A Temporal Planning System for Durative Actions of PDDL2.1

Antonio Garrido[1] and **Maria Fox** [2] and **Derek Long** [2]

Abstract. Many planning domains have temporal features that can be expressed using durations associated with actions. Unfortunately, the conservative model of actions used in many existing temporal planners is not adequate for some domains which require more expressive models. Level 3 of PDDL2.1 introduces a model of durative actions that includes local conditions and effects to be satisfied at different times during the execution of the actions, thereby giving the planner greater freedom to exploit concurrent actions. This paper presents a temporal planning system which combines the principles of Graphplan and TGP in order to plan with such actions. Although the system is consequently more complex than TGP, the experimental results demonstrate it remains feasible as a way to deal with durative actions.

1 INTRODUCTION

Temporal planners appearing in the recent literature, such as parcPLAN, TGP or TP4 [3, 8, 5] have demonstrated some success in dealing with actions with duration. These planners have adopted a conservative model of actions that is a modest extension of that used by non-temporal planners, which 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, called PDDL2.1 [4], provides a level 3 with a model of durative actions which subsumes the conservative model of actions. Level 3 allows actions to overlap even when their preconditions or effects refer to the same propositions. This paper presents a Temporal Planning SYStem (from now on TPSYS) to manage the model of durative actions proposed in level 3 of PDDL2.1. TPSYS is based on a three-stage process, which combines the ideas of Graphplan [1] and TGP [8] to deal with such durative actions. Hence, the main contributions of this paper are:

- An analysis of how durative actions can be managed in a Graphplan-based approach.
- An explanation of how a compact temporal graph can be generated, without *no-op* actions and delete-edges.
- An extension of the mutual exclusion reasoning to manage PDDL2.1 durative actions, based on the work of TGP.
- A description of the plan extraction stage and the way it obtains the plan of optimal duration (in terms of the makespan).
- Some experimental results showing the importance of the mutex reasoning in richer models of actions, as indicated in [8].

[1] Universidad Politecnica de Valencia, Spain, email: agarridot@dsic.upv.es
[2] University of Durham, UK, email: {maria.fox, d.p.long}@dur.ac.uk

2 MOTIVATION

Unlike PDDL, PDDL2.1 allows the modelling of temporal planning domains to achieve a fuller exploitation of concurrency. This entails 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. Moreover, there might be initial effects of 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 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)*. However, as presented in [4], this may exclude many valid plans. In PDDL2.1 this can be avoided by asserting ¬*at(plane,origin)* as an initial effect.

In addition, if we want to know that an aircraft 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 durative action equivalent to this *fly* action would not represent the fact of *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 actions which require this proposition, such as a possible action *refuel-during-flight*.

3 ACTION MODEL AND TERMINOLOGY

Durative actions 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 also effects to be asserted immediately after the actions start.

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

- *Conditions. The three types of local conditions of a durative action are: i) $SCond_a$, the set of conditions to be guaranteed at the* *start of the action; ii) Inv_a, the set of invariant conditions to be guaranteed over the execution of the action; and iii) $ECond_a$, the set of conditions to be guaranteed at the end of the action.*

- *Duration. The duration of the action is a positive value represented by $D_a \in \mathbb{R}^+$.*
- *Effects. The two types of effects of a durative action are: i) $SEff_a = \{SAdd_a \cup SDel_a\}$, with the positive and negative effects respectively to be asserted at the start of the action; and ii) $EEff_a = \{EAdd_a \cup EDel_a\}$, with the positive and negative effects respectively to be asserted at the end of the action.*

Durative actions entail an important difficulty: there exist some effects ($SEff_a$) which can be obtained before the action a ends. Hence, it might be possible that an initiated action could not end because its end conditions ($ECond_a$) are not satisfied in the future. In that case, all the start effects (and the actions which are dependent on them) should be invalidated. We call these kind of actions *conditional actions* because they are provisional until their end conditions are guaranteed, and we define them as:

Definition 2 (Conditional action) *An action a is a conditional action if $(SEff_a \neq \emptyset) \wedge (ECond_a \neq \emptyset)$ holds. In this way, the set of propositions $SEff_a$ of a conditional action a only becomes valid when all propositions in $ECond_a$ are satisfied.*

Conditional actions occur in domains in which durative actions are required precisely for some effect achieved through the duration of execution of an action (it is bounded by that duration). Such initial effects cannot be exploited as end effects because they do not persist beyond the end of the action. For example, in a logistics domain the plane is *flying* only during the action *fly*, so the initial effect *flying(plane)* cannot be exploited beyond the end of the *fly* action. Furthermore, the successful termination of a durative action 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 [8] and conditions/effects which come to play at a specific time point and must persist only over finitely bounded intervals [2], but PDDL2.1 does not support this yet.

