Scheduling Actions with State-Dependent Resource Requirements

Aga Skotowski and Ella Atkins

Space Systems Laboratory
University of Maryland
College Park MD 20742
aga@cs.umd.edu ella@ssl.umd.edu

Abstract

In planning and scheduling domains such as space-based data collection, task execution cost is highly dependent on system state. Transitioning between states may also be costly. In this paper we consider the problem of scheduling tasks in the case where the cost of executing a task depends upon the state in which it is executed. To separate out the state-dependent aspects of the question, we examine several simplified versions of the problem in which we consider the interactions of task execution cost and state transition cost without the complications introduced by task dependencies and preconditions. We then describe a solution by exhaustive search. Finally, we outline a promising line of research involving a statistical analysis of the manner in which task execution cost depends on state.

Introduction

Planning and scheduling systems require spatio-temporal reasoning when the choice of actions and execution times depends on object or event location and time. Qualitative reasoning both enhances understandability and combats complexity, but it may also limit an algorithm's capabilities in domains where the relation between an action's utility and location or time is a complex function. Alternatively, algebraic approaches can accurately model domain dynamics as a function of continuous-valued state ($\boldsymbol{x}$, t), but also require a mathematical specification of the remaining domain features and of the goals as cost minimization / utility maximization procedures.

Consider the problem of scheduling tasks for a space-based observatory such as the Hubble Space Telescope. Domain experts work together to select tasks that maximize scientific data collection. A scheduling algorithm then assigns tasks to computing and sensing resources based on priority and worst-case timing requirements. The scheduler must also satisfy constraints on observation windows (i.e., spatial locations from which targets are visible) when assigning tasks to time slots (Miller 1995). However, to date the problem of science planning and scheduling has been largely decoupled from the planning of motion. Spacecraft are either inserted into fixed drift orbits or have impulse (orbit change) maneuvers planned manually from the ground. Orbital motion is then specified for the science planner, which cannot optimize translational motion and generally does not optimize rotational motion as it assigns tasks to time slots.

At a launch cost of $10,000 per pound, fuel is a premium non-renewable resource required for spacecraft control. As NASA moves toward formation flight missions, both position and attitude maneuver optimization are crucial, since continuous low-thrust control is needed for the high-precision "virtual" formation shapes required for accurate science data collection (Millam 2001). Formation geometry and pointing constraints are a function of the task to be performed (e.g., interferometry). In an optimal plan, the formation will slowly evolve over a long sequence of observation targets rather than requiring large-scale, costly maneuvers between tasks.

The above example generalizes to the important but little-studied problem of planning when the cost of tasks or actions varies depending on the state in which those actions are executed. As a prelude to the full planning problem, we have been working to describe and solve the simpler *scheduling* problem for the case where task resource requirements are state-dependent. The spacecraft is our motivating example. Working from a given set of tasks to accomplish, the scheduler takes into account the fact that each task may require more or less fuel depending on the state in which it is executed, and that transitioning between states may also be costly in terms of time and fuel. We use fuel as a generic cost unit due to its importance in the spacecraft domain. The problem can be posed with any number of different resources, provided the scheduler also has functions to evaluate the utility of units of one resource compared to units of another. The challenge for the scheduler, then, is to weigh the cost of changing state (say, by moving or reorienting the spacecraft) against the potential savings in task execution cost.

The purpose of most spacecraft is to perform work on or take sensor data from the external world, and we assume that this is the goal of the set of tasks the scheduler is attempting to accomplish. However, in considering this simplified version of the problem, we assume that a prepared task list, previously generated by mission controllers or a separate planner, is made available to the scheduler. Because this task list has already taken into account conditions in the external world, detailed knowledge of the external features of the world is in a sense irrelevant to the scheduling algorithm.

Instead, the state space that the scheduler explores during its search comprises the possible internal states of the spacecraft. These internal states may include features such as attitude, position, remaining fuel, manipulator pose, payload capacity, or operational status. We search over these states because it is precisely the internal state features that can be expected to cause task cost to vary as a function of state (for example, by requiring a spacecraft to reorient before collecting data from a given target).

In the rest of this paper, we briefly survey the various ways in which the scheduling problem can quickly grow to be exponentially complex. To begin addressing scheduling with state-dependent task costs, we describe a simplified set of problems in which task execution costs are dependent on state, but tasks have no dependencies or preconditions to fulfill. These problems can have state transition costs that are either constant or variable. We discuss the effect of each on scheduling complexity. We then outline an exhaustive search algorithm for the constant transition cost case. Finally, we discuss our future research directions and present our conclusions.

Background

A large amount of research has been done on different variations of the scheduling problem. Simple models begin with non-preemptable periodic tasks, which require only time on one processor, and which are scheduled offline by algorithms such as rate monotonic scheduling (Krishna and Shin 1996). More realistic, and consequently more complicated, are scenarios that involve any combination of the following:

- Tasks which require several resources each, or a situation where there are multiple instances of resources, such as two or more cameras.
- Online scheduling, the only choice when the required tasks will not be known in advance.
- Preemption, which allows tasks to interrupt each other and can help equalize access to some resources as well as allow flexible response during online scheduling.
- Tasks with precedence constraints, which require certain other tasks to complete before they can be released.
- Tasks with varying priorities that depend on state, or tasks whose resource requirements change with state.

