A Continuous Planning Framework with Durative Actions

Alexandra Coddington
Department of Computer Science
University of Durham
Durham, DH1 3LE, United Kingdom
A.M.Coddington@durham.ac.uk

Abstract

This paper describes a continuous planning framework to be used by a planning agent situated within an environment, which reasons about goals with priorities and deadlines, and actions with duration. The framework assumes that goals may be generated continuously, which requires the interleaving of planning and execution. Constraints upon time may mean it is not possible for all goals to be achieved – as a consequence the planning agent must be able to prioritise its goals. A crucial component of this framework is a temporal manager which enables the planner to reason about whether or not there is sufficient time available to achieve all goals, and to calculate deadlines for actions and outstanding subgoals. The main contribution of this paper is an examination of the way in which the partial order planning paradigm could be extended to reason with PDDL2.1 level 3 durative actions.

1. Introduction

The focus of much work on planning is primarily concerned with developing planners which are efficient and which produce plans that are both sound and complete [1, 15]. In these systems, goals are presented by an external user, and planning ceases either once a plan has been generated which achieves these goals or if the planner fails to generate a plan. There is no facility for the achievement of new goals that are presented to the planner once the planning process has commenced. In addition, once a plan has been created it remains unexecuted. The framework introduced in this paper has been designed for an agent situated within an environment which requires continuous planning. It is assumed that the agent has a set of *motivations* which enable it to generate and prioritise goals. New goals may be posed to the system while it is planning, which requires planning and execution to be interleaved. Goals have deadlines and actions take time to execute which means a core component of the framework is responsible for reasoning about whether or not goals may be achieved by their deadlines, and for assigning deadlines to actions and subgoals. Goals also have an associated priority so that instead of simply failing to return a plan if there is insufficient time to satisfy one or more goals by their deadlines, the system is able to abandon the achievement of some goals in favour of achieving others. Finally, the agent's *motivations* may be used as part of a heuristic to enable the planner to select the most promising plan for further refinement.

Recently there has been a renewed interest in the Artificial Intelligence Planning community in temporal planning, where plans contain actions with duration and goals with deadlines. The Planning Domain Description Language (PDDL) developed initially in 1998 for the first International Planning Competition held during the Artificial Intelligence Planning & Scheduling conference (AIPS98) has been extended – the current version, PDDL2.1 level 3, allows planning domain modellers to specify actions with duration. As a consequence, a number of new planning domains have been developed which include actions with duration (or durative actions) for the 3rd International Planning Competition held during AIPS 2002. This paper describes how the classical partial order planning paradigm may be extended to reason with PDDL2.1 level 3 durative actions with a view to incorporating such extensions within the continuous planning framework.

This paper presents the continuous planning framework in section 2 and then describes *motivations* and how they enable the planning agent to generate and prioritise goals, and select good plans for subsequent refinement. The core component, the *Plan to achieve goal* component which uses an extended partial order planning paradigm is presented and discussed. The remainder of the paper then describes how the partial order planning paradigm could be extended to reason with the extra expressive power of PDDL2.1 level 3 durative actions. Finally, conclusions and further work are discussed.

Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME'02)
1530-1311/02 $17.00 © 2002 IEEE

