A Temporal Planning System for Level 3 Durative Actions of PDDL2.1

Antonio Garrido
Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia
Camino de Vera s/n, Valencia (Spain)
{**agarridot@dsic.upv.es**}

Abstract

Many planning domains have temporal features that can be expressed as durations associated with actions. Unfortunately, the conservative model of actions of most temporal planners is not appropriate for some domains which require richer models. Level 3 of PDDL2.1 introduces a model of durative actions which includes local conditions and effects to be satisfied at different times during the execution of the actions, thereby giving the planner freedom to plan concurrent actions. This paper presents a temporal planning system (TPSYS), which combines the ideas of Graphplan and TGP, to plan with such durative actions. The approach necessitates the modification of some aspects of the basic planning algorithm: the mutex reasoning, the generation of the temporal graph and the search for an optimal plan. Although the algorithm becomes more complex, the experimental results demonstrate it remains feasible as a way to deal with durative actions.

Introduction

Typically, classical planning systems simplify real problems by imposing unreal constraints on the problems. Particularly, planners rely on a model of actions in which all actions have the same duration. Although this assumption may be adequate for some planning problems, it becomes inadequate when dealing with temporal planning problems. For instance, this assumption is not true in real temporal environments, where different actions take different times of execution and concurrent actions are required to minimise the duration of the plan. Consequently, in temporal environments the optimisation criterion must be changed because the interest lies in obtaining a plan of minimal duration rather than a plan of minimal number of actions.

Most temporal planners appeared in the recent literature, such as parcPLAN, TGP or TP4 (El-Kholy & Richards 1996; Smith & Weld 1999; Haslum & Geffner 2001) have yielded some success when dealing with temporality on actions. Nevertheless, these temporal planners have adopted the same conservative model of actions of non-temporal planners. This means that two actions cannot overlap in *any way* if an effect or precondition of one is the negation of an effect or precondition of the other. Although this makes it possible to produce reasonable plans in most benchmark planning domains, there exist some domains which require a richer model of actions, and in which better quality plans can be found if a richer model of actions is used.

PDDL2.1 (Fox & Long 2001) is the new version of the standard language (PDDL) for the encoding of the planning domains which has been proposed for the the AIPS-2002 Planning Competition. PDDL2.1 provides five levels to define planning problems. Concretely, the level 3 introduces a new model of actions, called durative actions, which makes it possible to allow actions to overlap even when their preconditions or effects refer to the same propositions. This is possible because traditional preconditions and effects are now annotated with time points.

This paper presents a Temporal Planning SYStem (from now on TPSYS) in order 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 (Blum & Furst 1997) and TGP (Smith & Weld 1999). 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.
- The 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 duration of the actions) as an acyclic flow of actions through the temporal graph.
- Some experimental results showing the importance of the mutual exclusion reasoning in richer models of actions, as indicated in (Smith & Weld 1999).

This paper is organized as follows. In the second section, we briefly review the motivations for introducing the model of durative actions of level 3 of PDDL2.1. The third section introduces the action model, the components of a durative action and the terminology used through the paper. The TPSYS algorithm and its three stages are described in the fourth section. This section provides the modifications

the planning algorithm necessitates to deal with durative actions. Some experimental results are shown in the fifth section, demonstrating the feasibility of the system proposed. The sixth section discusses two approximations for dealing with durative actions in traditional planners. Finally, the conclusions are presented in the seventh section.

Motivation

PDDL does not allow the definition of actions with duration, which imposes an important limitation in real temporal problems. In developing PDDL2.1 to allow the modelling of temporal planning domains it was considered critical to allow a fuller exploitation of concurrency than can be captured using the strong mutex relation of the conservative model of actions, as the used in TGP (Smith & Weld 1999). This entails a more precise modelling of the state transitions undergone by different propositions within the durative interval of the action. In particular, the preconditions of the starting point of the action do not necessarily need to be maintained throughout the interval. There may be *pre*conditions of the final effect of the action that can be achieved concurrently rather than maintained throughout the interval. Hence, it becomes necessary to distinguish invariant from non-invariant conditions because there might be invariant conditions that cannot be affected during the interval of execution. 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.

We motivate the modelling of the state transitions with the following example of the classical logistics domain in the conservative model of actions. 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 (Fox & Long 2001), this may exclude many valid plans. In PDDL2.1 this can be easily avoided by asserting ¬*at(plane,origin)* as an initial effect.

