is extended until level 28 and the search is restarted from scratch. This is the **second** indication of inefficiency: actions planned in previous search iterations are not reused in this new search iteration. Specifically, it would be very advantageous if the subplan for taking one phenomenon image was reused in the new search iteration without having to compute the subplan once again. For example, if the planner would fix the set of actions {`A1,A2,A5`} for `phenomenon1`, then `A7`, `A9` and so on, the problem would be much simpler to solve as long as new search iterations are executed. However, TPSYS does not take advantage of previous planned actions, which makes it commit the same search failures but at a search level that is one level deeper.

Following our analysis, we can find a **third** indication of inefficiency due to chronological backtracking: when a proposition is no longer true at a level, the algorithm discards that choice and simply backtracks (there is no any attempt to support that condition through the insertion of new actions). The main inconvenience of this behaviour is that correct plans might be discarded and a lot of effort wasted. In our example, let us suppose that `satellite1` is required to point at `phenomenon2` after taking an image of `phenomenon1`. In this case, it would be preferable to plan `turn_to(satellite1,phenomenon2,phenomenon1)` —extending the graph if necessary— rather than discarding the actual plan and backtracking.

In summary, the first inefficiency forces the consideration of all possible permutations of actions to satisfy the goals at each level. The second and third inefficiencies force us to restart a blind search from scratch from each new level (again taking into account all possible permutations of actions). Although these inefficiencies are not exclusive to TPSYS, and also occur in Graphplan-based search planners, their influence is more noticeable in a temporal planning approach. In the previous example, the Graphplan search would only reach level 7 in the graph, whereas search in TPSYS reaches level 38, thus repeating the same inefficiencies over and over at each level.

7.2.2 Outline of the new search method

We present a new search that works in two stages (see Figure 15). First, a backward search generates an initial relaxed plan from the information of the planning graph to be used as a *skeleton* of the final plan. Second, a forward-chaining process allocates the execution time of the actions in the relaxed plan by means of a non-complete heuristic process. During this second stage, actions are only definitively allocated in the plan when they are applicable and are not mutex with each other. This search can be outlined as follows:

- **Stage 3.1** progressively creates a relaxed plan extracting actions from the temporal planning graph that results from the second stage. Actions in this relaxed plan, which can be mutex, support all the problem goals. However, they are not allocated in time, so there is not yet a commitment on their execution time.

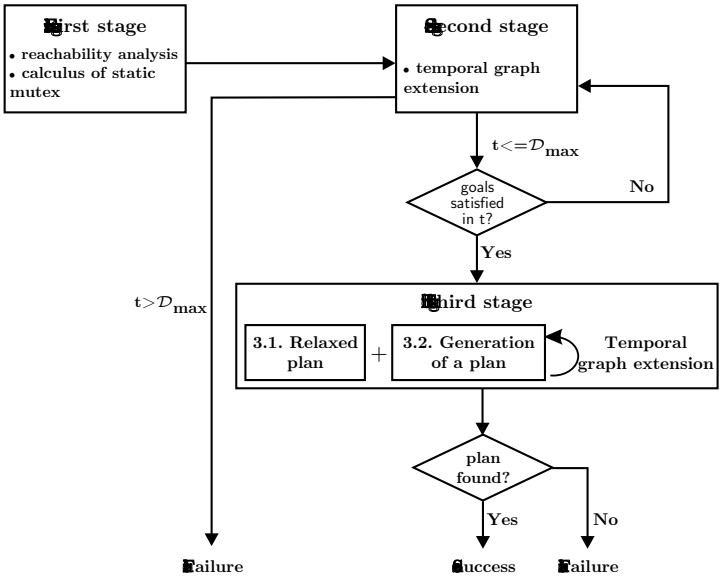


Figure 15: New structure of TPSYS when using a least-commitment heuristic search in the third stage.

- **Stage 3.2** allocates the actions of the relaxed plan in time. This stage performs a progressive heuristic process over the relaxed plan and the information contained in the planning graph. Throughout the allocation process, subgoals may become unsupported, in which case new actions are planned to support the goals again and repair the mutex interactions. Inserting new actions in the plan usually implies increasing the total plan duration, so it becomes necessary to extend the planning graph. Unlike the search based on Graphplan, where the graph is extended at the second stage, the graph is extended when required in this new search method. This modifies the third stage of TPSYS, which is no longer executed in an interleaved way with the second stage (see Figure 15).

7.2.3 Stage 3.1. Generation of an initial relaxed plan

This stage generates an initial relaxed plan from the information of the temporal planning graph to be used as the basis of the final plan.

Definition 16 (Relaxed plan) *A relaxed plan* Π *is defined 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 nor commitment on their start time are considered.*

A relaxed plan always contains two fictitious actions with no duration called `IS` and `FS`, which represent the first and the last action of the plan, respectively. `IS` achieves the propositions of the initial state, whereas `FS` requires the problem goals. Π is generated

similarly to the relaxed solution plan in the FF planner (Hoffmann, 2000; Hoffmann & Nebel, 2001) with the exception that our planner handles durative actions.

1. $GTS \leftarrow \mathcal{F}_s$ *// obligatory propositions*
2. $\Pi \leftarrow \{\mathtt{IS} \cup \mathtt{FS}\}$ *// obligatory actions*
3. **while** $GTS \neq \emptyset$
4. extract p from GTS
5. **if** p is not satisfied in Π
6. a $\leftarrow$arg min(*number of mutex that* $\mathtt{a_i}$ *imposes in* Π), $\forall \mathtt{a_i}$ that supports p
7. **if** p is an obligatory proposition $\wedge$ a is the only action supporting p
8. mark a as obligatory in Π
9. mark $\{SCond(\mathtt{a}) \cup Inv(\mathtt{a}) \cup ECond(\mathtt{a})\}$ as obligatory in Π
10. $\Pi \leftarrow \Pi \cup \{\mathtt{a}\}$ *// no commitment on the start time of* a *yet*
11. $GTS \leftarrow GTS \cup \{SCond(\mathtt{a}) \cup Inv(\mathtt{a}) \cup ECond(\mathtt{a})\}$

Figure 16: Algorithm for the generation of an initial relaxed plan Π.

Figure 16 describes the algorithm for the generation of an initial relaxed plan. The algorithm uses the data structure GTS to store the subgoals to be satisfied. GTS is initialised with the problem goals (step 1). The algorithm inserts actions to support the goals of GTS starting from the last level of the temporal graph. Step 6 selects the action which imposes the lowest number of mutex in Π. In problems with multiple resources, this selection tends to homogeneously use as many resources as are available, provides more information to the relaxed plan and allows us to improve the plan quality by overlapping a higher number of actions. Finally, the action is inserted into Π (step 10) and its conditions into GTS (step 11) while the algorithm continues until GTS gets empty. There are two interesting concepts introduced in the algorithm: *obligatory proposition* and *obligatory action*.

Definition 17 (Obligatory proposition) *A proposition* p *is obligatory iff* p *holds in any solution plan for the problem.*

Definition 18 (Obligatory action) *An action* a *is obligatory iff* a *is the only action that supports an obligatory proposition.*

These two concepts are very helpful when constructing the plan as they determine which propositions and actions must always be present in a plan. The presence of obligatory propositions/actions in the plan will never be affected by the modifications produced in the plan, that is, by the insertion or deletion of other actions. Obviously, $\mathcal{F}_s$, IS and FS are obligatory. The verification of obligatory propositions and actions is done in step 7 of the algorithm, where a and its conditions are marked as obligatory if a is the only action that supports an obligatory proposition p (steps 8–9).

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 (see Lemma 1). 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

1. `set_of_plans` $\leftarrow \Pi$, generated in Algorithm of Figure 16
2. **while** `set_of_plans` $\neq \emptyset$
3. extract the lowest cost plan Π_i from `set_of_plans`
4. **if** ($\mathcal{F}_s$ is satisfied in $Alloc_i \wedge Relax_i = \emptyset$)
5. success *// plan found*
6. **else**
7. **if** $\forall \mathtt{a_j} \in Alloc_i \mid \exists \mathtt{p_j} \in \{SCond(\mathtt{a_j}) \cup Inv(\mathtt{a_j}) \cup ECond(\mathtt{a_j})\}$ that is not satisfied
8. insert new plans into `set_of_plans` to satisfy $\mathtt{p_j}$
9. **else**
10. $\mathtt{a} \leftarrow \arg\max(\textit{allocation priority}), \forall \mathtt{a_j} \in \mathtt{Relax_i}$ to start in $\mathtt{time\_of\_execution_i}$
11. **if** a is mutex in $Alloc_i$
12. **if** a is non-obligatory
13. remove a from $Relax_i$
14. **else**
15. postpone start time of a in $Relax_i$
16. **else**
17. **if** a is applicable
18. allocate a in $Alloc_i$ at $\mathtt{time\_of\_execution_i}$
19. **else**
20. insert new plans into `set_of_plans` to make a applicable
21. update $\mathtt{time\_of\_execution_i}$

Figure 17: Algorithm for planning and allocating the actions.

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 requires postponing the allocation of actions, and/or planning new actions to solve the unsupported (sub)goals.

7.2.4 Stage 3.2. Planning and allocation of actions

This stage performs two tasks: i) allocating the actions of the relaxed plan, and ii) planning new actions to solve unsupported goals. This is done incrementally, generating the plan in a forward way. 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: $Relax_i$ and $Alloc_i$. $Relax_i$ contains the actions which have not yet been allocated, and so they can be removed from Π_i. $Alloc_i$ contains the actions which have been allocated in time and will never be removed from Π_i. Initially, $Relax_i$ contains all the actions in Π_i (the initial Π_i is the relaxed plan computed in stage 3.1) and $Alloc_i$ is empty. This stage ends once $Relax_i$ gets empty, obtaining the actions of the plan which support all the problem goals in $Alloc_i$.

The intuitive idea is to move forward in time, simulating the real execution of Π_i, and progressively taking care of the actions which can start their execution (see algorithm in Figure 17). The current time of execution in Π_i, $\mathtt{time\_of\_execution_i}$, 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 and $Relax_i$ is empty, the algorithm terminates with success (steps 4–5). If any action in $Alloc_i$ has unsatisfied conditions, the algorithm inserts

new actions (steps 7–8), generating new plans (branching point). Once all the conditions in $Alloc_i$ are supported, the algorithm studies the actions in $Relax_i$.

