On the inclusion of temporal capabilities in a real-world POCL planner: A successful story*

Luis Castillo, Juan Fdez-Olivares, and Antonio González

Departamento de Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática. Universidad de Granada, 18071 Granada, SPAIN.
{L.Castillo,faro,A.Gonzalez}@decsai.ugr.es

1 Introduction

Artificial Intelligence planning community is shifting its interest into more realistic domains able to successfully apply these technologies to solve real world problems. Requirements such as more expressiveness of knowledge representations or a greater efficiency of planning algorithms are some of the most common arguments in this direction. Following the line of achieving more expressiveness, this paper describes a step in this direction and tells a story on the successful inclusion of PDDL 2.1 level 3 [6] temporal capabilities into a non-temporal, partial order, causal link-based planner (POCL) called *MACHINE* which has been used to solve real-world manufacturing problems [1, 2]. Problems on the modeling of time and the control of concurrency are addressed providing some extra functionality that ends to be more realistic than PDDL 2.1.

A question that arises during the definition of these temporal capabilities is: "Is the definition of a temporal ontology necessary to build a temporal planner?". This paper, following others such as [11, 10, 3], shows that a temporal planner can be easily built on top of a POCL planner only by adding some temporal managing capability and also that POCL planning can be a more intuitive platform to reach a temporal planning architecture than GraphPlan-based platforms [12, 7] or state based-planning [9, 5].

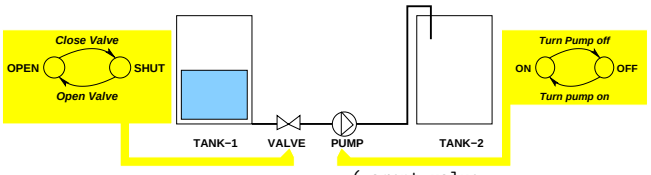
2 An overview of MACHINE

MACHINE is a POCL refinement planner[1] [2]. Its domains are composed of agents and every agent has a set of available actions. *MACHINE* is based on a *closed change assumption*, that is, there are no exogenous effects and agents are the only entities able to change the environment. The behavior of every agent is described by means of a Finite State Automaton (FSA) so that every action implies both a change of state in the agent and some effects in the environment (see Figure 1). Both the change of state and the effects are considered to be instantaneous.

Unlike most of the models of action, in which the end of an action occurs somewhere between the execution of two consecutive actions of the same agent

* This work has been supported by the CICYT under project TAP99-0535-C02-1.

[1] It does not support conditional effects, numeric values nor metric time

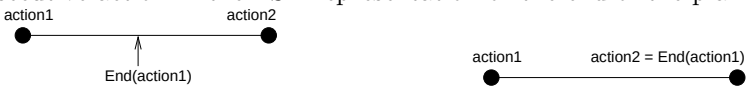



```
(:agent pump
   (:states (off on))
   (:action pump-on
    :parameters (?CHEM)
    :pre-condition
       (and (contains ?CHEM tank-1)
            (state pump off))
    :sim-condition
       (open-flow tank-1 tank-2)
    :effects
       (and (not (contains ?CHEM tank-1))
            (not (state pump off))
            (contains ?CHEM tank-2)
            (state pump on)))
   (:action pump-off
    :parameters nil
    :pre-condition
       (state pump on)
    :sim-condition nil
    :effects
       (and (not (state pump on))
            (state pump off))))
```

```
(:agent valve
   (:states (shut open))
   (:action open-valve
    :parameters nil
    :pre-condition
       (state valve shut)
    :sim-condition
       nil
    :effects
       (and (not (state valve shut))
            (open-flow tank-1 tank-2)
            (state valve open)))
   (:action close-valve
    :parameters
       nil
    :pre-condition
       (state valve open)
    :sim-condition
       nil
    :effects
       (and (not (state valve open))
            (not (open-flow tank-1 tank-2))
            (state valve shut))))
```

Fig. 1. A simple domain showing a valve and a pump to transport the content of tank-1 into tank-2 and its description

(sometimes it is a timed event, sometimes it is given by the achievement of all of the effects of the action), *MACHINE* is based on the Newtonian assumption that once an action has started its execution, it will continue so until some other planned event interrupts it. In this case, this ending event can only be either a consecutive action in the FSA representation or the end of the plan.