In addition, if we want to know the fact of being *flying* during the action *fly*, it would be enough by asserting the proposition *(flying-plane)* as an initial effect of the starting point and ¬*(flying-plane)* as a final effect of the end point. But, in a conservative model of actions, the equivalent action for this *fly* durative action would not represent the fact of being *flying* due to the impossibility of including 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 the possible action *refuel-during-flight*.

Although 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 in (Fox & Long 2001) because it is possible, at level 3 of PDDL2.1, 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 plan. Epsilon has to be chosen appropriately for a given domain and problem, because it represents a discretization of the time-line into indivisible units, the end points of which mark the points at which actions can be initiated or terminated.

Action Model and Terminology

Unlike traditional actions of PDDL, durative actions present more conditions to be guaranteed for the success of the action. Moreover, durative actions do not only have effects that hold at the end of the actions but also effects to be asserted immediately after the actions start.

Definition 1 ***Components of a durative action*** *(see Figure 1). 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.*

Although level 3 allows the modelling of numeric conditions and effects as well as logical transitions, this version of TPSYS does not manage them yet.

Durative actions entail an important difficulty: there exist some effects ($SEff_a$) which can be obtained before the action 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

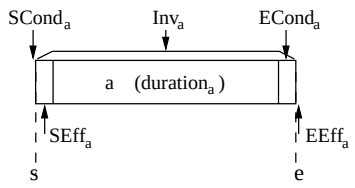

Figure 1: Components of a durative action a.

(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***. *One action a with $D_a > 0$ is a conditional action if $(SEff_a \neq \emptyset) \wedge (ECond_a \neq \emptyset)$ holds. 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 are motivated by observing that there are 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. Further, when plans are validated, 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 goal specifications might allow one to consider goals that must persist only over finitely bounded intervals (Do & Kambhampati 2001), but PDDL2.1 does not yet support this.