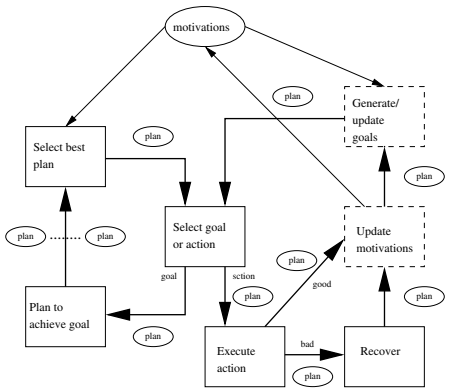

Figure 1. The continuous planning/execution framework

2. The continuous planning framework

The continuous planning framework is illustrated in figure 1 and was designed for an autonomous agent (such as a robot) situated within an environment. An underlying assumption is that the agent (via the *Generate/update goals* component) can generate new goals in response to its current context (this is encapsulated within the current initial state model, the plan, as well as the agent's motivations). For the purpose of this paper it is assumed that this component can be emulated by a user keying in new goals. Solid rectangular boxes represent the various processes in the framework that are the focus of this research – these processes have been implemented using Allegro Common Lisp. The dashed boxes represent two components responsible for updating the agent's motivations and for generating goals. These have not been implemented and will not be discussed in this paper – details can be found in [11, 2]. The ovals represent knowledge sources – these represent the current plan, (including the initial and goal states as well as a set of actions and constraints) and the agent's motivations.

This framework can be viewed as a dynamic system in which the agent continually generates goals in response to its perceived current and predicted future states of the environment as well as in response to its motivations. Each newly generated goal has a deadline by which it must be achieved, as well as a value indicating its importance or priority. Newly generated goals are added to the representation of the planning problem. The process *Select goal or action* determines whether one of the goals should be achieved or whether one of the actions (belonging within a plan) should be executed. When this process chooses to achieve a goal, the goal is passed to a planner which plans to achieve the goal. An important part of the planning process involves determining whether or not goals may be achieved by their deadlines as well as assigning deadlines to actions and subgoals. When planning to achieve a goal, the planner generates a search space of alternative plans which requires a plan evaluation metric to select the most promising plan for further refinement (*Select best plan*). This metric is designed to take into account the total duration, the number of actions, the number of high priority goals that have been achieved, as well as the degree to which the plan supports the agent's motivations.

When a decision is made to execute an action, the *Execute action* component updates the plan and the model of the current state to reflect the changes that have occurred following execution. If the actual outcome differs significantly (i.e. enough to undermine the plan in some way) from the predicted outcome, the component *Recover* is responsible for repairing the plan. In addition, as a consequence of changes to the environment and plan following execution, the agent's motivations may change (these are updated by the component *Update motivations*), which in turn may cause new goals to be generated or existing goals to be updated.

The continuous planning framework is similar to Sage [8] – an extension of UCPOP [12] which supports simultaneous actions execution and which integrates planning and execution. Sage however, does not reason about time or take into account the context of the planning agent when choosing plans for subsequent refinement. The remainder of the paper will focus mostly on the component responsible for generating plans. A full description of the remaining components can be found in [2].

3. Motivations

Motivations may be thought of as long term higher level drives or emotional states which direct an agent which is situated within an environment to both plan and act. Associated with each motivation is a value indicating its current weight – this value changes over time and provides a driving force directing the generation of goals to satisfy the motivation. For example, a truck-driving agent might have a motivation concerned with conserving fuel. The weight associated with this motivation depends upon the level of fuel in the truck – as the level of fuel decreases and falls below some threshold, a goal to replenish the fuel supply will be generated. The weight associated with motivations has a direct effect upon the priority of goals generated to satisfy those motivations. Finally, whilst acting to achieve goals, the agent may cause changes to its motivations. Through executing an action, an agent may support or undermine its motivations. For example, the truck-driving agent will undermine the motivation to conserve fuel when it drives from one location to another, and will support that motivation when it refuels. It is therefore possible to evaluate

the degree to which the actions in a plan (and therefore the plan itself) support or undermine the agent's motivations. This is exploited as part of a plan evaluation metric. In summary, motivations, together with the agent's context (the perceived current state and predicted future states captured in the plan), enable the agent to generate and prioritise goals with deadlines, and enable the agent to evaluate plans, favouring plans which best support the motivations. Full details concerning motivations and the way in which they cause goals to be generated can be found in [11].