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

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

Figure 1 shows the definition of the durative actions *board* and *fly* in a simple logistics domain. According to Definition 1, the actions have *at start* and *over all* conditions with the conditions to be satisfied just at the beginning of the action and during all its execution, 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 sight 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 the next section.

```
(:durative-action board
 :parameters (?p - person ?a - aircraft
              ?c - city)
 :duration (= ?duration (boarding-time ?c))
 :condition (and (at start (at ?p ?c))
                 (at start (free ?a))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (at ?p ?c)))
              (at start (not (free ?a)))
              (at end (in ?p ?a))))
(:durative-action fly
  :parameters (?a - aircraft ?c1 ?c2 - city)
  :duration (= ?duration (fly-time ?c1 ?c2))
  :condition (and (at start (at ?a ?c1)))
  :effect (and (at start (not (at ?a ?c1)))
               (at end (at ?a ?c2))))
```

Figure 1. Definition of *board* and *fly* in level 3 of PDDL2.1.

4 THE TEMPORAL PLANNING SYSTEM

In TPSYS, a temporal planning problem is specified as the 4-tuple $\{\mathcal{I}_s, \mathcal{A}, \mathcal{F}_s, \mathcal{D}_{max}\}$, where $\mathcal{I}_s$ and $\mathcal{F}_s$ represent the initial and final situation, respectively. $\mathcal{A}$ represents the set of durative actions in the planning domain. Time is modelled by $\mathbb{R}^+$ and their chronological order. $\mathcal{D}_{max}$ stands for the maximum duration allowed by the user. Although this bound is not defined in PDDL2.1 and it could be difficult to be decided, it allows the user a good way to constrain the goals deadline and the makespan of the plan as in [2].

TPSYS is executed in three consecutive stages. After the first stage, the second and the third stages are executed in an interleaved way until a plan is found or the duration exceeds $\mathcal{D}_{max}$.

4.1 First stage: static mutex reasoning

Graphplan-based planners identify binary mutual exclusion relations between actions and between propositions. As with TGP, TPSYS needs to calculate action-action, proposition-action and proposition-proposition mutex relationships. Since proposition-proposition mutex appears as a consequence of action-action mutex, this first stage only calculates the action-action and proposition-action *static* mutex relationships. These mutex relationships are static because they only depend on the definition of the actions and they always hold. Therefore, there is no reason to postpone their calculation to the next stages, thereby speeding up the second and third stages. The process of calculating the mutex relationships is complicated by the semantics of PDDL2.1, which embodies a more permissive mutual exclusion relation than the languages of other temporal planners such as TGP. In particular, the strong mutex of the conservative model of actions must be modified to allow durative actions to be applied in parallel even in cases in which they refer to the same propositions.

There exist four action-action mutex situations. Situation 1 (*at start*) represents the mutex in which actions cannot start at the same time because start effects are contradictory or start effects and start conditions are conflicting. Situation 2 (*at end*) represents the mutex in which actions cannot end at the same time because end effects are contradictory or end effects and end conditions are conflicting. Situation 3 (*at end-start*) represents the mutex in which two actions cannot end and start at the same time, i.e. the actions cannot meet, because the end effects of one action conflict with the start conditions or effects of the other action. This mutex (which does not appear in TGP) might seem a stronger requirement than is really required, but it takes account of 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. However, TPSYS takes the correctness-preserving assumption of including an epsilon ($\epsilon > 0$) between the action which ends and the action which starts to avoid this mutex and to make easier the implementation of the algorithm. Finally, situation 4 (*during*) represents the mutex in which one action cannot start or end during the execution of the other because the start or end effects of the former are conflicting with the invariant conditions of the latter.

The proposition-action mutex relationships are also calculated in the first stage. As demonstrated in [8], when actions have different duration in a Graphplan-based approach, mutex between propositions and actions help in the deduction of more inconsistencies between propositions when actions are executed in parallel.

Definition 4 (Static pa-mutex) *A proposition p is statically mutex with action a iff $p \in \{SDel_a \cup EDel_a\}$.*

4.2 Second stage: extension of the temporal graph

The second stage performs the extension of the temporal graph. The temporal graph consists of a directed, layered graph which alternates temporal levels of propositions and actions, represented by $P_{[t]}$ and $A_{[t]}$ respectively. The levels are chronologically ordered by their instant of time, by means of a label t which represents the instant of time in which propositions are present and actions can start or end.