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

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 (stopping being 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 the correspondence rule below 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*** $\mathcal{R}_{a_i \mapsto a_d}$. *The correspondence rule maps an instantaneous action a_i, with Pre_{a_i}, $Effs_{a_i} = Add_{a_i} \cup Del_{a_i}$ into a durative action a_d in the following way:*

```
(: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 (flight-time ?c1 ?c2))
  :condition (and (at start (at ?a ?c1)))
  :effect (and (at start (not (at ?a ?c1)))
               (at end (at ?a ?c2))))

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

Figure 2: Definition of a simple domain in level 3 of PDDL2.1.

$$SCond_{a_d} = ECond_{a_d} = Inv_{a_d} = Pre_{a_i}$$
$$SAdd_{a_d} = EAdd_{a_d} = Add_{a_i}$$
$$SDel_{a_d} = EDel_{a_d} = Del_{a_i}$$
$$duration_{a_d} = 0$$

Figure 2 shows the definition of the simple logistics domain *zeno-travel* for durative actions of level 3 of PDDL2.1. The three actions are *board*, *fly* and *debark*, which have duration, conditions and effects. 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 have the effects to be asserted at the beginning and the end of the execution of the action.

At first blush the extension of a Graphplan-based planner to deal with durative actions of level 3 would seem quite easy. However, durative actions imply important changes in the way the temporal graph is generated and in the way the search for a plan is performed. These modifications are presented in the next section.

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

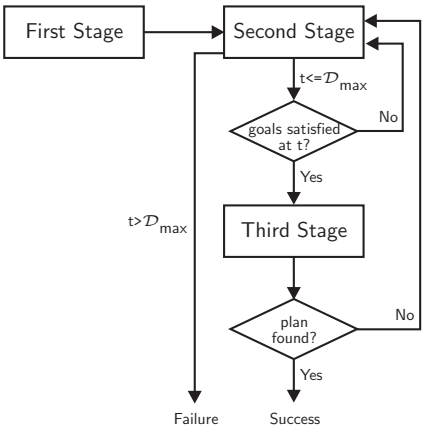

Figure 3: The three stages of TPSYS.

be decided, it allows the user a good way to constrain the goals deadline and the makespan of the plan as in (Do & Kambhampati 2001).

TPSYS is executed in three consecutive stages (see Figure 3). After the first stage, the second and the third stage are executed in an interleaved way until a plan is found or the duration exceeds $\mathcal{D}_{max}$.

First stage: Preprocessing and Mutex Reasoning

Graphplan approaches define binary mutual exclusion relations between actions and between propositions. As TGP, TPSYS needs to calculate action-action mutex relationships, action-proposition mutex and proposition-proposition mutex. Since proposition-proposition mutex appears as a consequence of action-action mutex (Blum & Furst 1997), this stage only calculates the action-action and action-proposition *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 calculus to the next stages, 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. The components of durative actions in PDDL2.1, presented in Definition 1, have some important implications for mutex reasoning. In particular, the strong mutex used by traditional temporal planners, such as TGP, must be modified to allow durative actions to be applied in parallel even in cases in which they refer to the same propositions. In traditional approaches, if two actions have interfering propositions they cannot be executed in parallel, but when dealing with PDDL2.1 durative actions it may be possible for such actions to co-occur.

There exist four action-action mutex situations, presented in Table 1. Case 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. Case 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. Case 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 are conflicting with the start conditions or effects of the other action. This mutex (which does not appear at Graphplan) 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. Furthermore, the computationally efficient testing of validity of a plan relies on not having to consider all possible orderings of so-called simultaneous happenings. This issue is discussed in depth in the PDDL2.1 semantics. Moreover, Graphplan is tailored to work with simple propositional formulae and it cannot be assumed that the positive assertion of a proposition will not interact harmfully with more complex precondition formulae. 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, case 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.

In addition to the action-action static mutex, the proposition-action mutex relationships are also calculated in the first stage. As demonstrated in (Smith & Weld 1999), when actions have different duration in a Graphplan-based approach, mutex between propositions and actions help deduce more inconsistencies because they better connect mutex between actions to mutex between propositions when actions are executed in parallel.

Definition 5 ***Static pa-mutex (proposition/action mutex)***. *One proposition p is statically mutex with action a iff $p \in \{SDel_a \cup EDel_a\}$.*

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 temporal levels of actions, represented by $P_{[t]}$ and $A_{[t]}$ respectively (Garrido, Onaindía, & Barber 2001). 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, their execution. The way of extending the temporal graph is performed in a similar way to Graphplan. Particularly, the process consists of generating all the actions a_i in action level $A_{[t]}$ of the graph as soon as their start conditions are non pairwise mutex in the proposition level $P_{[t]}$, generating their start and end effects in the proposition levels $P_{[t]}$ and $P_{[t+D_{a_i}]}$, respectively. This process finishes once all the propositions in the final situation are present, non pairwise mutex in a proposition level $P_{[t]}$, and the actions which achieved them have already ended.