4. Planning to achieve goals

Figure 2 shows the subcomponents of the planner (*Plan to achieve goal*) in more detail. A goal (selected by the component *Select goal or action*) is presented to the planner which generates a plan to achieve that goal. Currently, the planner uses an extended partial order planning paradigm (for example SNLP [9]). Once a plan has been generated, the component *Estimate deadlines* is responsible for both determining whether there is sufficient time available to achieve all goals in the plan as well as for assigning deadlines to actions (and therefore subgoals). If this process fails it means there is insufficient time available to achieve all of the goals within the plan in which case the plan is edited to remove a goal, together with its associated actions and constraints. Once a plan has been edited, the *Estimate deadlines* component reestimates deadlines. When the temporal component has successfully assigned deadlines to actions, the process is complete.

The *Plan to achieve goal* component of the continuous planning framework was implemented using an extended partial order planning paradigm for several reasons. Firstly, partial order planners output plans that offer a higher degree of execution flexibility than those generated by Graphplan [1] and state search planners and are arguably better suited for frameworks in which planning and execution are interleaved [10]. In particular, the set of persistence constraints (which maintain the truth of preconditions), may be used when monitoring the outcome of execution to see whether the remaining plan is still valid. In addition, partial order planning is arguably more suited than Graphplan or state search planners to the requirement that new goals may be generated during the planning process as the new goals may simply be added to the set of outstanding goals without affecting the planning process. Graphplan style planners, in contrast, would have to recommence the plan extraction process to take into account the new goals. Smith [14] argues that partial order planners offer a more promising approach for handling domains with durative actions, temporal and resource constraints. The main drawback of partial order planning has been the lack of a good heuristic for selecting plans for further refinement; search control is of fundamental importance for partial order planning. However, recent work by [10] challenges the prevailing pessimism about the scalability of partial order planning by presenting novel heuristic control techniques.

In order to implement the *Achieve goal* component the partial order planning paradigm has been extended in two ways to support the fact that goals have an associated deadline and a value indicating their importance, while actions have duration.

- When creating a new action or further instantiating an existing action in order to achieve a goal it is necessary to estimate the duration of that action. The duration of an action may depend upon the values assigned to its parameters. For example, the duration assigned to an instance of the operator schema *drive-to(?x, ?y)* will depend upon the values assigned to the variables *?x* and *?y*. In this case, the exact duration can only be determined when both *?x* and *?y* have been instantiated. In the current implementation of the framework, a worst case estimate of the duration of each incomplete action instantiation is provided which requires a degree of domain knowledge. For example, if the domain contains a network of locations within a town, the worst case estimate of the duration associated with instances of the *drive-to(?x, ?y)* would be the time taken to travel between the two furthest apart locations.

- In order to edit the plan, a record must be kept of the dependencies that exist between actions and goals. Currently, each action contains a list of the goals to which they contribute.

- Goals have a value indicating their priority. Actions which contribute to those goals are also assigned a value indicating their priority – if an action contributes to a single goal, the action inherits the value indicating the importance of that goal, if the action contributes to more than one goal it is assigned the sum of the values indicating the importance of each goal. Preconditions of actions have the same priority as their associated action.

- Actions are assigned values indicating the degree to which they support the agent's motivations (for further details see [2]).

The purpose of the *Estimate deadlines* component is twofold: to enable the *Select goal or action* component to determine which goal is to be achieved or whether an action is to be executed; to reason about whether there is sufficient time available to achieve all goals by their respective deadlines. The algorithm adopts a pessimistic approach when estimating deadlines – if actions are only partially ordered with respect to each other those actions are each assigned

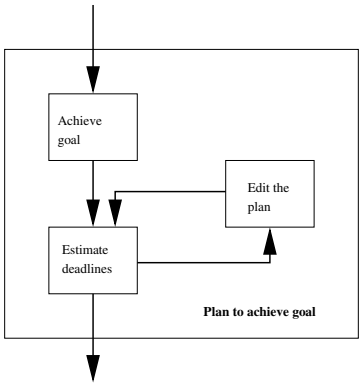

Figure 2. Planning to achieve a goal

the earliest possible deadline. For example, if three actions a_1, a_2 and a_3, each with a duration of 3 minutes and which remain unordered with respect to each other, are chosen to achieve a goal with the deadline 17:00, the algorithm estimates the deadline for each action to be 16:51. This means that during the early stages of plan refinement the deadlines estimated for actions and subgoals are likely to be too early. This is in contrast to DEVISER [16] which estimates execution windows bounded by the earliest possible execution time (which is too early) as well as the latest possible execution time (which is too late). If there is insufficient time available to achieve a goal, the algorithm fails and returns the goal.