The algorithm selects the action a with the maximal priority to be allocated in time (step 10). If a is mutex with actions in $Alloc_i$ and it is non-obligatory, a is removed from $Relax_i$ (step 13), delaying the achievement of its goals to a future time of execution. The reason for removing 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 15). If a is not mutex and applicable, a is allocated in time (step 18). Although there exist alternative actions to a, if a can be allocated, those actions are not considered in Π_i. This step is the first indication of loss of completeness. Finally, if a is not applicable, the branching point of step 20 inserts new actions, generating new plans, to make a applicable.

The branching points of the algorithm are in steps 8 and 20, where it becomes necessary to insert new actions in new plans to support unsatisfied conditions. For each new action $\mathtt{a_j}$ supporting a condition, a new plan Π_j with action $\mathtt{a_j}$ marked as obligatory in Π_j is generated (and inserted into set_of_plans). Note that action $\mathtt{a_j}$ is not allocated in time, but inserted with its earliest start time extracted from the temporal graph. This is part of the least-commitment technique performed in the allocation of actions when they are inserted into the plans.

The algorithm has two important points of selection: steps 3 and 10. Step 3 selects the plan Π_i with the lowest cost from set_of_plans, where the cost is heuristically estimated as follows:

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

The cost of a plan Π_i consists of the sum of the cost due to $Alloc_i$ and $Relax_i$. These costs are calculated from the following factors:

- $unsup(\mathtt{a}, Alloc_i)$ is an estimation of the number of actions necessary to solve the unsupported conditions of a in $Alloc_i$ at the current $\mathtt{time\_of\_execution}_i$. This value is calculated from the information of the temporal graph, ignoring the delete effects of the actions (as in the FF heuristic).

- $offset(\mathtt{a}, Alloc_i)$ is an estimation of the temporal offset necessary for action a to be executed without mutex in $Alloc_i$. This value is calculated from the information of $Alloc_i$, checking the earliest start time at which a can be allocated without conflicting with other allocated actions. Intuitively, $offset$ gives us an idea about the complexity of allocating each action in $Alloc_i$.

- $add(\mathtt{a}, \mathtt{FS})$ indicates the number of conditions of FS that a supports.

- $del(\mathtt{a}, \mathtt{FS})$ indicates the number of conditions of FS that a deletes.

- $PA\_mutex(\mathtt{a}, \mathtt{FS})$ indicates the number of mutex between the conditions of FS and action a. This value is calculated from the information about mutex stored in the temporal graph. Intuitively, $PA\_mutex(\mathtt{a}, \mathtt{FS})$ represents the impossibility of simultaneously having the conditions of FS and a.

- $dur(\Pi_i)$ and $dur(\mathtt{a})$ indicate the duration (makespan) of plan Π_i and action a, respectively.

The coefficients $\alpha, \beta, \gamma, \zeta, \theta, \lambda, \mu \geq 0$ because they have a positive impact on the cost of the plan. On the contrary, $\delta \leq 0$ because it has a negative impact on the cost[6].

The point of selection in step 10 selects the action a with the maximal priority to be allocated in $Relax_i$ at the current $\mathtt{time\_of\_execution}_i$. This priority is heuristically estimated as follows:

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

The allocation priority of action a depends on the following local factors:

- $unsup(\mathtt{a}, Alloc_i)$ and $dur(\mathtt{a})$ are defined as above.

- $succ(\mathtt{a}, Relax_i)$ indicates the number of direct successors of a in $Relax_i$. This value is calculated from the actions in $Relax_i$. Intuitively, *succ* indicates the number of actions that depend on a because these actions require any effects of action a (*causal links*).

- $meetsucc(\mathtt{a}, Relax_i)$ indicates the number of director successors of a in $Relax_i$ that can start immediately after a ends; i.e *meets* relation. This value is calculated from the actions in $Relax_i$.

The factors *succ* and *meetsucc* indicate the importance of a to its successor actions in $Relax_i$. Intuitively, the more successors action a has, the more relevant a is in the plan. Similarly, the successor actions that meet with a indicate the number of actions that can be executed without mutex after a. Note that *succ* and *meetsucc* only consider direct successors and discard indirect successors that might not be executed in the plan. The coefficients $\sigma, \tau \geq 0$ because they imply the selection of an appropriate action. On the contrary, $\rho, \psi \leq 0$ because they imply that the action is not yet *promising enough* to be allocated[7].

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.

6. The values of the coefficients are used to normalise the estimations. The values experimentally calculated and currently used in TPSYS are: $\alpha = 2.75$, $\beta = 0.001$, $\gamma = 2.5$, $\zeta = 0.001$, $\delta = -2.5$, $\theta = 2.5$, $\lambda = 2.75$ and $\mu = 0.001$. Clearly, the values of the coefficients that deal with number of actions, propositions or mutex are higher than the values of the coefficients that deal with durations, because durations can be high.
7. The values of the coefficients are again used to normalise the estimations. The values currently used in TPSYS are: $\rho = -1.5$, $\sigma = 0.5$, $\tau = 0.5$ and $\psi = -0.05$.

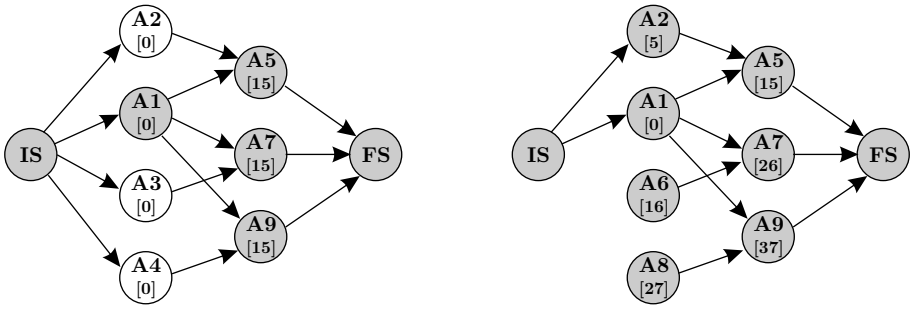


Figure 18: Plans for the `satellite` problem. The first (a) picture is the initial relaxed plan (values in brackets represent the earliest start times from the temporal graph). The second (b) picture is the solution plan (values in brackets represent the real start time of execution). Shaded actions represent obligatory actions.

In addition, as in other approaches of local search such as LPG (Gerevini & Serina, 2002a), we can implement different heuristic methods by setting the values of the coefficients, which in our case are statically calculated. 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 such as LPG. For instance, the static cost of inserting an action in LPG is based on the number of unsupported conditions, which cannot represent the real complexity of solving each condition. On the contrary, in our heuristics, the coefficient *unsup* deals with that complexity because it estimates the real cost of supporting it in the temporal graph. Therefore, LPG mainly focuses its estimations on the value of the coefficients, whereas our heuristic focuses on the estimation of the factors. The real values of the coefficients are simply used to normalise each factor.

This algorithm allows plans to be incrementally generated without discarding allocated actions and with no redundancy due to symmetry. Although algorithm of Figure 17 explores the complete space of actions to make actions applicable, the allocation priority imposes an order of execution and discards the rest of the feasible orderings (second indication of loss of completeness). For instance, in the `satellite` problem, when studying the actions "`take_image`" $\{\texttt{A5}, \texttt{A7}, \texttt{A9}\}$, the algorithm allocates one action and postpones the others. This avoids the complete exploration of all the permutations of `A5`, `A7` and `A9`, thus preventing the planner from generating symmetric plans.

7.2.5 An example of the incremental generation of a temporal plan

In order to illustrate the behaviour of the new search process, we will use the `satellite` problem. The search (stage 3.1) starts at time $t = 27$ when the extension of the temporal graph terminates (see Figure 14), and it generates the relaxed plan which is shown in Figure 18-a. Actions {`IS,FS,A1,A5,A7,A9`} are obligatory because they are necessary in any plan to take images of `phenomenon1`, `phenomenon2` and `phenomenon3`. Actions $\{\texttt{A2}, \texttt{A3}, \texttt{A4}\}$

are non-obligatory because it is possible to turn from `phenomenon0` towards any other phenomenon. This relaxed plan prevents the planner from starting from an empty plan and provides a good outline of the final plan. In fact, in this problem, 5 out of the 7 actions in the relaxed plan are present in the final plan.

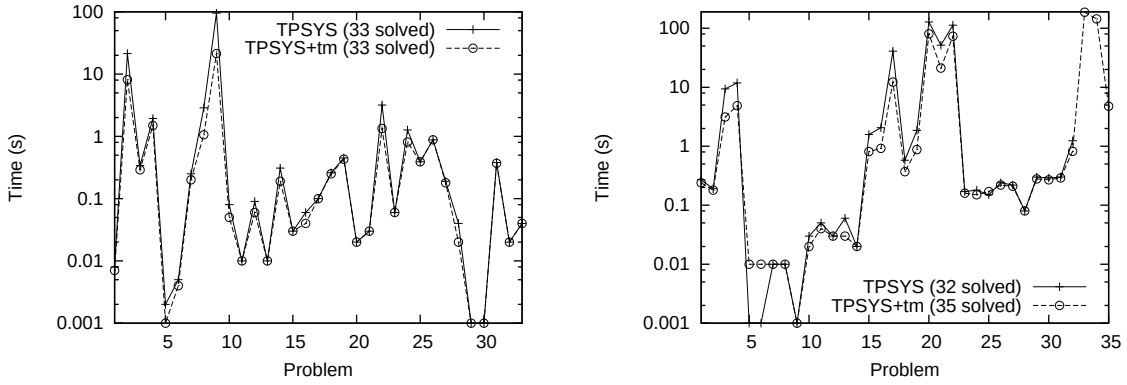
Stage 3.2 starts with the relaxed plan Π_{i} in Figure 18-a and $\texttt{time\_of\_execution}_i = 0$. Let us suppose that the action to be instantiated (and inserted into $Alloc_i$) from the set {`A1,A2,A3,A4`} is `A1`. Consequently, non-obligatory actions {`A2,A3,A4`} which can be executed at time 0 are removed from $Relax_i$ because they are mutex in $Alloc_i$. The following $\texttt{time\_of\_execution}_i$ in $Relax_i$ when actions can be instantiated is 15. Let us suppose that action `A5` is selected. `A5` is not applicable (its condition `pointing(satellite1,phenomenon1)` is unsupported). Instead of discarding the current plan and studying the nearly identical actions `A7` and `A9` with similar unsupported conditions, the algorithm tries to solve that condition by generating as many new plans as actions supporting the condition. These actions are inserted into the new plans as obligatory actions and with their earliest start time. One of the plans Π_{j} will contain action `A2` which can start at time 0. `A2` is mutex in $Alloc_j$, but it is obligatory. Therefore, `A2` is not removed but postponed until $\texttt{time\_of\_execution}_j = 5$ when it is finally allocated. Then, in $\texttt{time\_of\_execution}_j = 15$, action `A5` is allocated into $Alloc_j$ (and the first problem goal is achieved at time 16). The remaining actions {`A7,A9`} are allocated in a similar way in times 27 and 37, respectively. The incremental generation of the plan continues until all the problem goals (conditions of action `FS`) are supported. This situation happens in time 38, achieving the plan depicted in Figure 18-b (the same optimal plan detailed in Figure 9).