4.2.1 Requirements in the extension of the temporal graph

The extension of the temporal graph contains some subtle details due to the local conditions and effects of durative actions. Each temporal level t needs to be divided into two parts, *end*-part and *start*-part, in which the following action-action ($AA_{[t]}$ mutex), proposition-action ($PA_{[t]}$ mutex) and proposition-proposition ($PP_{[t]}$) mutex relationships are calculated. We use the notation $AA_{[t]}$, $PA_{[t]}$ and $PP_{[t]}$ to represent the mutex relationships that hold at time t. These mutex relationships are temporary and can disappear in time, in contrast with the notation AA and PA that represent the static mutex relationships which always hold. The actions which end (start) at action level $A_{[t]}$ are stored in $A_{[t]end}$ ($A_{[t]start}$). Analogously, the propositions achieved as end (start) effects are stored in $P_{[t]end}$ ($P_{[t]start}$).

On one hand, the mutex relationships to be calculated in the *end*-part are $AA_{[t]end-end}$, $PA_{[t]end-end}$ and $PP_{[t]end-end}$. On the other hand, the mutex relationships to be calculated in the *start*-part are $AA_{[t]start-start}$, $AA_{[t]end-start}$, $PA_{[t]start-start}$, $PP_{[t]end-start}$ and $PP_{[t]start-start}$. The reason for breaking down these mutex relationships into *end*-part and *start*-part lies in keeping their calculus simpler, as can be seen in the following definitions:

Definition 5 ($AA_{[t]end-end}$) *Two actions a, b are end-end mutex at time t if one of the following holds: i) a, b are $AA_{end-end}$, ii) $ECond_a, ECond_b$ are $PP_{[t]end-end}$, or iii) a, b are $AA_{[t-\min(D_a,D_b)]start-start}$.*

Definition 6 ($PA_{[t]end-end}$) *Let p be a proposition and a be an action. For each action b_i which achieves p at t, let $\Upsilon_{i[t]}$ be the condition under b_i is mutex with the persistence of p at time t, i.e. $\Upsilon_{i[t]} = [(p, b_i$ are $PA) \vee (p, ECond_{b_i}$ are $PP_{[t]end-end})]$. Proposition p and action a are end-end mutex at time t if the following condition holds: $\bigwedge_i [\Upsilon_{i[t]} \wedge (a, b_i$ are $AA_{[t]end-end})]$.*

Definition 7 ($PP_{[t]end-end}$) *Let p, q be two propositions and $\{a_i\}, \{b_j\}$ be the sets of actions which achieve p and q at time t, respectively. Propositions p, q are end-end mutex at time t if both of the following conditions hold: i) $\forall b_j : p, b_j$ are $PA_{[t]end-end}$, and ii) $\forall a_i : q, a_i$ are $PA_{[t]end-end}$.*

Definition 8 ($AA_{[t]start-start}$) *Two actions a, b are start-start mutex at time t if one of the following holds: i) a, b are $AA_{start-start}$, or ii) $SCond_a, SCond_b$ are $PP_{[t]start-start}$.*

Definition 9 ($AA_{[t]end-start}$) *Two actions a (ending at t) and b (starting at t) are end-start mutex at time t if one of the following holds: i) a, b are $AA_{end-start}$, or ii) $ECond_a, SCond_b$ are $PP_{[t]end-end}$.*

Definition 10 ($PA_{[t]start-start}$) *Let p be a proposition and a be an action. For each action b_i which achieves p at t, let $\Psi_{i[t]}$ be the condition under b_i is mutex with the persistence of p at time t, i.e. $\Psi_{i[t]} = [(p, b_i$ are $PA) \vee (p, SCond_{b_i}$ are $PP_{[t]start-start})]$. Proposition p and action a are start-start mutex at time t if the following condition holds: $\bigwedge_i [\Psi_{i[t]} \wedge (a, b_i$ are $AA_{[t]start-start})]$.*

Definition 11 ($PP_{[t]end-start}$) *Let p be a proposition first achieved at time t by the set of actions $\{a_i\}$ which end at t. Analogously, let q be another proposition first achieved at t by the set of actions $\{b_j\}$ which start at t. Propositions p, q are end-start mutex at time t if the following condition holds: $\forall a_i, b_j : a_i, b_j$ are $AA_{[t]end-start}$.*