If there is insufficient time available to achieve a goal, the *Edit the plan* component removes that goal together with its associated actions and constraints. This requires a record of the dependencies between actions and goals to be maintained as described in section 4. Once a plan has been edited, deadlines are reestimated for the actions in a plan. Should there still be insufficient time available to achieve all goals, the plan is again edited. This process continues until there is sufficient time available to execute the remaining plan.

Finally, a plan evaluation heuristic has been implemented which takes into account the degree to which a plan supports the agent's motivations, the number of higher priority achieved goals, the total execution time, as well as the number of actions. Further details of the heuristic may be found in [2].

4.1. Discussion

The decision to edit the plan once it has been determined that there is insufficient time available to achieve all of the goals, was based upon the desire to emulate human decision making – humans tend to abandon some goals in favour of others if there is insufficient time available to achieve all goals. It seems preferable to preserve as much of the original plan as possible as opposed to replanning from scratch. However, this approach has various associated costs – in particular, it is necessary during the planning process to maintain a record of the dependencies between actions and goals to facilitate plan editing. In addition, once a plan has been edited, deadlines have to be reassigned to the remaining actions in the plan, and, if there is still insufficient time available, the cycle of editing and reassigning deadlines is repeated. An alternative approach would be to simply replan from scratch, once it is established that there is insufficient time available to achieve all goals, by presenting only a subset (selected by taking into account the priorities and deadlines of each goal) of the original set of goals (generated by the component *Generate/update goals*) to the planner. In the future it is intended to perform a set of experiments to determine whether or not the decision to edit the plan is more or less efficient than replanning from scratch. If replanning from scratch proves to be less costly, some of the main benefits of partial order planning such as being able to plan to achieve goals using a skeletal partial order plan, will be lost.

5. Partial order planning for PDDL2.1 level 3 durative actions

The partial order planning paradigm used as the basis for the *Plan to achieve goal* component of the continuous planning framework makes the classical temporal planning assumption whereby actions with duration are viewed as a black box – the preconditions of durative actions must be true at the starting point of execution and remain true throughout the interval during which the action is executed, while effects become true at the end point of execution but are undefined during the interval of execution. Both temporally extended GraphPlan based systems such as TGP [15] and partial order planners such as DEVISER [16] make this assumption about durative actions. However, this assumption excludes many valid plans as there is no way of syntactically distinguishing between preconditions (propositions that are required to be true only until the starting point of the action) and invariant conditions (propositions that are required to remain true throughout the interval of execution), while effects are only defined at the end point of execution.

In contrast, PDDL2.1 level 3 durative actions [6] provide greater expressive power by allowing the domain modeller to specify local pre and postconditions of the end-points of the interval over which execution of the durative action takes place, as well as any invariant conditions that must hold throughout that interval [6]. This is achieved by using temporally annotated conditions and effects: the annotation of a condition states whether the associated proposition must be asserted at the *start* of the interval, the *end* of

```
(:durative-action board
   :parameters (?p - person
                ?a - airplane
                ?c - city)
   :condition (and (at start (at ?p ?c))
                   (at start (at ?a ?c))
                   (over all (at ?a ?c)))
   :effect (and (at start (not (at ?p ?c)))
                (at end (in ?p ?a))))
```

Figure 3. A PDDL2.1 `board` operator.

the interval or *over* the interval; the annotation of an effect asserts whether the proposition occurs immediately or at the end point of the interval. Figure 3 shows how a PDDL2.1 level 3 durative action models a person boarding an aeroplane. A number of researchers have recently developed extensions of GraphPlan or state search planners capable of reasoning with PDDL2.1 level 3 durative actions [4, 7].