8. Experimental Results

In this section we present the experimental results on the planning approach of TPSYS. First, we evaluate the original Graphplan-based search *vs.* the search extended with temporal memoization. Second, we study the impact of the different heuristics presented in section 7.1.5 to reduce the search space. Third, we compare the original Graphplan-based search *vs.* the search based on least-commitment and heuristic techniques presented in section 7.2. Finally, we compare the search based on least-commitment and heuristic techniques with other state-of-the-art temporal planners.

All the experiments belong to different problems which were used in the International Planning Competitions (IPC–1998[8], IPC–2000[9] and IPC–2002[10]). The problems belong to the domains `blocks`, `ferry`, `gripper`, `bulldozer`, `elevator`, `logistics`, `driverlog`, `depots`, `zenotravel`, `satellite`, `rovers`, etc. Specifically, we selected 40 problems for IPC–1998, 50 for IPC–2000 and 62 for IPC–2002, which are grouped together in the figures. The problems of IPC–1998 and IPC–2000 were redefined to fit them into the non-conservative model of PDDL2.1. Obviously, this redefinition does not entail any lack of expressiveness because PDDL2.1 subsumes PDDL. All the experiments were run on a Pentium IV 2GHz with 512Mb of memory and censored after 300 seconds.

8. IPC–1998 domains are available in `ftp://ftp.cs.yale.edu/pub/mcdermott/domains`
9. IPC–2000 domains are available in `http://www.cs.toronto.edu/aips2000`
10. IPC–2002 domains are available in `http://www.dur.ac.uk/d.p.long/IPC`



(a) - IPC–1998 (b) - IPC–2000

Figure 19: Comparison of the execution time of TPSYS *vs.* TPSYS with temporal memoization (*tm*) in problems of IPC–1998 and IPC–2000.

8.1 Evaluation of temporal memoization in the Graphplan-based search

The experiments of this section focus on the evaluation of the performance of TPSYS when finding optimal plans *w.r.t.* makespan. Therefore, we will compare the original version of TPSYS with an extended version that uses the implementation of the temporal memoization. Figure 19 shows the results of this comparison.

The results indicate that the temporal memoization reduces the search space, that helps solve problems faster in IPC–1998 and IPC–2000, and solves more problems in IPC–2000. In the simplest problems, the search improvements are nearly inappreciable because the overhead in the memoization hardly compensates the redundancy in the search. However, in larger and more complex problems of IPC–2000, the improvements due to memoization are more significant, solving problems which were previously unsolvable (see Figure 19-b).

In general, the temporal memoization improves the behaviour of the search. However, the amount of information memoized when handling actions with duration is more limited than in a classical planning approach because the existence of mutex between actions planned at different levels prevents the algorithm from doing the memoization. This reduces the information memoized in temporal planning, making the improvements in a temporal approach less spectacular than in Graphplan.

8.2 Evaluation of heuristics to reduce the search space

8.2.1 Analysis of the methods for the reduction in the search depth

The three methods presented in section 7.1.5 are: M1 (Reduction in the number of levels generated in the 2nd stage), M2 (Reduction in the number of levels explored in the 3rd stage), and M3 (Skipping levels in the 3rd stage after a failed search). The three of them allow us to reduce the number of levels *available* in the search. Figures 20 and 21 show the comparison of these methods in problems of IPC–1998 and IPC-2000, respectively. These

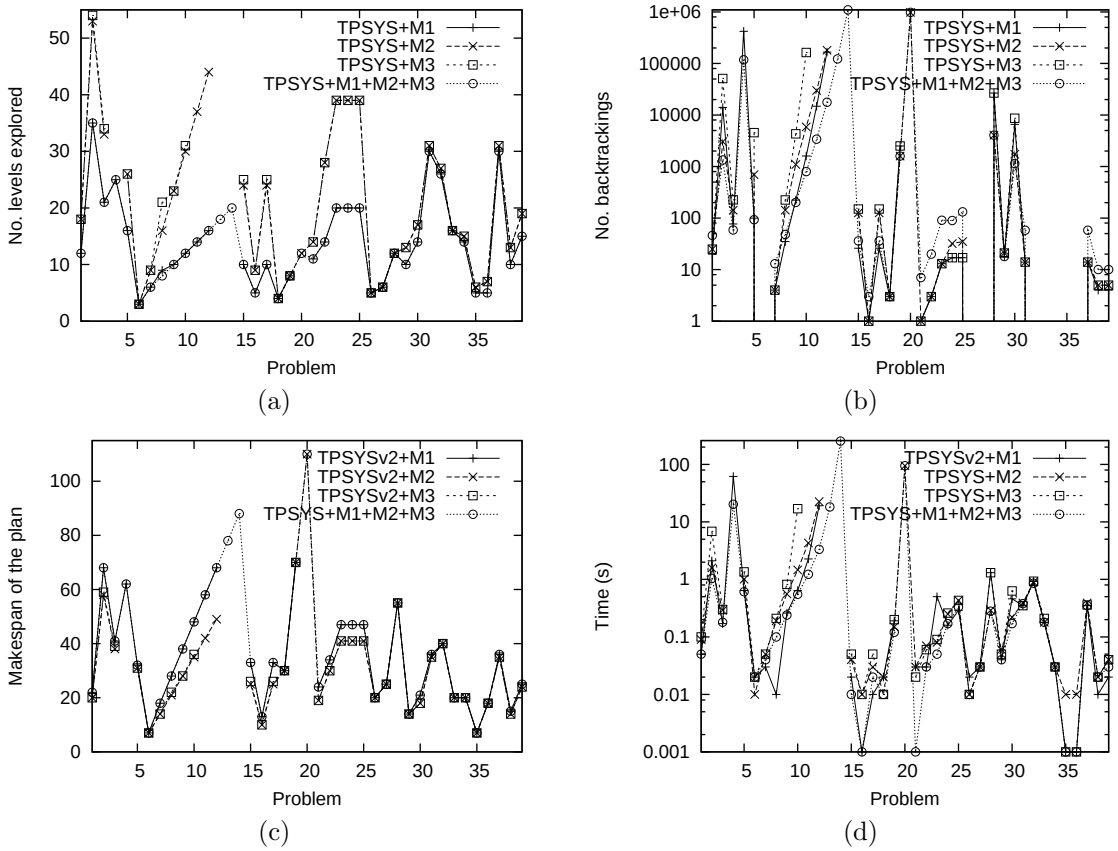


Figure 20: Comparison of the heuristic methods M1, M2 and M3 implemented in TPSYS in IPC–1998.

methods affect the number of levels explored and, consequently, the number of backtrackings in the search.

Method M1 usually explores the lowest number of levels (see Figures 20-a and 21-a). This has direct consequences in the number of backtrackings (see Figures 20-b and 21-b). As a result, method M1 solves problems in better execution times (see Figures 20-d and 21-d). However, the makespan of the plans is better in methods M2 and M3 (see Figures 20-c and 21-c). The reason for this is quite simple: the temporal graph does not contain enough levels where mutex can end. This increments the distance between the expansion levels, postponing the execution of the actions and incrementing the makespan of the plans. Method M2 provides the best tradeoff between the makespan of the plan and the execution time. In IPC–2000 (see Figure 21), such a method solves many problems which were unsolvable by the other methods. Unlike M1, M2 performs the search in a temporal graph that contains all the reachable levels, so the makespan of the plan tends to be shorter. Method M3 solves the lowest number of problems, especially in IPC–2000. However, the

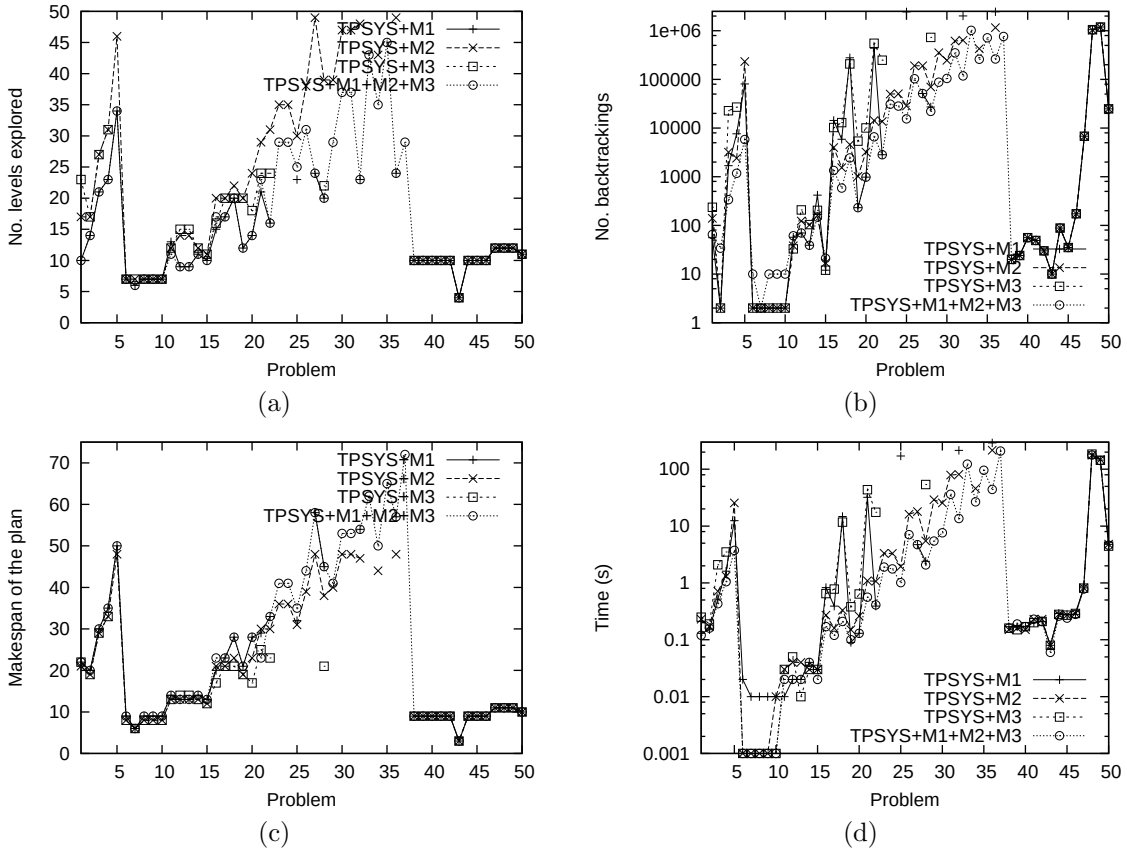


Figure 21: Comparison of the heuristic methods M1, M2 and M3 implemented in TPSYS in IPC–2000.