Case	Condition for the mutex	Type of mutex	Relation
1	$(SAdd_a \cap SDel_b \neq \emptyset) \vee (SAdd_b \cap SDel_a \neq \emptyset)$ $((SAdd_a \cup SDel_a) \cap (SCond_b \cup Inv_b) \neq \emptyset)$ $((SAdd_b \cup SDel_b) \cap (SCond_a \cup Inv_a) \neq \emptyset)$	$AA_{start-start}$	
2	$(EAdd_a \cap EDel_b \neq \emptyset) \vee (EAdd_b \cap EDel_a \neq \emptyset)$ $((EAdd_a \cup EDel_a) \cap (ECond_b \cup Inv_b) \neq \emptyset)$ $((EAdd_b \cup EDel_b) \cap (ECond_a \cup Inv_a) \neq \emptyset)$	$AA_{end-end}$	
3	$((EAdd_a \cup EDel_a) \cap (SCond_b \cup Inv_b) \neq \emptyset)$ $((EAdd_b \cup EDel_b) \cap (SCond_a \cup Inv_a) \neq \emptyset)$ $(EAdd_a \cap SDel_b \neq \emptyset) \vee (EDel_a \cap SAdd_b \neq \emptyset)$ $(EAdd_b \cap SDel_a \neq \emptyset) \vee (EDel_b \cap SAdd_a \neq \emptyset)$	$AA_{end-start}$	
4	$(Inv_a \cap SDel_b \neq \emptyset) \vee (Inv_b \cap SDel_a \neq \emptyset)$ $(Inv_a \cap EDel_b \neq \emptyset) \vee (Inv_b \cap EDel_a \neq \emptyset)$	$AA_{during-during}$	

Table 1: Conditions for the static action-action mutex relationships between two durative actions a and b.

Modifications in the Extension of the Temporal Graph Although the idea of extending the temporal graph is conceptually simple, it contains some subtle details due to the local conditions and effects of durative actions. In each temporal level it is necessary to study first the effects achieved by the actions which end (whose *at end* conditions hold), and then the effects achieved by the actions which start. In consequence, each temporal level t is 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 must be 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 at action level $A_{[t]}$ are stored in $A_{[t]end}$, whereas the actions which start at action level $A_{[t]}$ are stored in $A_{[t]start}$. Analogously, the propositions achieved at the *end*-part are stored in $P_{[t]end}$, and the propositions achieved at the *start*-part are stored in $P_{[t]start}$.

On one hand, the mutex relationships to be calculated in the *end*-part are: $AA_{[t]end-end}$ with the actions which are mutex ending at t; $PA_{[t]end-end}$ with the propositions which are mutex with the actions which end at t; and $PP_{[t]end-end}$ with the propositions which are mutex at t after ending all the actions. On the other hand, the mutex relationships to be calculated in the *start*-part are: $AA_{[t]start-start}$ with the actions which are mutex starting at t; $AA_{[t]end-start}$ with the mutex between the actions which end and start at t; $PA_{[t]start-start}$ with the propositions which are mutex with the actions which start at t; $PP_{[t]end-start}$ with the propositions which are mutex at t and have been achieved by actions which end at t and actions which start at t, respectively; and $PP_{[t]start-start}$ with the propositions which are mutex at t after starting all the actions. The main reason for breaking down these mutex relationships into *end*-part and *start*-part lies in making their calculus simpler, as can be seen in the following definitions:

Definition 6 $AA_{[t]end-end}$. *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 7 $PA_{[t]end-end}$. *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. $\Upsilon_{i[t]} = [(p, b_i \text{ are } PA) \vee (p, ECond_{b_i} \text{ are } PP_{[t]end-end})]$. Proposition p and action a are end-end mutex at time t if the following condition holds: $\bigwedge_i [\Upsilon_{i[t]} \wedge (a, b_i \text{ are } AA_{[t]end-end})]$.*

Definition 8 $PP_{[t]end-end}$. *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 9 $AA_{[t]start-start}$. *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 10 $AA_{[t]end-start}$. *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 11 $PA_{[t]start-start}$. *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. $\Psi_{i[t]} = [(p, b_i \text{ are } PA) \vee (p, SCond_{b_i} \text{ are } PP_{[t]start-start})]$. Proposition p and action a are start-start mutex at time t if the following condition holds: $\bigwedge_i [\Psi_{i[t]} \wedge (a, b_i \text{ are } AA_{[t]start-start})]$.*

Definition 12 $PP_{[t]end-start}$. *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 13 $PP_{[t]start-start}$. *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 one action starting or ending at time t. $PP_{[t]}$ mutex represents the impossibility of having two propositions together at time t. These calculus of the mutex relationships obtains the same mutex as Graphplan and, thereafter, they provide very useful information to improve the process of search by avoiding combination of actions, propositions and propositions/actions which cannot be satisfied simultaneously, thus reducing the space search (Blum & Furst 1997).

As can be seen in the previous definitions, the calculus of the mutex relationships in the *end*-part and *start*-part are nearly identical, with the only difference of recovering and storing the information in different structures. Thus, in some cases the structures could be the same improving the efficiency. Concretely, the implementation of the second stage only keeps one structure $PP_{[t]}$ for $PP_{[t]end-end}$, $PP_{[t]end-start}$ and $PP_{[t]start-start}$.