Since the advent of GraphPlan and the many GraphPlan inspired successors, partial order planning has been neglected due to its comparatively poor performance. However, the extra burden of reasoning about time even making the simplistic black-box assumptions about durative actions (stated above) causes the performance of temporally extended GraphPlan based systems (such as [15]) or state search planners to deteriorate in comparison to their performance in non-temporal domains. The extra expressive power of PDDL2.1 with regard to modelling durative actions seems likely to lead to an even greater deterioration in performance if only because of the additional constraint reasoning that must be done to ensure temporal consistency [3].

Partial order planning [9, 12], on the other hand, is more suitable for modelling concurrency between actions with different durations, and, because it avoids full instantiation, may be more efficient when reasoning with planning domains specified using PDDL2.1. It was therefore decided to investigate extending the partial order planning algorithm to solve problems specified using the level 3 durative action specification of PDDL2.1 level 3 with a view to incorporating it in the continuous planning framework described in section 2. In the remainder of this paper, the algorithm is described in further detail.

6. Extending partial order planning

One way of planning with PDDL2.1 level 3 durative actions (see [6]) is to decompose them into their instantaneous start and end actions, taking care to maintain the relationship between start and end and also between those points and any invariant conditions specified by the durative action. The meaning of a durative action is obtained by the construction of two instantaneous *end-point* actions, with a standard STRIPS semantics (assuming there are no non-atomic conditions or effects), and a collection of instantaneous *monitoring* actions responsible for maintaining invariant conditions over the specified duration. This approach provides a simple basis for handling durative actons in a partial order framework.

If no invariant conditions (of the form `(over all p)`) are specified, a durative action DA can simply be transformed into two STRIPS actions, one for each of the endpoints, *start* and *finish* associated with DA. When invariants are specified it is necessary to ensure that the invariant remains true over the interval that occurs between the *start* and *end* point associated with DA. In a partial order planner this may be achieved by modelling invariant conditions as special causal links which may be protected using standard partial order planning threat resolution procedures. The only aspect of durative actions which is not captured in this transformation using standard partial order planning machinery is the duration associated with DA. This may be modelled by minor modifications to both the representation of temporal constraints as well as to the temporal constraint consistency checker. The extension that is required is to ensure that if an action a_{start} is temporally constrained to come after b_{start}, and a_{end} is constrained to come before b_{end}, then the duration of a is strictly less than the duration of b. In the partial order framework, checking that this requirement is satisfied will help to prune inconsistent alternatives early. Alternatively it might be left to the final partial plan linearisation process to ensure that sufficient time elapses between the start and end points of durative actions, but this is a less efficient solution because of the failure to identify some temporally inconsistent plans.

The process is as follows: When instantiated, a durative action DA is converted into two actions DA_{start} and DA_{end} which are added to the actions A belonging to the partial plan P.

1. The *name* field of DA_{start} (DA_{end}) is the *name* field of DA appended with the suffix *start* for DA_{start} (or *end* for DA_{end}).
2. The *parameters* field of DA_{start} (DA_{end}) contains the set of typed variables belonging within the precondition and effect propositions of DA_{start} (DA_{end}).
3. The *precondition* of DA_{start} (DA_{end}) is equal to the conjunction of the set of all propositions *p*, such that `(at start p)` `((at end p))` is a condition of DA.
4. The *effect* of DA_{start} (DA_{end}) is equal to the conjunction of the set of all simple effect propositions *e*, such that `(at start e)` `((at end e))` is an effect of DA.

Should the durative action DA contain invariant conditions of the form `(over all p)`, causal links are created of the form $DA_{start} \xrightarrow{p} DA_{end}$ and added to the set

```
(:action board-start
   :parameters (ernie plane ?c1 - city)
   :condition (and (at ernie ?c1)
                   (at plane ?c1))
   :effect (not (at ernie ?c1)))

(:action board-end
   :parameters (ernie plane)
   :condition ()
   :effect (in ernie plane))
```

Figure 4. The PDDL2.1 `board` operator is converted into two simple action instances, `board-start` and `board-end`.

of causal links L. The implications of modelling invariant conditions as causal links is discussed further in section 6.1 below.