makespan of the plans is similar to plans of M2. It relies on the complete temporal graph which helps action mutex end earlier, thus reducing the makespan of the plans. Finally, we have combined the three methods in the version TPSYS+M1+M2+M3. This combination is very useful and provides good execution times in most of the problems, especially in IPC–2000. Moreover, the combination of the three methods solves problems that no other method can solve. The main drawback of this combination relies on the makespan of the plans, which is now worse (similarly to method M1 because the number of levels in the temporal graph is the same).

8.2.2 Analysis of the heuristics for the reduction in the branching factor

The six heuristics defined in section 7.1.5 are: H1 (min-duration), H2 (min-EET), H3 (max-conds-IS), H4 (min-duration-plan), H5 (min-acts-plan) and H6 (random). They allow us to reduce the branching factor by selecting the actions to satisfy the goals at each level. Unlike methods for reducing the search depth, these heuristics hardly modify the number of levels to explore. Therefore, Figures 22 and 23 only show the makespan of the plans and the execution times. Figures 22-a,b and 23-a,b show the results for the heuristics H1–H3 based on invariable information of the actions, whereas Figures 22-c,d and 23-c,d show the results for the heuristics H4–H6. These heuristics provide different choices of action selection depending on the state where they are applied.

Heuristics H1–H3 show no significant differences which indicates that one heuristic outperforms the others. Although heuristics H4 and H5 are more expensive to calculate, they are more precise and solve more problems. Heuristic H6 obtains the longest makespan plans. However, it is important to highlight that H6 has short execution times and solves problems which are unsolvable by the other heuristics.

8.2.3 Comparison of the heuristic approaches to reduce the search space

Table 2 shows the number of problems solved for each method/heuristic. The methods for reducing the search depth solve more problems, and the alternative which solves the greatest number of problems is the combination of M1+M2+M3. Although M2 solves the greatest number of problems, M1 has the best execution times. Moreover, M1 is useful in any temporal approach which constructs a temporal planning graph. This allows us to apply this method also in the heuristic search based on least-commitment presented in section 7.2. As for the reduction in the branching factor, the heuristics based on information about the state of search (H4 and H5) behave better than the heuristics based on invariable information (H1–H3). Although the random heuristic H6 generates plans that are longer than the rest of the heuristics, it solves more problems and, generally, with the best execution times.

Since none of the heuristic approaches is complete-preserving, the *safest* approaches are the methods that reduce the search depth. The methods M1–M3 discard some levels in the search, but they do not discard any action in the goal satisfaction. On the contrary, the heuristic functions H1–H6 discard some actions when satisfying the goals. Consequently, if the search starts from one level where a plan can be found, any of the methods M1–M3 will find it, but the heuristic functions H1–H6 might not find it.

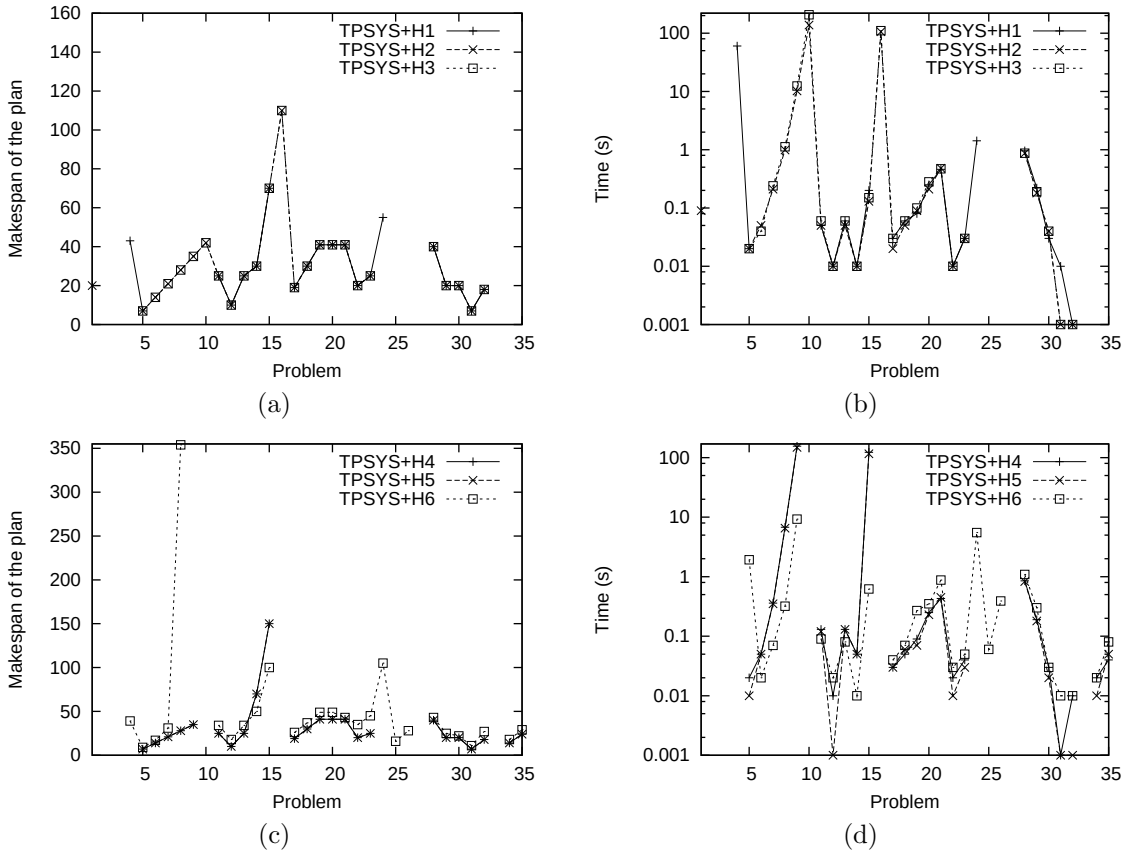


Figure 22: Comparison of the heuristic functions H1–H6 implemented in TPSYS in problems of IPC–1998.

	Reduction in the search depth				Reduction in the branching factor					
Problems	M1	M2	M3	M1+M2+M3	H1	H2	H3	H4	H5	H6
IPC–1998	36	36	33	39	20	25	24	24	24	27
IPC–2000	40	47	35	50	31	29	29	32	32	33
Total:	76	83	68	89	51	54	53	56	56	60

Table 2: Number of problems solved for each heuristic (40 problems in IPC–1998 and 50 in problems of IPC–2000).

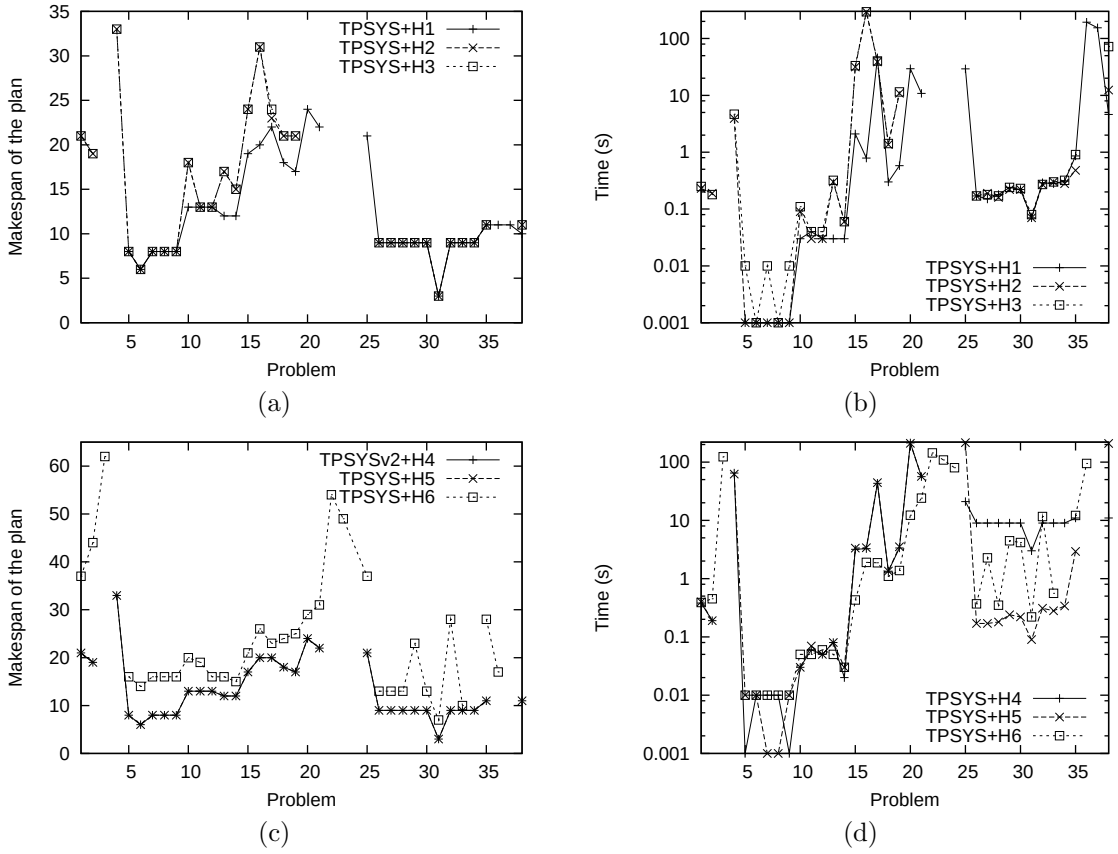


Figure 23: Comparison of the heuristic functions H1–H6 implemented in TPSYS in problems of IPC–2000.