Definition 12 ($PP_{[t]start-start}$) *Let p, q be two propositions and $\{a_i\}, \{b_j\}$ be the sets of actions which achieve p and q at time t, respectively. Propositions p, q are start-start mutex at time t if both of the following conditions hold: i) $\forall b_j : p, b_j$ are $PA_{[t]start-start}$, and ii) $\forall a_i : q, a_i$ are $PA_{[t]start-start}$.*

Intuitively, $AA_{[t]}$ mutex relationships represent the impossibility of two actions ending, starting or abutting together at the same time t. $PA_{[t]}$ mutex represents the impossibility of having a proposition and an action starting or ending at time t. $PP_{[t]}$ mutex represents the impossibility of having two propositions together at time t. The calculus of the mutex relationships provides very useful information which can be used to improve search efficiency by pruning invalid combinations of actions, propositions and propositions/actions.

An important point to take into account when dealing with durative actions in a Graphplan-based approach is the condition to finish the extension of the temporal graph. In TGP, this condition holds once all the propositions of the final situation are non pairwise mutex. However, conditional actions assert *at start* effects which might satisfy goals in the final situation *before* these actions end. 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 situation is still conditional. Therefore, it is necessary to propagate some additional *heuristic* information about the validity of the propositions achieved in the temporal graph. In this case, the same disjunctive reasoning on propositions of Graphplan can be applied to the instants of time at which the propositions stop being conditional.

Definition 13 (End time of a conditional proposition) *Let p be a conditional proposition and $\{a_i\}$ the set of conditional actions which achieve p. In the proposition level $P_{[t]}$ (at time t), the end time in which p stops being conditional, $\max_{etc}$ (the maximum end time conditional) is calculated as $\min(\alpha_i)$, where α_i is defined as:*

- $\max(\max_{etc}(SCond_{a_i}) + D_{a_i}, \max_{etc}(ECond_{a_i}), t)$, *if p is achieved in an end-part of the graph.*
- $\max(\max_{etc}(SCond_{a_i}) + D_{a_i}, t)$, *if p is achieved in a start-part of the graph.*

4.2.2 Algorithm for the extension of the temporal graph

Figure 2 shows the algorithm for the temporal graph extension with the modifications presented above. Starting at time $t = 0$, the algorithm generates new proposition and action levels (*end*-part and *start*-part), calculating all the mutex relationships. $No - op$ actions and delete-edges are not stored in the graph. The extension continues until the propositions in the final situation are achieved and they are not conditional. If the maximum time allowed by the user $\mathcal{D}_{max}$ is exhausted, the algorithm returns '*Failure*'.

```
$t = 0$
while $(t \leq \mathcal{D}_{\max}) \wedge (\mathcal{F}_s$ is not satisfied in $P_{[t]}) \wedge$
        $(\mathcal{F}_s$ has not conditional propositions) do
    forall $< a_i, s_i, t >$ which can end at $A_{[t]end}$ do
        $A_{[t]end} = A_{[t]end} \cup a_i$
        $P_{[t]end} = P_{[t]end} \cup EAdd_{a_i}$
        Generate start-part mutex
    forall $< b_j, t, e_j >$ which can start at $A_{[t]start}$ do
        $A_{[t]start} = A_{[t]start} \cup b_j$
        $P_{[t]start} = P_{[t]start} \cup SAdd_{b_j}$
        Generate end-part mutex
    $t$ = next level in the Temporal Graph
```

Figure 2. Algorithm for the temporal graph extension.

Lemma 1 (The extension of the temporal graph is complete) *If the temporal graph extension ends at time t, the algorithm generates all the necessary temporal levels (at which actions can end or start) between time 0 and t.*

Proof The proof is direct by definition of the algorithm. The algorithm generates all the actions $\{b_j\}$ whose $SCond_{b_j}$ hold in each temporal level. Since each action level contains all the actions present in the previous action levels —analogously for the proposition levels—, once one action b_j appears this action will appear in all the following levels in which b_j could end and start. □

4.3 Third stage: extraction of a plan

The third stage performs the extraction of an optimal plan, as an acyclic flow of actions, through the temporal graph. Now, durative actions allow different ways to achieve propositions, not only by their *at end* effects but also by their *at start* effects. Consequently, planning a durative action commits a plan to satisfying the start, invariant and end conditions. This breaks the traditional right to left *directionality* of Graphplan or TGP as shown in the following example.