An important point to take into account when dealing with durative actions in a Graphplan-based approach and which forced us to modify the algorithm is the condition to finish the extension of the temporal graph. In Graphplan or 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 be included 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 any of the propositions in the final situation is still conditional (it has not been validated yet), losing the benefits of the Graphplan-based graph extension. Loosely speaking, it means that the action which achieves that effect has not ended yet and the effects could be invalid (unavailable) if the *at end* conditions of the action fail. In order to tackle this drawback, it becomes 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 on the instants of time at which the propositions stop being conditional. This propagation mechanism is quite straightforward, according to the following definition:

Definition 14 ***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.*

Algorithm for the Extension of the Temporal Graph. After introducing the modifications which are necessary to extend the temporal graph, we present the algorithm (see Figure 4) for extending the temporal graph. Starting at time

Algorithm *Temporal Graph Extension*
$t = 0$
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
  endforall
  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
  endforall
  $t =$ next level in the *Temporal Graph*
endwhile

Figure 4: Algorithm for the temporal graph extension performed in the second stage.

$t = 0$ (with all the mutex structures empty), the algorithm generates new proposition and action levels (*end*-part and *start*-part), calculating all the mutex relationships. First, the *end*-part of the temporal graph is generated with the actions which can end (their end conditions are satisfied). Hence, the algorithm updates $A_{[t]end}$ with the actions which end at time t, $P_{[t]end}$ with their end effects, and calculates all the mutex relationships presented above. Then, the algorithm generates the *start*-part of the graph with the actions which can start (their start conditions are satisfied). The algorithm updates $A_{[t]start}$ and $P_{[t]start}$ with the actions which start at time t and their start effects, respectively, calculating all the mutex relationships. Here, new temporal levels are generated according to the duration of the actions generated. This way, for each b_j generated in $A_{[t]start}$, the temporal levels $P_{[e_j]}$ and $A_{[e_j]}$ are created, where obviously $e_j = t + D_{b_j}$. $No-op$ actions and delete-edges (which represent the negative effects) are not stored in the temporal graph during its extension. This extension continues until the propositions in the final situation are achieved and they are not conditional, i.e. the actions which achieve them have ended and those propositions are valid. Moreover, the extension also finishes if the maximum time allowed by the user $\mathcal{D}_{max}$ is exhausted, returning '*Failure*' (see Figure 3).

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 1 *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. Each action level contains all the actions present in the previous action levels —analogously for the proposition levels. This way, once one action b_j appears in an action level, this action will appear in the next levels, and all the temporal levels in which b_j could end and start are calculated and created.*

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 extended in the second stage. In a Graphplan-based approach the plan is obtained by moving through the graph in a backward way. The process consists of obtaining the actions which achieve the propositions to be satisfied. Now, durative actions allow different ways to achieve these propositions, not only by their *at end* effects but also by their *at start* effects. Moreover, in order to plan an action all its conditions must be satisfied, which with durative actions entails to satisfy the start, end and invariant conditions. This breaks the traditional right to left *directionality* of Graphplan or TGP as shown in the following example.

Let us suppose an instant of time t during the extraction of a plan at which a proposition p must be satisfied. Let us suppose that action a achieves p at t as a start effect ($p \in SAdd_a$). If a has end conditions ($ECond_a$), they will have to be satisfied at time $t' = t + D_a$, forcing the algorithm to move again to an already visited instant of time $t' > t$. For this reason, the algorithm first selects the set of actions $\{a_i\}$ which achieve each proposition as end effects in order to keep the traditional directionality of the search.

Moreover, before planning an action a it is necessary to study whether a is compatible with the actions already planned, i.e. that the new action a does not modify the invariant conditions of the other actions ($AA_{during-during}$ mutex relationships of Table 1), discarding a if it is not compatible.