8.3 Evaluation of Graphplan-based search *vs.* search based on least-commitment and heuristic techniques

In this section, we will compare the original search of TPSYS based on Graphplan and the search based on least-commitment and heuristic techniques (from now on TPSYS-LC) to evaluate the benefits of the new search.

8.3.1 Impact of the symmetry in TPSYS-LC

This experiment studies the viability of the least-commitment search of TPSYS-LC to overcome the main inefficiencies of the Graphplan backward search presented in section 7.2.1. We have used the well-known domains `ferry` and `gripper` from IPC–1998, and `elevator` and `logistics` from IPC–2000. These domains impose a high degree of symmetry and redundancy when handling cars, balls, passengers and packages, respectively. Moreover, the permutation of these objects provokes a combinatorial explosion which negatively affects the search.

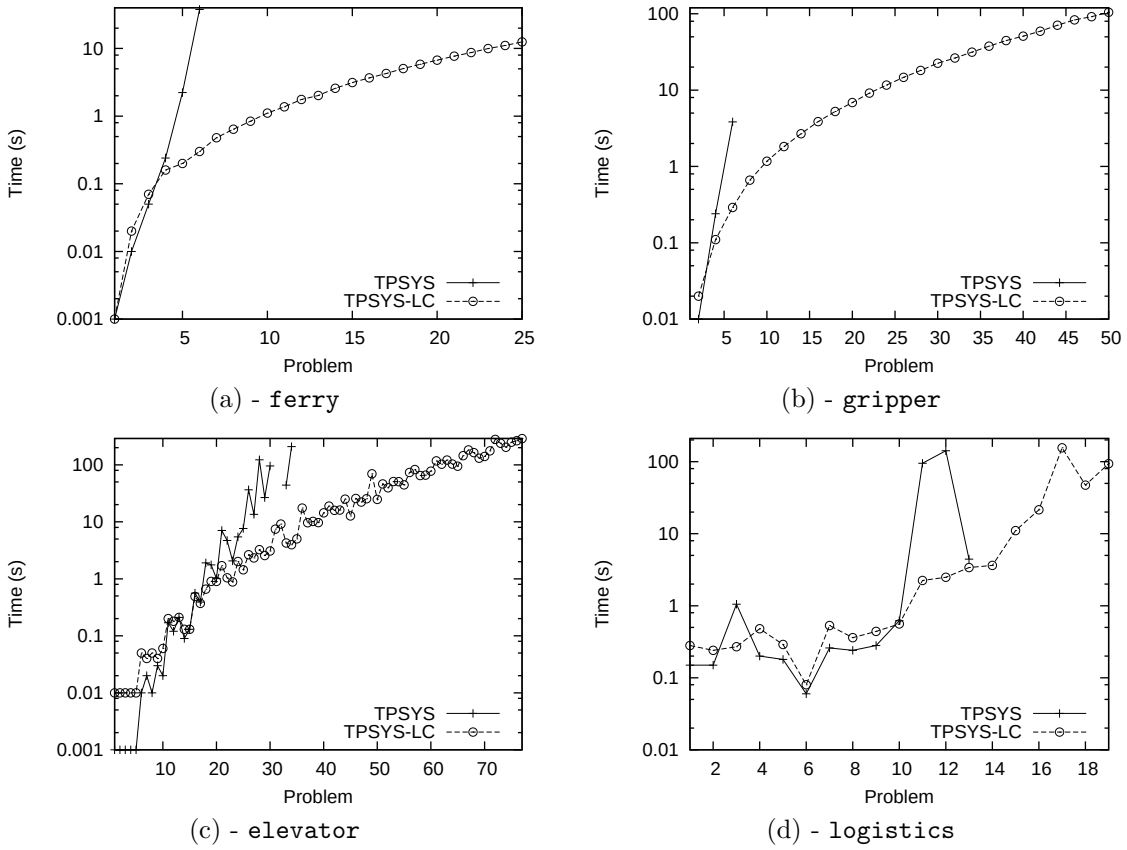
Figures 24-a,b show the results of the comparison between TPSYS and TPSYS-LC in the `ferry` and `gripper` domains, respectively. Although it is not surprising that the least-commitment search (TPSYS-LC) is faster than the Graphplan backward search (TPSYS), it is worth mentioning that TPSYS-LC scales up much better than TPSYS, especially in the `gripper` domain. TPSYS has difficulties in solving problems with more than 6 balls, but TPSYS-LC can easily solve problems with more than 50 balls. Moreover, TPSYS-LC found optimal plans without backtracking for all the problems. Figures 24-c,d show analogous results for the `elevator` and `logistics` domains. Again, TPSYS-LC solves more problems and the execution time is more scalable, especially in the `elevator` domain.

8.3.2 TPSYS-LC in more generic problems

This experiment deals with some problems of the *SimpleTime* track of IPC–2002. The complexity of IPC–2002 problems depends on several factors: exploitation of parallelism among actions, problem symmetry, number of problem goals, alternative solutions, etc. Figure 25 shows the results, depicting that TPSYS-LC solves 38 problems, while TPSYS only solves 10 problems. This indicates that both approaches still have difficulties in solving all the problems. On one hand, the difficulties of the Graphplan backward search arise as a result of the redundancy in the complete, blind search process. On the other hand, the difficulties of the least-commitment search arise 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. Currently, this is one of the main limitations of the least-commitment search.

8.4 Comparison of the search based on least-commitment and heuristic techniques with other state-of-the-art temporal planners

In this section, we focus on the quality of the plans generated by the least-commitment search *vs.* other planners in some problems of the *SimpleTime* track of IPC–2002. Although in temporal planning the most important criterion for quality is the makespan, we will also consider the number of actions of the plan as an additional criterion for the plan quality.



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

(c) - `elevator` (d) - `logistics`

Figure 24: Impact of the symmetry in the search approaches of TPSYS and TPSYS-LC in different domains.

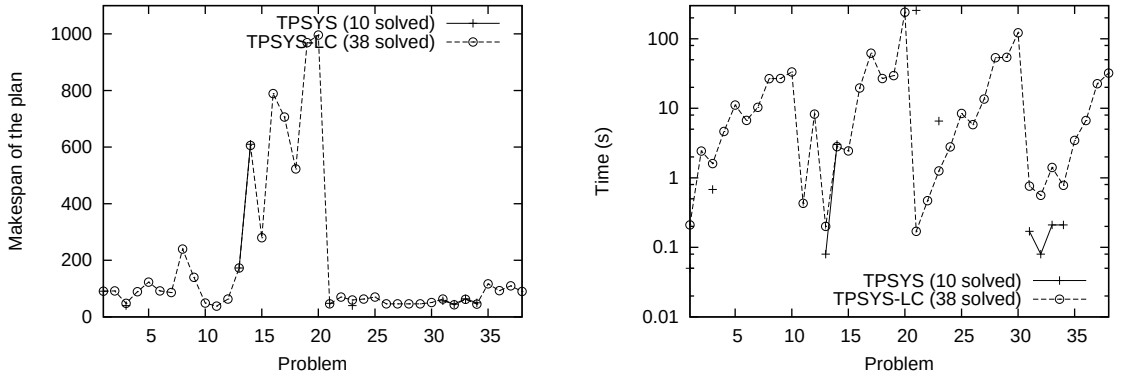


Figure 25: Comparison of TPSYS and TPSYS-LC in problems of IPC–2002.

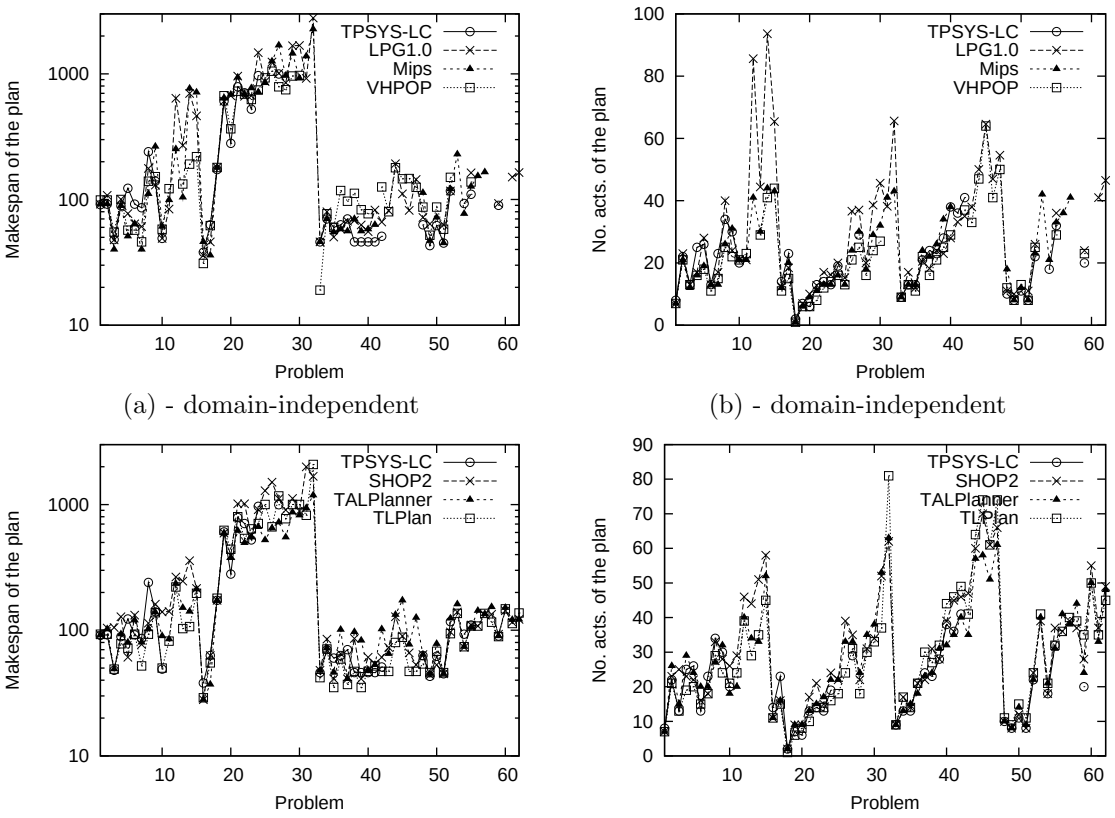
We have compared the quality of the plans of TPSYS-LC with other (both domain-independent and domain-dependent) temporal planners, such as LPG1.0[11] (Gerevini & Serina, 2002a), Mips[12] (Edelkamp, 2002), SHOP2 (Nau, Cao, Lotem, & Muñoz Avila, 1999; Nau, Muñoz Avila, Cao, Lotem, & Mitchell, 2001), TALPlanner (Kvarnstrom & Doherty, 2000), TLPlan (Bacchus & Kabanza, 2000) and VHPOP (Younes & Simmons, 2002). We have rerun the problems again in both LPG1.0 and Mips, while, for the rest of planners, we have used the results provided in IPC–2002[13].