In addition, the temporal constraint $DA_{start} \prec DA_{end}$ is added to the set of constraints O.

When the level 3 durative action template *board* described in figure 3 is selected to achieve the goal $(in\ ernie\ plane)$ it is instantiated and transformed into the actions $board_{start}$ and $board_{end}$ of figure 4.

The temporal constraint $board_{start} \prec board_{end}$ is added to the set of temporal constraints O.

The invariant condition *(over all (at plane ?c1))* is transformed into a causal link

$$board_{start} \xrightarrow{(at\ plane\ ?c1)} board_{end}$$

and added to the set of causal links L. The justification for this is discussed in the following section.

6.1. Modelling invariant conditions

The decision to model invariant conditions of the form `(over all p)` as causal links of the form $DA_{start} \xrightarrow{p} DA_{end}$ assumes (if the causal link is interpreted in the traditional partial order planning manner) that the action DA_{start} establishes the invariant condition p (i.e. DA_{start} contains p as an effect proposition) while the action DA_{end} consumes p (i.e. DA_{end} contains p as a condition proposition). In fact, there are three cases to consider:

- DA_{start} contains p as a precondition;
- DA_{start} contains p as an effect;
- DA_{start} does not contain p either as a precondition or as an effect.

In the first of these cases the causal link $DA_{start} \xrightarrow{p} DA_{end}$ is added, and the goal (p, DA_{start}) is added to the open conditions of the plan. This expresses the requirement that p be maintained over the whole interval of the action, but it is a non-standard use of causal links to use them to promise that a condition will be maintained before it has actually been achieved. On the other hand, the plan will be invalidated if p cannot be achieved for DA_{start}, and the sooner it is known that p must be maintained the less wasted search will be incurred.

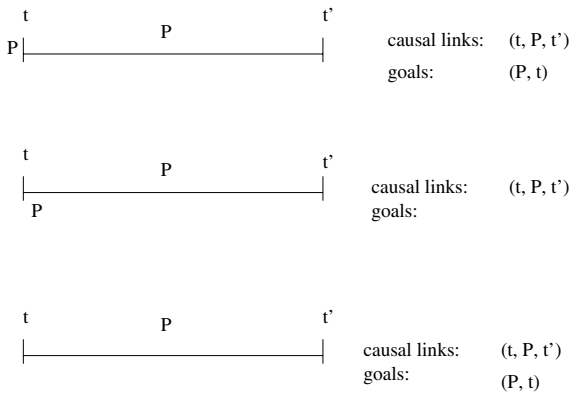

Figure 5. Three Invariant Situations

The second case is a simple one - the causal link $DA_{start} \xrightarrow{p} DA_{end}$ is added to the plan when DA_{start} and DA_{end} are added, because DA_{start} is itself the achiever of its own invariant condition.

The last case, in which p is an invariant but neither a precondition or a start effect, is slightly more subtle. In fact, because the temporal relations between time points in the plan are all strict precedence relations (partial order planners typically do not reason about synchronous activity) this case can be treated as equivalent to the first. That is, the causal link $DA_{start} \xrightarrow{p} DA_{end}$ is added to the plan, along with the goal (p, DA_{start}). The goal ensures that an achiever will be found for p, the causal link ensures that p will be preserved until the end point. The subtlety exists because, if we could exploit synchronicity it would be necessary to be precise about the exact point at which p needs to be asserted to satisfy the invariant. Because we cannot, we are forced to ensure that p is asserted strictly before the point at which it is required (which is immediately after application of the action).

Figure 5 describes the three cases and the causal links that must be added. The *board* operator, given in section 3, is an example of the first case (the plane must be at the city as a condition of board and throughout the duration of boarding). Further details are given in [3].