The algorithm for the extraction of an optimal plan is shown in Figure 5. 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 starting at s_i and ending at 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. The algorithm proceeds in the following way. While there are (sub)goal propositions in $GoalsToSatisfy$, 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. Although all the set of actions $\{a_i\}$ which are compatible with actions in $Plan$ must be considered for completeness, the actions which achieve p as end effects are firstly selected to keep the traditional right to left directionality. If action a_i is not mutex with the actions in $Plan$, then a_i is planned updating the structures $Plan$ and $GoalsToSatisfy$ with a_i and the start, invariant and end conditions of a_i, respectively.

Since the temporal graph extension finishes as soon as all the propositions in the final situation are present, non pairwise mutex, and the plan extraction is complete, the algorithm obtains the optimal plan in terms of the duration of the actions (Garrido, Onaindía, & Barber 2001).

Lemma 2 ***The extraction of a plan is a complete process.***

Algorithm *Plan Extraction*
$GoalsToSatisfy = \mathcal{F}_s$ 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 $Plan$
    $Plan = Plan \cup < a_i, s_i, e_i >$
    $GoalsToSatisfy = GoalsToSatisfy \cup SCond_{a_i} \cup Inv_{a_i} \cup ECond_{a_i}$
  endif
endwhile

Figure 5: Algorithm for the plan extraction performed in the third stage.

Proof 2 *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 the plan of optimal duration.*

Proof 3 *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 by the algorithm and is optimal. This implies one of the following cases: i) the temporal level t' has not been generated during the extension of the temporal graph, or ii) the temporal level t' has been generated but the extraction stage has not considered the plan $\mathcal{P}'_{t'}$ from that level 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. In consequence, this contradicts the initial choice of the existence of $\mathcal{P}'_{t'}$. Hence, $\mathcal{P}_t$ is the plan of optimal duration.*

Application Example

We present a simple application example, based on the logistics domain *zeno-travel* presented in Figure 2. This example allows us to illustrate the extension of the temporal graph. In order to keep the temporal graph simple enough, the example to be solved consists of transporting one person, $ernie$, from $city - a$ to $city - b$ by using a $plane$ which is initially in $city - a$. The duration of the actions is 5 for $board$ and $debark$, and 10 for fly. Table 2 shows the proposition levels and the action levels. For each proposition level $P_{[t]}$, only the $P_{[t]end}$ part of the graph is shown because the actions of the domain have no positive *at start* effects —negative effects are not stored in the temporal graph. For each action level $A_{[t]}$, both the $A_{[t]end}$ and $A_{[t]start}$ are shown with the actions which end, and start, respectively at each instant of time. In time $t = 0$, actions *board(ernie,plane,city-a)* and *fly(plane,city-a,city-b)* are generated, but because they are $AA_{start-start}$ mutex the propositions *in(ernie,plane)* and *at(plane,city-b)* are mutex until time $t = 15$, in which the action *debark(ernie,plane,city-b)* is generated, thus obtaining the goal *at(ernie,city-b)* in time $t = 20$. As can be seen,

Level t	$P_{[t]}$ $P_{[t]end}$	$A_{[t]}$ $A_{[t]end}$	$A_{[t]start}$
0	*at(plane,city-a), at(ernie,city-a)*	-	*board(ernie,plane,city-a), fly(plane,city-a,city-b)*
5	*at(plane,city-a), at(ernie,city-a), in(ernie,plane)*	*board(ernie,plane,city-a)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b)*
10	*at(plane,city-a), at(ernie,city-a), in(ernie,plane), at(plane,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b),*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b), fly(plane,city-b,city-a)*
15	*at(plane,city-a), at(ernie,city-a), in(ernie,plane), at(plane,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), fly(plane,city-a,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), debark(ernie,plane,city-b), fly(plane,city-a,city-b), fly(plane,city-b,city-a)*
20	*at(plane,city-a), at(ernie,city-a), in(ernie,plane), at(plane,city-b), at(ernie,city-b)*	*board(ernie,plane,city-a), debark(ernie,plane,city-a), debark(ernie,plane,city-b), fly(plane,city-a,city-b), fly(plane,city-b,city-a)*	-

Table 2: Outline of the temporal graph extension for the application example.

although the actions have differing duration, the extension of the temporal graph is equivalent to Graphplan. The process of extraction of a plan selects the instances of actions which obtain the goals, then the start and end conditions of these actions, and so on. The plan obtained consists of the following sequence of actions:

$$\begin{array}{rll} 0+\epsilon: & \textit{board(ernie,plane,city-a)} & [5] \\ 5+2\epsilon: & \textit{fly(plane,city-a,city-b)} & [10] \\ 15+3\epsilon: & \textit{debark(ernie,plane,city-b)} & [5] \end{array}$$

The offset ϵ in the instant of time at which the actions are executed is a necessary feature for a valid plan of PDDL2.1 (Fox & Long 2001). This ϵ is included to avoid the simultaneity of the actions when they meet, as presented in the case 3 of the mutex relationships of Table 1.

Experimental Results