Figures 26-a,b and 26-c,d show the results of this comparison for the domain-independent and domain-dependent planners, respectively. In general, the quality of the plans generated by TPSYS-LC is equivalent, and even better, than the rest of the domain-independent planners. Moreover, if we analyse the number of actions, TPSYS-LC generates plans with even fewer actions. The results when comparing TPSYS-LC with domain-dependent planners are similar, with the only exception of TLPlan, which usually generates shorter plans. In consequence, this shows that the least-commitment search implemented in TPSYS-LC is competitive with other state-of-the-art planners, both domain-independent and domain-dependent ones (Garrido & Onaindía, 2003).

If we focus only on domain-independent planning, Figure 27 shows a more detailed comparison of TPSYS-LC, LPG1.0 and Mips[14] *w.r.t.* the makespan of the plan and its execution time in problems of `driverlog`, `zenotravel` and `satellite` domains. Figure 27-a shows the results in the `driverlog` domain. Here, the plans produced by TPSYS-LC are slightly longer than LPG1.0 and Mips plans. However, in the `zenotravel` and `satellite` domains (see Figures 27-c,e) the makespan of the plans of TPSYS-LC is shorter than LPG1.0 and Mips. As Figures 27-b,d,f show, both LPG1.0 and Mips are clearly faster than TPSYS-LC (they solve more problems in 300 s). Thus, the most important property of TPSYS-LC is its tradeoff between the quality of the plan generated and its execution time.

Evaluation of a search process in an *anytime* search The tradeoff of TPSYS-LC between plan quality and execution time is important, especially when it is compared with LPG, which has some similarities *w.r.t.* local and heuristic search. The main advantage of LPG is its speed, but its main disadvantage is its non-deterministic behaviour, which can generate very dispersing plans for the same problem. This drawback could be solved by means of an *anytime* search: the search is maintained beyond finding a plan, which is used as a lower bound while searching for other plans. Following this line, we have fixed a deadline of 300 s. for the execution time and selected the best plan of TPSYS-LC and LPG. The results of this experiment are depicted in Figure 28. This figure compares the

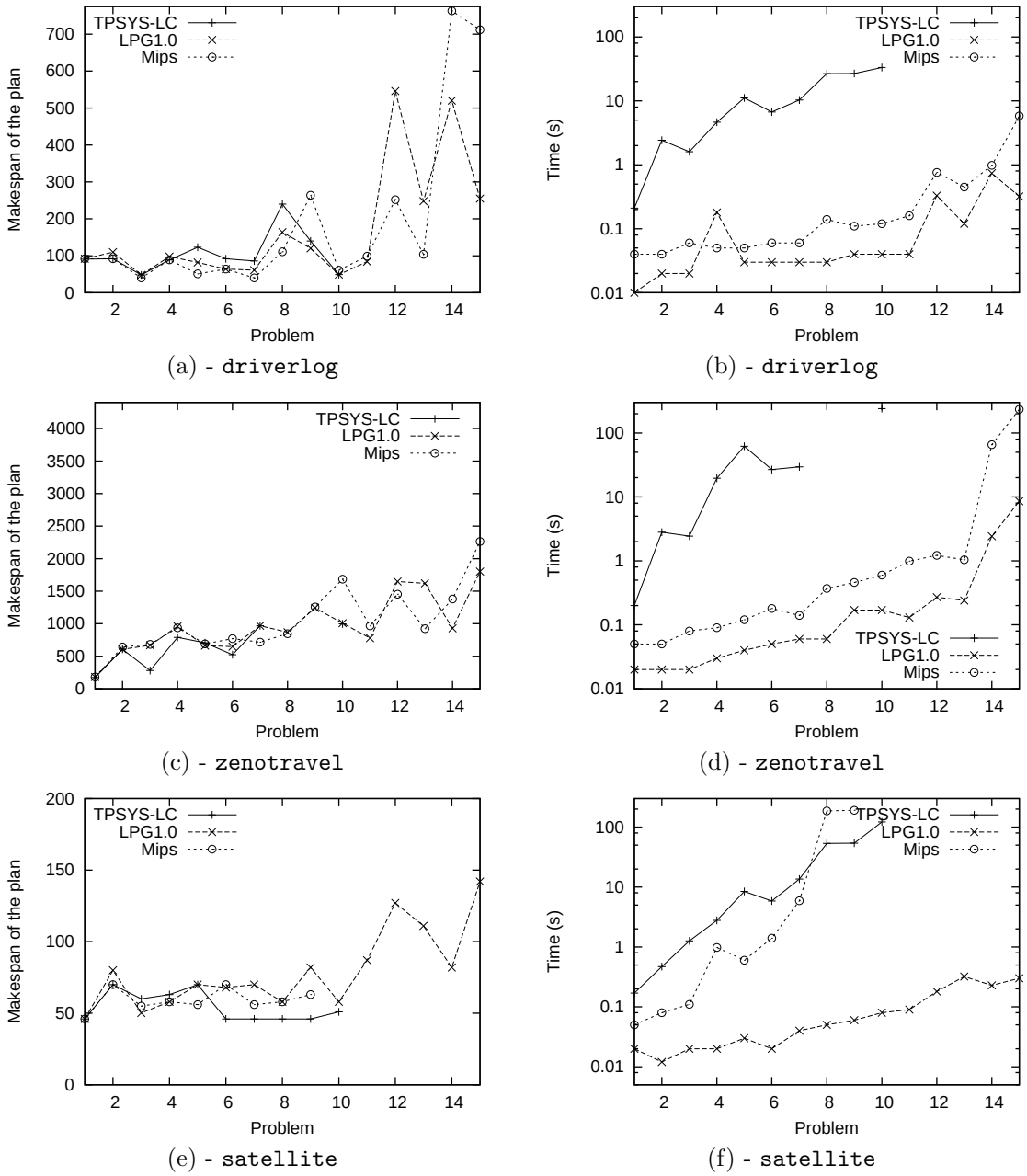
11. LPG is based on a non-deterministic local search and, consequently, it would not be fair to work with the plan from only one execution. In our experiments with LPG we have run each problem ten times and extracted the median values. We used the LPG1.0 version as provided in: `http://prometeo.ing.unibs.it/lpg`
12. We used the Mips version as provided in: `http://www.informatik.uni-freiburg.de/~mmips`
13. Although Sapa (Do & Kambhampati, 2001), TP4 (Haslum & Geffner, 2001) and IxTeT (Ghallab & Laruelle, 1994) also participated in IPC–2002, we do not have enough results in the *SimpleTime* track to be included in the comparison.
14. We have chosen these two planners as they handle temporal features and showed distinguished performance in the last IPC–2002.



(a) - domain-independent (b) - domain-independent

(c) - domain-dependent (d) - domain-dependent

Figure 26: Comparison of the quality (makespan and number of actions) of the plans generated by TPSYS-LC and other temporal planners in problems of IPC–2002.



(a) - `driverlog` (b) - `driverlog`

(c) - `zenotravel` (d) - `zenotravel`

(e) - `satellite` (f) - `satellite`

Figure 27: Results of the comparison between TPSYS-LC, LPG1.0 and Mips in problems of IPC–2002.

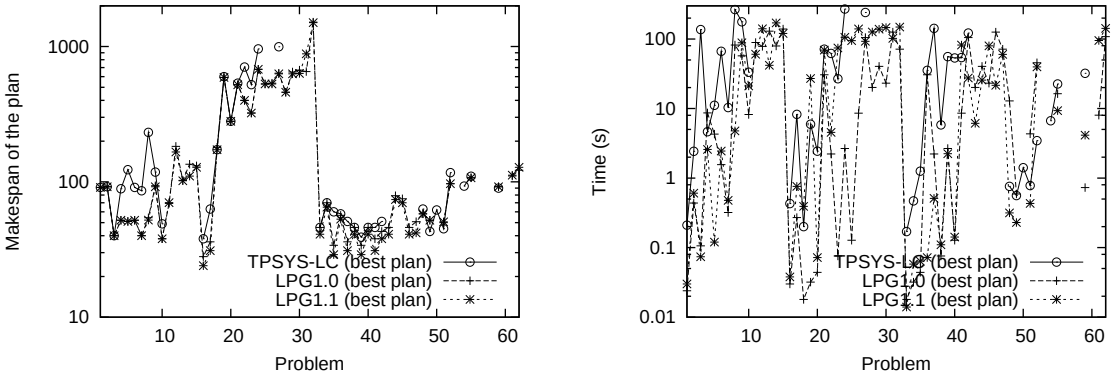


Figure 28: Comparison of the quality of the best plans generated by TPSYS-LC and the two versions of LPG in problems of IPC–2002.

best plans of TPSYS-LC with the best plans of LPG1.0 and LPG1.1[15] (Gerevini & Serina, 2002b). The main conclusions we can extract from Figure 28 are:

- The first plan that TPSYS-LC finds is the best plan found in 300 s. in most cases. Specifically, only 25% of the plans could be improved in 300 s. in TPSYS-LC. On the contrary, nearly 98% of the plans could be improved within the 300 s. deadline in both LPG1.0 and LPG1.1.