Following this assumption, actions are easily considered like *intervals*: the qualitative period of time between an action and either the following action of the same agent or the end of the plan. For example, a plan to transport the content of `tank-1` into `tank-2` could be the one shown in the chronogram of Figure 2. This plan has four actions whose intervals of execution are `[Open Valve, Close Valve = End(Open Valve)]`, `[Close Valve, END = End(Close valve)]`, `[Pump On, Pump Off]` and `[Pump Off, END]`. This leads to a very easy interpretation of *concurrency*: two actions execute concurrently if their intervals of execution may overlap. For example, actions `[Open Valve, Close Valve]` and `[Pump On, Pump Off]` are concurrent. The idea of intervals of execution defined as intervals of actions of the same agent, and the possibility of these intervals to overlap, provide the basis of refinement operations like subgoal achievement and flaw detection and solving.

Actions can have pre-conditions, i.e., conditions which must hold at the beginning of the action, like PDDL preconditions [6], and simultaneous conditions

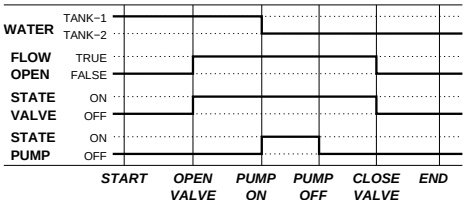


Fig. 2. The chronogram of a simple plan

(see Figure 1), i.e. conditions which must hold during its whole interval of execution, like PDDL level 3 invariant conditions. The achievement of these two types of conditions produces two different types of causal links, a previous causal link from the producer action [13] until the consumer action; or a simultaneous causal link from the producer action until the end of the consumer action [2]. For example, the simultaneous condition `(open-flow tank-1 tank2)` of action `Pump On` is satisfied by action `Open Valve` and protected during the interval `[Open Valve, Pump Off=End(Pump On)]`.

There are two different types of flaws that *MACHINE* is able to detect and solve. On the one hand, classic threats [13] are redefined to cope with these intervals of execution and these two types of causal links. Then, a threat appears when the interval of actions of some causal link may overlap with the interval of execution of some action that deletes the literal of the causal link. In this case, usual methods of promotion and demotion are used to solve the threat but taking into account the starting and ending points of both actions and causal links. On the other hand, *MACHINE* can handle interferences. An interference occurs when the intervals of execution of two actions overlap and both actions have some opposite effect, say p and *not* p. In this case promotion and demotion are used as explained before.

In summary, *MACHINE* is a non-temporal POCL planner that has been used to solve realistic problems in manufacturing domains, whose semantics is based on intervals of execution, defined as intervals between actions, thanks to the expressiveness provided by the FSA representation of agents. These intervals of actions are the basis of goal satisfaction and flaw detection and solving. In the following, the paper presents the extension of *MACHINE* to handle metric time, PDDL2.1 durative actions [6] and some more features.

3 Temporal aspects of the domain

Although *MACHINE* is able to solve general problems, it has been widely used to solve manufacturing problems [1, 2], mainly those in which the transition from one action to another in the plan depends exclusively on the satisfaction of a set of conditions. However, since it is not able to represent metric temporal relations, it is not either able to solve problems which require some type of metric temporization in the plan, such that some real-time processes or time critical applications. In order to solve this new type of problems, and given that the model of action and the planning architecture of *MACHINE* is quite specific, the main question was is: should we define a new temporal planning architecture

or reuse the existing one by adding temporal capabilities? The right choice is the last one. The reason of this is that the most important features of one of the reference documents in temporal domains (PDDL 2.1 level 3 [6]), like durative actions and invariant conditions, are already supported by the non-temporal model of action of *MACHINE*, thus only some sensitive extensions are needed to cope with the metric temporal features of domains.

The first issue is that, given that a domain is represented as a set of agents whose operation is defined by means of a FSA, one may assume that changes of state in these automata are instantaneous since they represent internal states. However, the remaining effects on the environment do not have to be instantaneous, that is, they might have some delay. During this delay, the effect can be considered to be undefined, so no continuous effects are going to be represented. This fact cannot be modeled explicitly in PDDL 2.1 level 3 since effects occurs instantaneously at the beginning or at the end of a durative action but it will have some important consequences in the metric interleaving of actions in a temporal plan. Following this assumption about delayed effects, once a producer action satisfies a subgoal of a consumer action, the consumer action only has to wait *at least* the delay needed for the producer action to achieve the effect. This would allow to introduce *temporal restrictions between producers and consumers as a continuous*, not necessarily restricted to the start or end points of the producer like in PDDL 2.1, thus achieving a more realistic interleaving. In addition, it must be taken into account that this will introduce uncertainty in the model, that is, once all of the conditions of a consumer action have been achieved at time t, the consumer action can be executed *at some arbitrary time point* greater than or equal to t whenever it did not interfere with the rest of the plan.