Currently, there does not exist an extensive collection of benchmarks for durative actions of PDDL2.1. Consequently, we have adapted some of the traditional domains of PDDL, such as *logistics*, *travel-bulldozer*, *ferry*, *gripper*, *monkey*, *blocksworld* and *zeno-travel* to the model of durative actions of PDDL2.1. Direct comparison between TPSYS and recent temporal planner such as Sapa (Do & Kambhampati 2001) or TP4 (Haslum & Geffner 2001) is difficult because they handle resources and even non-admissible heuristics which cannot guarantee the optimal solution. Nevertheless, we want to do direct comparison in the immediate future. Consequently, we compare TPSYS with TGP to demonstrate that the algorithm presented here remains feasible in dealing with traditional temporal planning problems. We use two versions of TGP: TGP, which consists of the original version of (Smith & Weld 1999), and TGP-ng, which extends TGP to keep minimal nogoods, doing backjumping during the backward search in the way proposed in (Kambhampati 2000). 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 3.

The results show that TPSYS behaves well enough in all the problems. Unlike TGP, TPSYS calculates more mutex relationships under the model or durative actions, which allows to reduce the search space in the plan extraction. This allows the complexity of TPSYS to follow the same order of magnitud of TGP —and even TGP-ng. The most important differences appear in the problems *att-log3* and *big-bull2*, in which TGP is clearly better than TPSYS. Although the differences between TGP and TGP-ng are not very significant in these tests, the benefits which can be obtained by exploiting the CSP techniques presented in (Kambhampati 2000) are very promising to dramatically improve the behaviour of the plan extraction stage.

Discussion

The temporal planning system described in this paper represents an approximation for dealing with durative actions of PDDL2.1 in a Graphplan-based approach. Therefore, most of the extensions used in Graphplan-based planners could be used here, such as memoization (Blum & Furst 1997) and regression (Kambhampati 2000) to improve the third stage, propositions in the initial (final) situation being placed (required) at any time during the execution of the plan, and exogenous events as presented in (Smith & Weld 1999).

Now, we discuss two alternative methods to tackle with durative actions with *at start* effects and *at end* conditions in a temporal planner with ability to manage instantaneous actions. Both of them consist of splitting each durative action into a collection of simple actions.

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

Problem	TPSYS	TGP (TGP-ng)
att-log0	0.42	0.02 (0.01)
att-log1	0.44	0.05 (0.01)
att-log2	0.47	0.06 (0.05)
att-log3	14.10	2.65 (2.50)
bulldozer-prob	0.88	0.55 (0.45)
big-bull1	0.58	0.80 (0.75)
big-bull2	14.31	2.15 (2.10)
ferry1	0.01	0.01 (0.01)
ferry2	0.03	0.02 (0.02)
ferry3	0.30	0.03 (0.02)
gripper2	0.03	0.03 (0.02)
gripper4	0.17	0.13 (0.16)
gripper6	6.88	4.53 (13.50)
monkey1-test	0.20	0.17 (0.15)
monkey2-test	0.63	0.75 (0.70)
tower2	0.02	0.03 (0.02)
tower4	0.28	0.45 (0.50)
tower6	2.52	3.60 (3.25)
zeno-travel1	0.01	0.01 (0.01)
zeno-travel2	0.02	0.01 (0.01)
zeno-travel3	0.02	0.01 (0.01)

Table 3: Comparison of TPSYS and TGP (results are in seconds).

represents the process of the action). All these three new actions will have neither *at start* effects nor *at end* conditions. Thus, a durative action a is divided into:

- $a1$, with no duration. $Pre_{a1} = \{SCond_a \cup Inv_a\}$ and $Eff_{a1} = \{SEff_a \cup ef_{a1}\}$.
- $a2$, with the duration of a (D_a). $Pre_{a2} = \{Inv_a \cup ef_{a1}\}$ and $Eff_{a2} = ef_{a2}$.
- $a3$, with no duration. $Pre_{a3} = \{ECond_a \cup Inv_a \cup ef_{a2}\}$ and $Eff_{a3} = EEff_a$.

The inclusion of the *artificial* effects ef_{a1} and ef_{a2} of actions $a1$ and $a2$ respectively, allows to generate the action $a2$ after $a1$, and $a3$ after $a2$, simulating the behaviour of the original action a. This way, during the plan extraction, action $a3$ only can be planned if $a2$ has been previously planned, and analogously, $a2$ only can be planned after planning $a1$. The main drawback of this method is the increment in the number of actions (in a factor of three per each durative action) and in the number of propositions (in a factor of two per each durative action) in the domain, which by itself may be prohibitive. Moreover, if one goal of the problem is satisfied by Eff_{a1}, i.e. the original $SEff_a$, only the action $a1$ would be planned (without needing to plan $a2$ nor $a3$), which would imply an unreal situation in which only a part of the indivisible action a is executed.

The second alternative is based on the semantic mapping described in (Fox & Long 2001), and consists of splitting each durative action into a collection of simple actions. The collection 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. The monitoring actions can be achieved by requiring the $no-ops$ corresponding to the invariants of an action to be active in the interval between the start and end points of that action. Therefore, they do not need to be built explicitly and only two actions have to be constructed per durative action. Doubling up the number of actions need not present a blow-up at instantiation time, because the durative actions can be instantiated first and then split, rather than vice versa. During plan extraction it is necessary to maintain the link between the actions representing the start and end points of