- LPG1.1 is still faster than TPSYS-LC, but when they generate plans of similar quality this difference is not so significant and TPSYS-LC is faster in a few problems (see Figure 28-b).

- The best plans of LPG1.1 are generally better than the plans of LPG1.0. This demonstrates that dealing with a non-conservative model of actions allows to improve the plan quality. However, LPG1.1 is slightly slower than LPG1.0 in some problems, which demonstrates that having a graph with temporal capabilities increases the cost of the planning algorithm.

The main disadvantage of an *anytime* search is that selecting an adequate deadline for the execution time tends to be quite difficult. Therefore, achieving a good tradeoff between plan quality and execution time is essential, and the way in which TPSYS-LC works seems to be quite appropriate in a temporal planning approach.

15. Unlike LPG1.0, LPG1.1 handles the non-conservative model of actions of PDDL2.1 internally (taking advantage of the initial conditions/effects) and it uses an *enriched* planning graph with temporal features, thus improving the calculus of its estimation functions.

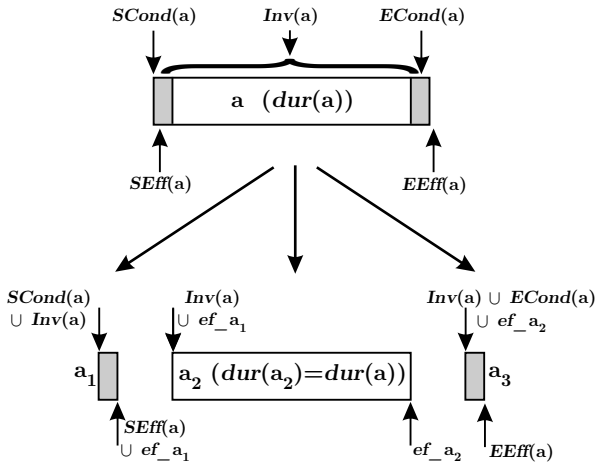


Figure 29: Division of a non-conservative durative action into three conservative actions.

9. Discussion

The temporal planning system described in this paper represents an approach for dealing explicitly with the non-conservative model of actions of PDDL2.1 in a Graphplan-based approach. However, other different alternatives could preprocess the planning domain, transforming it into a conservative model of actions or even into a classical model of actions without duration. Both alternatives are based on splitting each durative action into a collection of actions to handle *at start* and *at end* conditions/effects.

The first alternative splits each durative action into two instantaneous actions (which represent the start and end points of the durative action) and one action with duration (which represents the course of the action) as indicated in Figure 29. Therefore, the problem could be solved by a temporal planner with the ability to manage instantaneous actions. These three new actions will have neither *at start* effects nor *at end* conditions. Thus, a durative action $\mathtt{a}$ is divided into:

- $\mathtt{a}_1$, with $Pre(\mathtt{a}_1) = \{SCond(\mathtt{a}) \cup Inv(\mathtt{a})\}$, $Add(\mathtt{a}_1) = \{SAdd(\mathtt{a}) \cup \mathtt{ef\_a}_1\}$, $Del(\mathtt{a}_1) = SDel(\mathtt{a})$ and $dur(\mathtt{a}_1) = 0$.
- $\mathtt{a}_2$, with $Pre(\mathtt{a}_2) = \{Inv(\mathtt{a}) \cup \mathtt{ef\_a}_1\}$, $Add(\mathtt{a}_1) = \{\mathtt{ef\_a}_2\}$, $Del(\mathtt{a}_2) = \emptyset$ and $dur(\mathtt{a}_2) = dur(\mathtt{a})$.
- $\mathtt{a}_3$, with $Pre(\mathtt{a}_3) = \{Inv(\mathtt{a}) \cup ECond(\mathtt{a}) \cup \mathtt{ef\_a}_2\}$, $Add(\mathtt{a}_3) = EAdd(\mathtt{a})$, $Del(\mathtt{a}_3) = EDel(\mathtt{a})$ and $dur(\mathtt{a}_3) = 0$.

The inclusion of the *artificial* effects $\mathtt{ef\_a}_1$ and $\mathtt{ef\_a}_2$ of actions $\mathtt{a}_1$ and $\mathtt{a}_2$, respectively, guarantees that $\mathtt{a}_2$ will be generated after $\mathtt{a}_1$, and $\mathtt{a}_3$ after $\mathtt{a}_2$, simulating the behaviour of the original action $\mathtt{a}$. This way, during the search, action $\mathtt{a}_3$ can only be planned if $\mathtt{a}_2$ has been previously planned, and analogously, $\mathtt{a}_2$ can only be planned after planning $\mathtt{a}_1$. Although this transformation rarely modifies the behaviour of the temporal planner, the

main drawback is the 3× increment in the number of actions and 2× in the number of propositions in the domain, which by itself may be prohibitive in large domains. Moreover, if one goal of the problem is satisfied by $\mathtt{a}_1$ (i.e. $SAdd(\mathtt{a})$), only the action $\mathtt{a}_1$ would be planned (without needing to plan $\mathtt{a}_2$ nor $\mathtt{a}_3$), which would lead to an unreal situation in which only a part of the indivisible action a is executed. Consequently, the planner needs to include additional constraints to guarantee the execution of $\mathtt{a}_1$, $\mathtt{a}_2$ and $\mathtt{a}_3$ in an *atomic* way.

The second alternative is based on the semantic mapping described by Fox and Long (2001) and consists of splitting each durative action into a collection of simple actions without duration[16]. Therefore, the problem could be solved by any classical planner. The collection of actions includes two instantaneous actions (which represent the start and end points of the durative action) and a number of identical monitoring actions responsible for confirming the maintenance of invariants (Long & Fox, 2003). The monitoring actions can be achieved by requiring the no-op actions corresponding to the invariants of an action to be active in the interval between the start and end points of that action. Now, during search it is necessary to maintain the link between the actions representing the start and end points of a durative action, because none of them can be exploited without the other. In addition, it is necessary to manage the temporal constraints implied by the durations of the actions. As in the first alternative, the main drawback relies on doubling up the number of actions. However, this alternative can take advantage of the recent techniques used to improve the behaviour of Graphplan-based planners (Lopez & Bacchus, 2003; Sanchez Nigenda & Kambhampati, 2003; Zimmerman & Kambhampati, 2003).

In order to prevent the artificial *explosion* in the number of actions, we have opted to adapt the planner to deal explicitly with a non-conservative model of actions. This adaptation requires some modifications which have been presented throughout the paper, such as a more precise reasoning on mutex, an extension in the termination criterion for the temporal planning graph and other features that are dependent on the search approach used.

10. Conclusions through Related Work

Researchers in planning have made a great effort to extend and improve the capabilities of planners. If we focus on temporal planning, the last decade has seen many new works: O-PLAN (Currie & Tate, 1991), which integrates both planning and scheduling processes into a single framework, ZENO (Penberthy & Weld, 1994), IxTeT (Ghallab & Laruelle, 1994) and parcPLAN (El-Kholy & Richards, 1996), which cope with temporality on actions and temporal constraints, etc. Graphplan success has allowed the development of more modern temporal planners: TGP (Smith & Weld, 1999), whose ideas are very valuable in temporal environments, TP4 (Haslum & Geffner, 2001) and Sapa (Do & Kambhampati, 2001), which handle concurrent durative actions and use heuristic metrics to deal with resources in planning, LPG (Gerevini & Serina, 2002a) and Mips (Edelkamp, 2002). Both LPG and

16. This is the strategy that follows LPGP (Long & Fox, 2003). LPGP automatically transforms each non-conservative action into instantaneous actions, generates a classical planning graph as in STAN (Fox & Long, 1999b), and then searches a plan by means of a linear programming solver which guarantees the temporal constraints on the duration of the actions and the *atomicity* of the collection of actions.

Mips showed an outstanding performance in the last International Planning Competition, demonstrating that heuristic and local search are very useful in planning and especially in temporal planning. Least-commitment techniques have also been widely used in (partial order) planning, relying on the consistency of temporal constraints (IxTeT and HSTS, Muscettola, 1994), postponing the assignment of values to variables or the order of execution of actions, etc. (Weld, 1994). Moreover, Graphplan has also been augmented with least-commitment techniques by Cayrol, Reginer, and Vidal (2001). In Cayrol's work, the definition of mutex relationships between classical actions is relaxed, which allows for the achievement of the problem goals earlier in the planning graph.

The planner TPSYS presented in this paper builds on several of the previous works. First, TPSYS is inspired by the ideas of Graphplan and, specifically by TGP, and examines the general question of dealing with the non-conservative model of actions of PDDL2.1 in a Graphplan approach. However, the differences between TGP and TPSYS are important: i) TPSYS calculates the action/action and proposition/action static mutex in a preprocessing stage which allows the speed up of subsequent stages; ii) TPSYS explicitly stores all the levels where actions or propositions may appear (the temporal graph is complete and more informed), which makes the size of the planning graph of TPSYS larger than in TGP; iii) unlike TGP, the search in TPSYS finds a plan as an acyclic flow of actions throughout the temporal graph; and iv) TPSYS handles a more precise, non-conservative model of actions which implies fewer constraints on the execution of actions and a larger search space. On the other hand, our least-commitment approach for overcoming the limitations of the Graphplan backward search basically postpones the allocation in time of actions until they become non mutex and applicable. Thus, the algorithm generates a relaxed plan similar 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. The critical difference with LPG relies on several points. First, the planning graph is temporal and moves chronologically in time instead of planning steps. Although this planning graph involves a high level of complexity, it provides valuable information for estimating the times for actions and propositions. Second, the temporal graph is not only used to extract heuristic information but also to generate a relaxed plan (which may include obligatory actions), preventing TPSYS from starting the search from an empty plan. Third, this plan is repaired in a progressive, 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 uses a deterministic, systematic approach which, unfortunately, reduces its scalability in very large problems.

For simplicity, the main conclusions of this paper can be organised in terms of the three stages of our temporal planning system:

Calculus and analysis of static mutex

- A simple algorithm to perform reachability analysis prevents the planner from reasoning on actions and propositions that are irrelevant to the plan, thereby reducing the planning effort. This can also easily detect cases when the problem is unsolvable.
- The inclusion of a preprocessing stage to calculate the static mutex which always hold becomes very useful, especially in a non-conservative model of actions which imposes a more complex calculus of mutex (action/action and proposition/action).