Another issue that has to be considered is that, although continuous change is not explicitly considered, there are actions whose effects are somehow accumulative, and that they must have exactly a given duration for they to achieve exactly the desired effect. This is the case of motion actions, including transportation actions, or the pumping of some fluid like in Figure 1. In this case there must be a restriction on the expected duration of an action, that is, in the interval `[action, End(action)]`. In contrast to this, non-accumulative actions do not need to have a fixed duration.

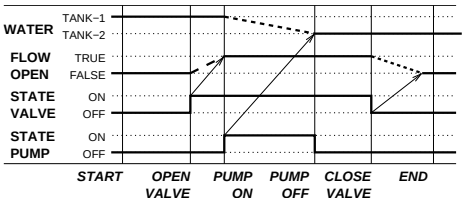


Fig. 3. A more realistic chronogram than the one shown in Figure 2

Let see an example. Figure 3 shows a more realistic view of plan in Figure 2. The duration of the interval `[Open Valve, Pump On]` is the delay needed to open the valve, but it can be greater. The duration of the interval `[Pump On, Pump Off]` must be exactly the time needed to transport the content of tank-1 into

tank-2, no more, no less. The duration of interval `[Open Valve, Close Valve]` is undefined but restricted, given that it contains a fixed duration interval.

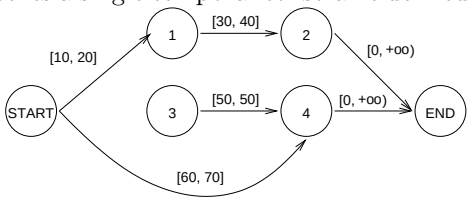
Finally, the specification of some problems might contain subgoals with deadlines, that is, subgoals that must be achieved exactly at a given time point in the plan. This is very important in some time critical domains such manufacturing or transportations, although it is not explicitly described in PDDL 2.1. These are the temporal aspects of the domain that need to be modeled in the new version of *MACHINE* called $MACHINE^T$. Next section explains how.

4 PDDL 2.1 durative actions and more

The main decision that has to be made is the choice of the underlying structure to represent a temporal order relation given the temporal restrictions that need to be represented. In this case, simple temporal networks [4, 11] are a very good choice.

4.1 Simple temporal networks

A simple temporal network [4] is a directed graph where vertices are time points and every arc represents a single temporal constraint defined on two time points.



They have been chosen because they have the following features: they allow to represent and propagate temporal relations in a partially ordered structure, relative or absolute constraints can be posted, flexible constraints can also be posted between time points in the form $[t_{min}, t_{max}]$ meaning that the distance between two time points is not exactly defined, but restricted by lower and upper bounds, exact temporal constraints can also be posted in the form $[t, t]$, partially known relations can be represented in the form $[t, +\infty)$, successive postings of constraints between two time points are easily solved by intersecting the constraints. These networks also provide all of the functionality needed for a POCL planner, that is, consistency checking, question answering in the form "is time point $1 \leq$ time point 2?" or "do intervals $[timepoint_1, timepoint_2]$ and $[timepoint_3, timepoint_4]$ overlap?" and propagation of constraints in polynomial time.

Time points could be used to represent start and end points of both actions and fluents in a plan. However this can lead to a slow propagation process in large plans. Given the *closed change assumption* of $MACHINE^T$, time points can be used to represent actions but taking into account that every temporal constraint between actions strongly depends on the existing delays of effects. This can achieve the same expressiveness but implies a much simpler network and a lower computational effort to propagate constraints.