Let t be a time at which a proposition p must be satisfied during the extraction of a plan. Let us suppose that action a achieves p at t as a start effect ($p \in SAdd_a$). If a has end conditions, they will have to be satisfied at time $t' = t + D_a$, forcing the algorithm to revisit a previously visited point of time $t' > t$.

Figure 3 shows the algorithm for the plan extraction. It uses two structures, one queue $GoalsToSatisfy$ formed by pairs $< p, t >$ with the goal proposition p to be satisfied at time t, and one list $Plan$ formed by $< a_i, s_i, e_i >$ 3-tuples with the planned action a_i executed in $[s_i..e_i]$. $GoalsToSatisfy$ is initialized with the propositions of the final situation to be satisfied at the instant of time at which the temporal graph extension has finished. $Plan$ is initially empty. While $GoalsToSatisfy$ is not empty, the algorithm dequeues a pair $< p, t >$ to be satisfied. Note that now, p could be already satisfied at time t because actions are planned in different points of time and not always in a right to left order. If p is not already satisfied at time t in $Plan$, actions that satisfy p at time t are selected in a backtracking point. All actions $\{a_i\}$ which are compatible with $Plan$ must be considered for completeness. If action a_i is not mutex with $Plan$, then a_i is planned updating the structures $Plan$ and $GoalsToSatisfy$ with a_i and its start, invariant and end conditions, respectively. Finally, if $GoalsToSatisfy$ gets empty, the optimal plan is found. Otherwise, the algorithm repeats the second and third stages, searching for a plan from a later temporal level, in a way similar to the interleaved search and construction process of Graphplan.

```
$GoalsToSatisfy = \mathcal{F}_s$ at the end time of second stage
$Plan = \emptyset$
while $(GoalsToSatisfy \neq \emptyset)$ do
    Dequeue $< p, t >$ from $GoalsToSatisfy$
    if $< p, t >$ is not already satisfied in $Plan$
        Select $< a_i, s_i, e_i >$ which satisfies $p$ at $t$ and
                    compatible with $Plan$
        $Plan = Plan \cup < a_i, s_i, e_i >$
        $GoalsToSatisfy = GoalsToSatisfy \cup SCond_{a_i}$
                    $\cup Inv_{a_i} \cup ECond_{a_i}$
```

Figure 3. Algorithm for the plan extraction.

Lemma 2 (The extraction of a plan is a complete process)

Proof The proof is trivial due to the fact that the algorithm considers all the possible actions (*backtracking point*) which satisfy each proposition p from $GoalsToSatisfy$. □

Theorem 1 (Optimality of the algorithm) *The first plan the algorithm extracts is a plan of optimal duration.*

Proof By contradiction, let $\mathcal{P}_t$ be the first plan (of duration t) the algorithm extracts. We assume this plan is not optimal, so we deduce that there exists a plan $\mathcal{P}'_{t'}$ (of duration $t' < t$) which has not been found and is optimal. This implies one of the following cases: i) the temporal level t' has not been generated in the second stage, or ii) the temporal level t' has been generated but the extraction stage has not considered the plan $\mathcal{P}'_{t'}$. The first case is false by Lemma 1 which claims the completeness of the temporal graph extension, and the second case is also false by Lemma 2 which claims the completeness of the plan extraction stage. This contradicts the initial choice of the existence of $\mathcal{P}'_{t'}$, and therefore $\mathcal{P}_t$ is a plan of optimal duration. □

5 EXPERIMENTAL RESULTS

We have adapted some of the traditional domains of PDDL, such as *logistics*, *blocksworld*, *zeno-travel*, etc., to the model of durative actions of PDDL2.1 to perform some experiments. Direct comparison between TPSYS and some planners such as Sapa [2] or TP4 [5] is difficult because they handle resources and even non-admissible heuristics which cannot guarantee the optimal solution. However, we want to do direct comparison in the immediate future. Therefore, we compare TPSYS with TGP to demonstrate that the algorithm presented here remains feasible when dealing with traditional temporal planning problems. The tests were censored after 60 seconds. The results of the tests obtained in a 64 Mb. memory Celeron 400 MHz. can be seen in Table 1.

The results show that TPSYS behaves well enough in all the problems. Although the model of durative actions entails more mutex relations to calculate and a larger space of search, the performance of TPSYS follows the same order of magnitude of TGP. The most important differences appear in the problems *att-log3* and *big-bull2*, in which TGP is clearly better than TPSYS. The reason relies on the higher number of instantiations in time (higher *branch factor*) that TPSYS has to carry out.