Extension of the temporal planning graph

- A temporal planning graph is a generalisation of the classical Graphplan planning graph. Persistence of the propositions can be handled by a more precise calculus of mutex, without the need for `no-op` actions. In addition, the different durations of the actions break the initial symmetry of the planning graph as the levels are no longer equidistant, thus increasing the size of that graph.

- The propagation of the dynamic mutex when dealing with a non-conservative model of durative actions becomes more complex. Moreover, the combination of initial effects and final conditions of PDDL2.1 forces the modification of the termination criterion: the planner must guarantee the successful termination of all the actions involved in the plan.

Search of a temporal plan

- The original Graphplan *right-to-left* directionality of the search is broken when dealing with actions of PDDL2.1 (due to the combination of local conditions and effects), and now a *right-to-left* and *left-to-right* strategy is necessary to support all the goals.

- A process of temporal memoization is very useful for improving the performance of the search. However, memoization in temporal planning implies taking into consideration the actions which have already been planned in order to calculate the conflict set of propositions and actions. EBL/DDB techniques proposed by Kambhampati (1999, 2000) are also applicable in a temporal setting, helping reduce the complexity of the search.

- The Graphplan search space can be reduced by reducing the depth or the branching factor. We have evaluated several methods and heuristics to do this and have shown that the *safest* approaches are the methods for reducing the search depth.

- Least-commitment and heuristic local search techniques can be applied in a temporal (or even classical) Graphplan-based approach. The substitution of the backward search by a two-stage search allows us to overcome some of its main inefficiencies. The new search creates an initial relaxed plan that is subsequently repaired by means of a progressive heuristic process, which combines the information calculated in the planning graph with a greedy search process, eliminates redundancy in the search and increases the planner scalability. The search process is guided by two estimation functions which use local features to select the next action or plan to be considered. The advantage is that the planner can exploit a higher level of action concurrency, which leads to plans that are highly competitive with other state-of-the-art planners under a deterministic approach. The main disadvantage, however, is that the search is not complete preserving.

The algorithm for least-commitment and heuristic search still has some limitations. Our current work focuses on the refinement of the heuristic functions to improve the quality of the plans, the performance of the planner and to avoid some of the inconveniences of the greedy search. In addition to this, we are also working on the increment in the expressiveness of the model of actions to deal with numeric values which allow the planner to reason on resource utilisation and to use multicriteria optimisation functions.

References

Bacchus, F., & Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, *116(1-2)*, 123–191.

Blum, A., & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*, 281–300.

Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In Biundo, S., & Fox, M. (Eds.), *Proc. European Conference on Planning (ECP-99)*, pp. 360–372. Springer.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*, 5–33.

Cayrol, M., Reginer, P., & Vidal, V. (2001). Least commitment in Graphplan. *Artificial Intelligence*, *130*, 85–118.

Currie, K., & Tate, A. (1991). O-Plan: the open planning architecture. *Artificial Intelligence*, *52(1)*, 49–86.

Do, M., & Kambhampati, S. (2001). Sapa: a domain-independent heuristic metric temporal planner. In Cesta, A., & Borrajo, D. (Eds.), *Proc. European Conference on Planning (ECP-2001)*, pp. 109–120.

Edelkamp, S. (2002). Mixed propositional and numeric planning in the model checking integrated planning system. In Fox, M., & Coddington, A. (Eds.), *Proc. Workshop on Planning for Temporal Domains (AIPS-2002)*, pp. 47–55.

El-Kholy, A., & Richards, B. (1996). Temporal and resource reasoning in planning: the parcPLAN approach. In *Proc. 12th European Conference on AI (ECAI-96)*, pp. 614–618.

Fox, M., & Long, D. (1998). The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, *9*, 367–421.

Fox, M., & Long, D. (1999a). The detection and exploitation of symmetry in planning domains. In Kaufmann, M. (Ed.), *Proc. 15th International Joint Conference on AI (IJCAI-99)*, pp. 956–961, Stockholm, Sweeden.

Fox, M., & Long, D. (1999b). Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, *10*, 87–115.

Fox, M., & Long, D. (2001). PDDL2.1: an extension to PDDL for expressing temporal planning domains. Tech. rep., University of Durham, UK.

Garrido, A., Fox, M., & Long, D. (2002). A temporal planning system for durative actions of PDDL2.1. In Harmelen, F. V. (Ed.), *Proc. European Conference on AI (ECAI-2002)*, pp. 586–590, Amsterdam. IOS Press.

Garrido, A., & Onaindía, E. (2003). On the application of least-commitment and heuristic search in temporal planning. In *Proc. Int. Joint Conference on AI (IJCAI-2003)*, pp. 942–947, Acapulco, Mexico. Morgan Kaufmann.

Garrido, A., Onaindía, E., & Barber, F. (2001). Time-optimal planning in temporal problems. In Cesta, A., & Borrajo, D. (Eds.), *Proc. European Conference on Planning (ECP-2001)*, pp. 397–402.

Gerevini, A., & Serina, I. (2002a). LPG: a planner based on local search for planning graphs with action costs. In *Proc. 6th Int. Conference on AI Planning and Scheduling (AIPS-2002)*, pp. 281–290. AAAI Press.

Gerevini, A., & Serina, I. (2002b). Planning through stochastic local search and temporal action graphs. Tech. rep. R.T.2002-05-28, University of Brescia, Italy.

Ghallab, M., & Laruelle, H. (1994). Representation and control in IxTeT, a temporal planner. In *Proc. 2nd Int. Conference on AI Planning Systems*, pp. 61–67. Hammond.

Halsey, K. (2002). Literacy survey. A review of temporal planning. Tech. rep., University of Durham, UK.

Haslum, P., & Geffner, H. (2001). Heuristic planning with time and resources. In Cesta, A., & Borrajo, D. (Eds.), *Proc. European Conference on Planning (ECP-2001)*, pp. 121–132.

Hoffmann, J. (2000). 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.

Hoffmann, J. (2002). Extending FF to numerical state variables. In Harmelen, F. V. (Ed.), *Proc. European Conference on AI (ECAI-2002)*, pp. 571–575, Amsterdam. IOS Press.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Kambhampati, S. (1999). Improving Graphplan's search with EBL and DDB techniques. In Dean, T. (Ed.), *Proc. Int. Joint Conference on AI (IJCAI-99)*, pp. 982–987, Stockholm, Sweden. Morgan Kaufmann.

Kambhampati, S. (2000). Planning graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP techniques in Graphplan. *Journal of Artificial Intelligence Research*, *12*, 1–34.

Kambhampati, S., Lambrecht, E., & Parker, E. (1997). Understanding and extending Graphplan. In *Proc. European Conference on Planning (ECP-97)*.

Kambhampati, S., & Sanchez Nigenda, R. (2000). Distance based goal ordering heuristics for Graphplan. In *Proc. Int. Conference on Artificial Intelligence Planning and Scheduling*, pp. 315–332, Menlo Park, CA. AAAI Press.

Kvarnstrom, J., & Doherty, P. (2000). TALPlanner: a temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, *30(1-4)*, 119–169.

Long, D., & Fox, M. (2001). Encoding temporal planning domains and validating temporal plans. In *Proc. 20th UK Planning and Scheduling SIG Workshop*.

Long, D., & Fox, M. (2002). Progress in AI planning research and applications. *UPGRADE The European Online Magazine for the IT Professional*, *3(5)*, 10–24.

Long, D., & Fox, M. (2003). Exploiting a Graphplan framework in temporal planning. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2003)*.

Lopez, A., & Bacchus, F. (2003). Generalizing graphplan by formulating planning as a CSP. In *Proc. Int. Joint Conference on AI (IJCAI-2003)*, pp. 954–960, Acapulco, Mexico. Morgan Kaufmann.

McDermott, D. (1998). *PDDL - The Planning Domain Definition Language*. AIPS-98 Planning Competition Committee.

Muscettola, N. (1994). HSTS: Integrating planning and scheduling. In Zweben, M., & Fox, M. (Eds.), *Intelligent Scheduling*, pp. 169–212. Morgan Kaufmann, San Mateo, CA.

Nau, D., Cao, Y., Lotem, A., & Muñoz Avila, H. (1999). SHOP: Simple hierarchical ordered planner. In *Proc. Int. Joint Conference on AI (IJCAI-99)*, pp. 968–973, Stockholm, Sweden.

Nau, D., Muñoz Avila, H., Cao, Y., Lotem, A., & Mitchell, S. (2001). Total-order planning with partially ordered subtasks. In *Proc. Int. Joint Conference on AI (IJCAI-2001)*, pp. 425–430, Seattle, WA. Morgan Kaufmann Publishers, Inc.

Nebel, B., Dimopoulos, Y., & Köehler, J. (1997). Ignoring irrelevant facts and operators in plan generation. In *Proc. European Conference on Planning (ECP-97)*, pp. 338–350, Toulousse, France.

Nguyen, X., Kambhampati, S., & Nigenda, R. (2002). Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, *135*, 73–123.

Penberthy, J., & Weld, D. (1994). Temporal planning with continuous change. *Proc. 12th Nat. Conference on AI*.

Sanchez Nigenda, R., & Kambhampati, S. (2003). Parallelizing state space plans online. In *Proc. Int. Joint Conference on AI (IJCAI-2003)*, pp. 1522–1523, Acapulco, Mexico. Morgan Kaufmann.

Smith, D., & Weld, D. (1999). Temporal planning with mutual exclusion reasoning. In *Proc. 16th Int. Joint Conference on AI (IJCAI-99)*, pp. 326–337, Stockholm, Sweden.

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

Weld, D. (1999). Recent advances in AI planning. *AI Magazine*, *20(2)*, 93–123.

Younes, H., & Simmons, R. (2002). On the role of ground actions in refinement planning. In *Proc. 6th Int. Conference on AI Planning and Scheduling (AIPS-2002)*, pp. 54–61. AAAI Press.

Zimmerman, T., & Kambhampati, S. (1999). Exploiting symmetry in the planning-graph via explanation-guided search. In *Proc. Nat. Conference on Artificial Intelligence*.

Zimmerman, T., & Kambhampati, S. (2003). Using available memory to transform graphplan's search. In *Proc. Int. Joint Conference on AI (IJCAI-2003)*, pp. 1526–1527, Acapulco, Mexico. Morgan Kaufmann.