4.2 A representation of time

The temporal aspects of the domain presented above are modelled in the following way. Every effect of an action will be a temporally annotated literal of the form $(literal, delay)$ to represent the delay associated to this effect expressed in time units. Fixed duration of accumulative-type actions are explicitly represented with a numeric value in a new argument of the action. Actions that do not have a predefined duration will have an $+\infty$ value (see Figure 4). Deadline goals will also be represented as temporally annotated literals.

```
(:agent pump
   (:states (off on))
   (:action pump-on
    :parameters (?CHEM)
    :max-duration 120
    :pre-condition
       (and (contains ?CHEM tank-1)
            (state pump off))
    :sim-condition
       (open-flow tank-1 tank-2)
    :effects
       (and ((not (contains ?CHEM tank-1)) 120)
            ((not (state pump off)) 0)
            ((contains ?CHEM tank-2) 120)
            ((state pump on) 0)))
    ...
```

```
(:agent valve
   (:states (shut open))
   (:action open-valve
    :parameters nil
    :max-duration *POS-INF*
    :pre-condition
       (state valve shut)
    :sim-condition
       nil
    :effects
       (and ((not (state valve shut)) 0)
            ((open-flow tank-1 tank-2) 10)
            ((state valve open) 0)))
    ...
```

Fig. 4. A temporal representation of the domain in Figure 1

4.3 Goal satisfaction and causal links

In addition to the goal satisfaction and causal link definition processes of MACHINE, the following extensions have been added to $MACHINE^T$.

- Every time an action a is used as producer of a temporally annotated effect (l, δ_l) for a subgoal of a consumer action b, a restriction $[\delta_l, +\infty)$ is added to the simple temporal network between the time points associated to a and b and propagated accordingly.
- Every action a with a fixed duration $t \neq \infty$ adds a new constraint of the form $[t, t]$ between the time points associated to a and $End(a)$. There may be actions with no fixed duration, in this case, for every such action a, a constraint is added to specify that this action must take at least the time needed to achieve its effects, that is, a constraint of the form $[max_{l \in effects(a)} \delta_l, +\infty)$ is added between time points of a and $End(a)$.
- For every deadline goal, as a temporally annotated goal (g, t_g), meaning that goal g has to be achieved exactly at time t_g, which has been solved by an action a with a temporally annotated effect (l, δ_l) a constraint is added of the form $[t_g - \delta_l, t_g - \delta_l]$ between the time points of action $START$ and a, given that time point of action $START$ has an absolute time zero[2].

[2] Note that in this model there is no slack in the satisfaction of deadline goals since, by their own nature, they must be satisfied exactly at a fixed time point.

These extensions give temporal interleaving capabilities to the problem solving process of *MACHINE*^{T}. Even more, two of the most important features of PDDL 2.1 level 3 durative actions are easily achieved. On the one hand, the FSA representation of agents provide the underlying knowledge to easily define start and end points for intervals of actions without requiring the insertion of dummy actions or the decomposition of the action into two different time points like in some PDDL 2.1 level 3 based planners [7, 3, 10, 8]. Given an action a_i, an action $a_j = End(a_i)$ can also be included in a plan and the duration of a_i will be the distance in the simple temporal network between time points associated to a_i and a_j. This distance can be fixed ($[t, t]$), restricted ($[t_{min}, t_{max}]$) or undefined ($[t, +\infty)$). Therefore, every action maintains its own preconditions, without the need to distinguish between *start conditions* or *end conditions*. This is particularly important for actions without a fixed duration time.

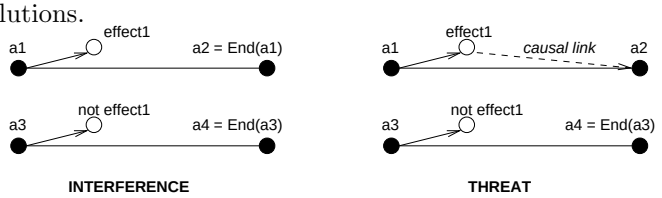
On the other hand, invariant conditions, i.e., conditions which must hold during the interval of execution of an action are already represented and satisfied like simultaneous causal links in *MACHINE*^{T} (see Section 2). The inclusion of a metric time only gives a numeric value to these intervals. This is particularly easy in POCL planners like *MACHINE*^{T} or *TANDOR*[10] or [3] thanks to the use of causal links and a continuous time-line available for promotions or demotions, but it is more difficult to represent in GraphPlan-based planners [7, 12] where some extra mechanisms need to be added to account for the preservation of a condition *through* several layers of the planning graph, or state-based planners [9, 5] where the time-advancing procedure has to be modified with both additional data structures (set of persistent conditions and/or their associated actions) and some modifications of the time advancing procedure to handle these structures.