a durative action because neither one can be exploited without the other. In addition, it is necessary to manage the temporal constraints implied by the durations of the actions. A planner based on this approach has been constructed and appears to perform well in initial experiments (Long & Fox 2001). The approach still suffers from the problem caused when the start of a durative action is added to the plan for its effect (the initial effect of the durative action) necessitating the addition of the end action to the plan if it has not already been chosen. This in turn can introduce new preconditions, so there is an iterative structure to the plan extraction algorithm. This is highly reminiscent of the DP-Plan approach (Baioletti, Marcugini, & Milani 2000) in which the directionality of Graphplan is exchanged for a Davis-Puttnam search process.

Conclusions through Related Work

Last years have seen many attempts of dealing with temporal planning. The parcPLAN approach (El-Kholy & Richards 1996) handles a rich set of temporal constraints, instantiating time points in a similar way to TPSYS. TGP (Smith & Weld 1999) 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 to speed up the rest of 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 reduces vastly the space costs but it increases the complexity of the search process, which may traverse cycles in the planning graph. In opposition, TPSYS uses a much more informed temporal graph which reduces the overhead during the search. Third, the mutex reasoning is managed in TGP by means of inequalities and sophisticated formulae, whereas TPSYS calculates the mutex relationships level by level in a more similar way to Graphplan. Finally, TPSYS uses a richer model of actions which implies: i) fewer constraints on the execution of the actions, ii) some modifications in the planning algorithm, and iii) a significantly larger space of search. More recent temporal planners, such as Sapa (Do & Kambhampati 2001) or TP4 (Haslum & Geffner 2001) handle concurrent actions and use heuristic metrics to deal with resources in planning. Sapa 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. On the other hand, TP4 uses admissible heuristic search to handle actions with time and resources, but it assumes a conservative model of actions.

This paper has presented a temporal planning system which handles durative actions provided by level 3 of PDDL2.1. Instead of using a conservative model of action, TPSYS manages actions with local conditions and effects. Although durative actions make the calculus of the mutex relationships, the temporal graph extension and the plan extraction stages more complex, they allow modelling of richer planning domains. Briefly, the main contributions of the paper have been the description of:

- The new components of level 3 durative actions based on (Fox & Long 2001) and the mutual exclusion relationships they entail.
- The modifications needed during the temporal graph extension. In the temporal graph extension, each temporal level has been divided into two parts to make easier the calculus of the mutex relationships.
- The modifications needed during the plan extraction. We have presented how the plan is found through the temporal graph without following the traditional right to left directionality.

The algorithm still has some limitations. According to our experiments, the performance of the algorithm degrades when there are many actions and propositions in the planning domain, due to the calculus of the mutual exclusion relationships. Moreover, the performance of the second stage degrades when the duration of the actions is wildly different. Particularly, the worst performance happens when the greatest common divisor of the durations of the actions is 1, which forces the algorithm to consider the maximum number of temporal levels, thus increasing the complexity of the third stage. For this reason, the areas of future work are focused on the inclusion of memoization techniques similar to the memoization performed in Graphplan and the inclusion of some of the CSP techniques presented in (Kambhampati 2000), which have been already tested on TGP. 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.

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

Baioletti, M.; Marcugini, S.; and Milani, A. 2000. DP-PLAN: An algorithm for fast solutions extraction from a planning graph. In *Proc. of AIPS*, 13–21.

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

Do, M., and Kambhampati, S. 2001. Sapa: a domain-independent heuristic metric temporal planner. In Cesta, A., and Borrajo, D., eds., *Proc. European Conf. on Planning (ECP-01)*, 109–120.

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

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

Garrido, A.; Onaindía, E.; and Barber, F. 2001. Time-optimal planning in temporal problems. In Cesta, A., and Borrajo, D., eds., *Proc. European Conf. on Planning (ECP-01)*, 397–402.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In Cesta, A., and Borrajo, D., eds., *Proc. European Conf. on Planning (ECP-01)*, 121–132.

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.

Long, D., and Fox, M. 2001. Fast temporal planning in a graphplan framework. Technical report, Dept. of Computer Science, University of Durham, UK.

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