Problem	TPSYS	TGP
att-log0	0.42	0.02
att-log1	0.44	0.05
att-log2	0.47	0.06
att-log3	14.10	2.65
bulldozer-prob	0.88	0.55
big-bull1	0.58	0.80
big-bull2	14.31	2.15
gripper2	0.03	0.03
gripper4	0.17	0.13
gripper6	6.88	4.53
monkey1-test	0.20	0.17
monkey2-test	0.63	0.75
tower2	0.02	0.03
tower4	0.28	0.45
tower6	2.52	3.60
zeno-travel1	0.01	0.01
zeno-travel2	0.02	0.01
zeno-travel3	0.02	0.01

Table 1. Comparison of TPSYS and TGP (results are in seconds).

6 RELATED WORK AND DISCUSSION

The last decade has seen many attempts at dealing with temporal planning. ZENO [7] uses a causal link framework with continuous change but it is relatively slow. The parcPLAN approach [3] handles a rich set of temporal constraints, instantiating time points in a similar way to TPSYS. TGP [8] introduces a complex mutual exclusion reasoning which is very valuable in temporal environments. The critical difference between TGP and TPSYS is based on several points. First, TPSYS calculates the static mutex relationships in a preprocessing stage which allows the speed up of subsequent stages. Second, TGP uses a more compact temporal graph in which actions and propositions are only annotated with the first level at which they appear. This vastly reduces the space costs but it increases the complexity of the search process. In contrast, TPSYS uses a more informed temporal graph which reduces the overhead during search. Third, the mutex reasoning is managed in TGP by means of a collection of temporal formulae, whereas TPSYS calculates the mutex relationships level by level in a way that is more similar to Graphplan. Finally, TPSYS uses a richer model of actions which implies: i) fewer constraints on the execution of the actions, ii) new requirements in the planning algorithm, and iii) a significantly larger search space. Sapa [2] is a metric temporal planner which uses a model of actions similar to PDDL2.1, but it does not perform mutex propagation as our system. Sapa scales up quite well, but it uses non-admissible heuristics which cannot guarantee the optimal plan. TP4 [5] uses admissible heuristic search to handle actions with time and resources, but it assumes a conservative model of actions.

7 CONCLUSIONS AND FUTURE WORK

As far as we know, the optimal temporal planning system described in this paper represents one of the first attempts to manage level 3 durative actions of PDDL2.1 in a Graphplan-based approach. Briefly, the main contributions of the paper have been the description of:

- The new components of level 3 durative actions based on [4] and the mutual exclusion relationships they entail.
- The new requirements of the temporal graph extension and the way in which the mutex relationships can be calculated.
- The modifications needed during the plan extraction. We have presented how the plan is found through the temporal graph breaking the traditional right to left directionality of Graphplan and TGP.

The algorithm still has some limitations. According to our experiments, the algorithm does not scale up to large problems with many actions and propositions due to the calculus of the mutual exclusion relationships. Moreover, the performance of the second and third stages degrades when the duration of the actions is wildly different, in particular when the greatest common divisor of the durations is 1. For this reason, the areas of future work are focused on the inclusion of memoization [1] and some of the CSP techniques presented in [6], which have already been tested in TGP and have proved very promising as a way of improving the behaviour of the plan extraction stage. We also want to extend TPSYS to handle additional features of level 3 of PDDL2.1, such as numeric conditions and effects and inequality relations on conditions.

Additionally, our immediate research consists of the analysis and comparison of other temporal planning approaches such as those presented in the recent international planning competition, associated with AIPS 2002, using the new benchmark temporal planning domains.

8 Acknowledgements

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

REFERENCES

[1] A.L. Blum and M.L. Furst, ‘Fast planning through planning graph analysis’, *Artificial Intelligence*, **90**, 281–300, (1997).

[2] M.B. Do and S. Kambhampati, ‘Sapa: a domain-independent heuristic metric temporal planner’, in *Proc. ECP-01*, (2001).

[3] A. El-Kholy and B. Richards, ‘Temporal and resource reasoning in planning: the parcPLAN approach’, in *Proc. ECAI-96*, pp. 614–618, (1996).

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

[5] P. Haslum and H. Geffner, ‘Heuristic planning with time and resources’, in *Proc. ECP-01*, (2001).

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

[7] J. Penberthy and D. Weld, ‘Temporal planning with continuous change’, *Proc. 12th Nat. Conf. on AI*, (1994).

[8] D.E Smith and D.S. Weld, ‘Temporal planning with mutual exclusion reasoning’, in *Proc. IJCAI-99*, pp. 326–337, (1999).