In addition to this, *MACHINE*^{T} provide an extra functionality regarding the metric relations between actions. The use of the delay of effects to introduce temporal constraints between the producer and the consumer, as explained before, is also used in *SAPA* [5] and it allows a continuous and more precise temporal ordering of producer and consumer actions than in PDDL 2.1, where consumer actions must be ordered only after the start or end time points of a durative action depending on whether the effect produced is *at start* or *at end*. For example, given the use of the producer action `Open Valve` and its delayed effect `((open-flow tank-1 tank-2) 10)` which takes 10 time units, to solve the same subgoal of the consumer action `Pump On` (see Figure 4) would add a temporal constraint of $[10, +\infty)$ between `Open Valve` and `Pump On`. This also has some important consequences on the treatment of harmful interleavings of actions.

4.4 Concurrent actions and flaws

The notion of concurrency in *MACHINE*^{T} maintains the meaning of interleaving intervals as explained before but now, it has a metric evaluation. For metric positive concurrency, there is not much to add, but there is something new about harmful concurrencies, threats and interferences. These cases are commonly called mutex actions in PDDL 2.1. In *MACHINE*^{T} they are detected following the same mechanism, that is, checking for possible overlapings of intervals of actions (two intervals of execution in the case of interferences and a causal link

and an interval of execution in the case of threats) and they have qualitatively different solutions.



In the case of interferences, promotion and demotion are applied taken into account the delays of the interfering effects in such a way. Say that action $a1$ with the delayed effect (l, δ_1) interferes with action $a3$ with the delayed effect $((not\ l), \delta_2)$. Then promotion is applied by adding the following constraint $[\delta_1, +\infty)$ from $a1$ to $a3$. Demotion consist in adding $[\delta_2, +\infty)$ between $a3$ and $a1$.

In the case of threats, promotion and demotion are similar. Say that action $a3$ with the delayed effect $((not\ l), \delta)$ interferes with the casual link (previous or simultaneous) from $a1$ to $a2$. Then promotion adds the constraint $[max_{l \in effects(a2)} \delta_l, +\infty)$ between $a2$ and $a3$. Demotion adds the constraint $[\delta, +\infty)$ between action $a3$ and action $a1$.

In both cases it should be clear that mentioned constrains are "added" to the existing constraints between the involved actions, intersected and propagated accordingly through the simple temporal network. If one of this operations causes any inconsistency, it will be detected by the consistency checking procedure of the network [4]. In any case, this method for detecting and solving harmful concurrencies is more adequate for real-world or time critical problems than the more restrictive ones proposed in PDDL 2.1 and TGP [12] since conflicts do not have to appear necessarily at start or end points, but also in any intermediate point.

4.5 Temporal Plans

Finally, plans will be a set of actions distributed on a simple temporal network such that every action has an annotated time at which its execution is expected (it is given by the existing constraint between the action itself and the action $START$) and the rest of constraints between actions are the result of constraint posting during the planning process as described before. The execution time for every action can be an exact time point of the form $[t, t]$, a bounded time interval $[t_{min}, t_{max}]$ or an undefined interval $[t, +\infty)$ (See Figure 5).

This temporal plan can be used in two ways. The first one is to know the exact time point at which an action should be launched. In this case, the algorithm for solutions extraction for simple temporal networks [4] has to be used since there may be several possible solutions. The second way is to monitor the execution of the plan. A temporal plan can be used to check if the execution of any action has been delayed for some reason (faults, unexpected delays, etc) and whether the existing delay is acceptable or not in order to continue the execution of the plan or abort. In this case, since every action has a time interval in which its execution is expected to start, if the delay is still in the correct bounds then it is acceptable, otherwise it is not.

4.6 Other issues

This temporal planning framework can also support several additional features. Firstly, makespans for the temporal plan can be very easily achieved (see Figure 5) by adding a constraint of the form $[makespan_{min}, makespan_{max}]$ between actions $START$ and END at the beginning of the planning process (in this case, too long plans will produce an inconsistency with respect to this constraint, detected by the consistency checking mechanism and rejected). $MACHINE^T$ could also be used to optimize the final makespan just by including the duration of the plan in the heuristic evaluation function, although this feature needs further study.