7. Conclusions and further work

In this paper a continuous planning framework was presented in which it is assumed that a situated planning agent is able to generate and prioritise goals taking into account its context and *motivations*. Goals have deadlines and ac-

tions have duration which means it may not be possible to achieve goals by their deadlines. In addition, planning and execution must be interleaved while the agent's motivations may be used as part of a heuristic to select the most promising plan for further refinement. The component responsible for generating plans (*Plan to achieve goal*) was based upon the partial order planning paradigm which was extended to reason about whether or not there is sufficient time available to achieve all goals, to estimate deadlines for actions, and to maintain a record of the dependencies between actions and goals to facilitate plan editing. The degree to which plans support the motivations of an agent, together with the number of goals of high priority offer an extra plan metric when selecting the best solution to a planning problem – the use of plan metrics other than the number of actions in a plan were discussed by the developers of PDDL2.1 [6].

Because the original implementation of this component made the classical black-box assumption concerning durative actions it was decided to investigate the feasibility of extending the partial order paradigm to reason about PDDL2.1 level 3 durative actions which give the domain modeller greater expressive power. The required extensions described in this paper are simple and elegant, but more work is needed to confirm whether or not this elegance is bought at the expense of an unmanageable search problem. A partial order planner capable of reasoning with PDDL2.1 durative actions is currently being implemented and future work will involve experimenting with search control in this system by investigating the heuristic techniques described in [10], symmetry-breaking techniques [5] and goal-ordering strategies [13].

Finally, once the partial order planning component *Plan to achieve goal* has been adapted to reason with PDDL2.1 durative actions, minor modifications will have to be made to the component responsible for assigning deadlines to actions *Estimate deadlines* to cope with the fact that durative actions are now represented using two consecutive start and end point instantaneous STRIPS actions. Such extensions will enable the continuous planning framework described in this paper to reason about goals with deadlines and durative actions with greater expressive power.

References

[1] A. Blum and M. Furst. Fast planning through plan-graph analysis. In *Proceedings of IJCAI-95*, 1995.

[2] A. M. Coddington. *Self-motivated Planning in Autonomous Agents*. PhD thesis, University of London, 2001.

[3] A. M. Coddington, M. Fox, and D. Long. Handling durative actions in classical planning frameworks. In *Proceedings of the 20th Workshop of the UK Planning and Scheduling SIG*, pages 44–58, 2001. ISSN 1368-5708.

[4] M. B. Do and S. Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of the European Conference on Planning*, 2001.

[5] M. Fox and D. Long. The detection and exploitation of symmetry in planning domains. In *Procs. of the 15th Int. Joint Conf. on Artificial Intelligence*, 1999.

[6] M. Fox and D. Long. PDDL2.1 : An extension to PDDL for expressing temporal planning domains. Technical report, Department of Computer Science, University of Durham, Durham, DH1 3LE, UK, 2001.

[7] A. Garrido, M. Fox, and D. Long. A temporal planning system to manage level 3 durative actions of PDDL2.1. In *Proceedings of the 20th Workshop of the UK Planning and Scheduling SIG*, pages 127–138, 2001. ISSN 1368-5708.

[8] C. A. Knoblock. Planning, execution, sensing and replanning for information gathering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[9] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on AI*, pages 634–639, 1991.

[10] X. Nguyen and S. Kambhampati. Reviving partial order planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2001)*, 2001.

[11] T. J. F. Norman. *Motivation-based Direction of Planning Attention in Agents with Goal Autonomy*. PhD thesis, University of London, 1997.

[12] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for adl. In *Procs. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1992.

[13] J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering and usage of landmarks in planning. In *Proceedings of the European Conference on Planning*, 2001.

[14] D. E. Smith, J. Frank, and A. K. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[15] D. E. Smith and D. S. Weld. Temporal planning with mutual exclusion reasoning. In *Proceedings of IJCAI-99, Stockholm*, pages 325–337, 1999.

[16] S. A. Vere. Planning in time: Windows and durations for activities and goals. *Pattern Analysis and Machine Intelligence*, 5:246–267, 1983.