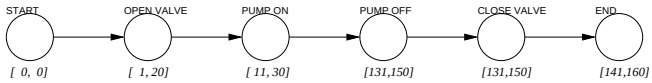
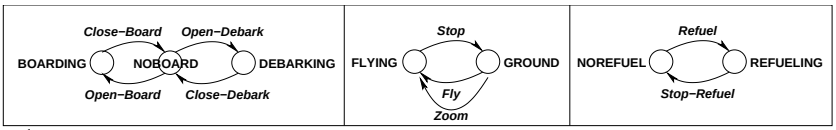


Fig. 5. A temporal plan for the domain in Figure 4 taking into account a maximum makespan of 160 time units

$MACHINE^T$ can also be used for more general domains, not only manufacturing systems, with the only requirements of representing agents by means of a FSA. In this case, Figure 6 shows the representation of the very well known zeno travel problem from the international planning competition held at AIPS'02[3] in terms of FSA, and a portion of the domain.




```
(:agent plane1
   (:states (ground flying norefuel refueling
             noboard boarding debarking))
   (:action fly
    :parameters (?city1 ?city2)
    :max-duration 180
    :pre-condition
       (and (plane-at plane1 ?city1)
            (refueled plane1)
            (state plane1 ground))
    :sim-condition
       nil
    :effects
       (and ((not (plane-at plane1 ?city1)) 0)
            ((not (state plane1 ground)) 0)
            ((not (refueled plane1)) 180)
            ((plane-at plane1 ?city2) 180)
            ((state plane1 flying) 0)))
   ...
```

```
...
(:action refuel
 :parameters nil
 :max-duration 73
 :pre-condition
    (state plane1 norefueling)
 :sim-condition
    (state plane1 ground)
 :effects
    (and ((not (state plane1 norefuel)) 0)
         ((refueled plane1) 73)
         ((state plane1 refueling) 0)))
...
```

Fig. 6. The zeno travel domain expressed in terms of $MACHINE^T$

[3] http://www.dur.ac.uk/d.p.long/competition.html

5 Conclusions

This paper has been intended to show that POCL planning is still a valid platform to face real world problems and that it is a very expressive paradigm to efficiently support temporal planning problems, when it is complemented with some temporal representation and reasoning capabilities. In addition, it has shown that a non-temporal model of action, such as that of *MACHINE*, can be a very good start point to include some of the most important features of PDDL 2.1 level 3 durative actions and even some additional interesting features.

References

1. L. Castillo, J. Fdez-Olivares, and A. González. A three-level knowledge based system for the generation of live and safe petri nets for manufacturing systems. *Journal of Intelligent Manufacturing*, 11(6):559–572, 2000.
2. L. Castillo, J. Fdez-Olivares, and A. González. Mixing expresiveness and efficiency in a manufacturing planner. *Journal of Experimental and Theoretical Artificial Intelligence*, 13:141–162, 2001.
3. A. Coddington. Handling durative actions in a continuous planning framework. In *AIPS'02 Workshop on Planning for Temporal Domains*, pages 33–40, 2002.
4. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
5. M. Do and S. Kambhampati. SAPA: a domain-independent heuristic metric temporal planner. In *European Conference on Planning*, pages 109–120, 2001.
6. M. Fox and D. Long. PDDl2-1: an extension to PDDL t for expressing temporal planning domains. Technical report, University of Durham, UK, 2001.
7. M. Fox and D. Long. Fast temporal plannig in a Graphplan framework. In *AIPS'02 Workshop on Planning for Temporal Domains*, pages 9–17, 2002.
8. A. Garrido, E. Onaindía, and F. Barber. Time-optimal planning in temporal problems. In *European Conference on Planning*, pages 397–502, 2001.
9. P. Haslum and H. Geffner. Heuristic time with time and resources. In *European Conference on Planning*, pages 121–132, 2001.
10. E. Marzal, E. Onaindía, and L. Ssebastiá. An incremental temporal partial-order planner. In *AIPS'02 Workshop on planning for temporal domains*, pages 26–32, 2002.
11. E. Rutten and J. Hertzberz. Temporal planner = nonlinear planner + time map manager. *Artificial Intelligence Communications*, 6:18–26, 1993.
12. D.E. Smith and D.S. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI'99*, pages 326–337, 1999.
13. D. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.