Especially relevant to observation scheduling and mission planning are real-time scheduling algorithms, a broad category which ranges from anytime planning to design-to-time planning. Anytime strategies iteratively improve plan quality as time permits, allowing human users to request the current best solution at any point in the search process (Dean et al. 1995). Design-to-time strategies adjust planner complexity based on available planning time (Garvey and Lesser 1993). A number of real-time researchers have also developed efficient quality-of-service dynamic scheduling algorithms to adjust resource utilization in real-time. Some systems interleave planning with scheduling, allowing the scheduler to guide planner backtracking by describing which tasks are causing resource bottlenecks (Atkins et al. 2001). Although it is difficult for real-time systems to provide guarantees on both quality and timeliness, they offer important advantages in handling the complexity of tasks such as mission planning (Musliner, Durfee, and Shin 1995).

Many of these problems are NP-complete, and it is therefore worthwhile to look for algorithms which produce schedules that are "good enough" rather than guaranteed optimal. The specific scheduling problem that we are interested in, that of scheduling tasks whose cost depends on state, has not yet been addressed by standard AI or real-time architectures. However, we hope to extend a generic problem solver to include such temporal reasoning in its search, as we describe later in the Future Work section.

Problem Description

In this paper we will consider the scheduling of very simplified tasks which have only one resource requirement: some amount of fuel, which may vary depending on the state the system is in when the task is executed. There are no dependencies or possible conflicts between the tasks, and the tasks neither require the satisfaction of any preconditions in order to run, nor change the internal state of the system when they are executed. They may, of course, change the external state of the world, with which our scheduling algorithm is not concerned. We assume the system can transition from state to state at will, though at some cost in fuel, and can execute any number of tasks in each state.

Let each task k have some associated function $f_k(s)$ which determines how much fuel that task will require as a function of the state in which it occurs. We will call this the fuel function. If there are N possible states, the functions $f_k(s)$ are defined over the domain of all N states. The actual values of the fuel function may be either given explicitly or derived by the planner as needed by computing some arbitrary function on one or more aspects of a given state.

We assume that the cost of transitioning from one state to another is fully specified in an $N \times N$ matrix. It could also be calculated dynamically as needed, especially for very large state spaces. In general, transitioning to a different internal state may require some combination of several resources, but for uniformity we will consider the transition cost, like the task execution cost, to be measured in units of fuel usage. If no transition exists between two states, that transition cost will be infinite.

The total cost of any schedule will consist of the costs of transitioning from each state to its successor (transition costs) added to the cost of executing each task in the state for which it is scheduled (execution costs). The problem is to select a series of states that will minimize this total cost.

Problem Subcases

Case 1: Constant Fuel Functions

The simplest case, in which the values of all fuel functions are constant, reduces to the standard planning problem in which task cost is independent of state. This problem is ably handled by a variety of planners, which will only need to take into account state transition costs and whatever other criteria might be applicable to the tasks.

Case 2: State-Dependent Fuel Functions, Constant Transition Cost

More interesting is the case in which task costs depend on the execution state, but the cost of transitioning between any two states is always the same: the $N \times N$ transition cost matrix is filled with some constant $c > 0$. Intuitively, this appears to be an easier problem to solve than the fully general case of any $N \times N$ transition cost matrix. If the transition cost is a constant, then paths through the state space have the property of monotonicity: we are guaranteed that a direct transition from state A to B will be the least cost path from A to B. There is no need to allow for the possibility that we may in fact achieve a lower transition cost by going from A to some intermediate state I and then to B.

The total transition costs will be lowest when the number of different states entered is minimized, but this reduction in cost may be offset by the possible increase in total task execution cost if some tasks would have used less fuel in another state. The magnitude of the constant c compared to the possible ranges of the fuel functions dictates how much emphasis should be placed on minimizing the total number of states entered.

We approach this problem by considering an algorithm in which each task selects one or two states as "preferred" states—that is, states in which this task would require a minimal amount of fuel. The extreme case of this would be each task k choosing the one state that is the minimum of its fuel function. The scheduler then attempts to schedule each task to execute in one of its preferred states.

But perhaps executing each task in one of its preferred states is not possible. There may be conflicts between the tasks, or restrictions on how many tasks can be executed in each state. In that case, the scheduler requests that some or all of the tasks enlarge their set of preferred states with several "next-best" states for each task. The scheduler proceeds to attempt scheduling again, using this larger and presumably more flexible set of preferred states. In other words, this algorithm runs a series of scheduling attempts. Whenever scheduling with a given set of preferred states proves infeasible, the algorithm attempts to "back off" to a less stringent set of constraints (in this case, more possible states for each task). These relaxed requirements increase the cost of the plan, since they may introduce more state transitions as well as allowing tasks to be scheduled in less-optimal states. However, the hope is that by intelligently choosing which requirements to relax at each iteration, the final plan will be a good approximation to a lowest-cost plan.

There are several different ways to relax the requirements while attempting to keep the total plan cost to a minimum. As a first approximation, let every task k in iteration i have a set of preferred states $P_{k,i}$. After an unsuccessful attempt to assign each task k to one of the states in $P_{k,i}$, the scheduler asks each task to generate $P_{k,i+1}$ by adding one or more states to $P_{k,i}$. The scheduler can now try scheduling the tasks with these larger sets of preferred states.

A better method is to ask only *some* of the tasks to enlarge their list of preferred states. For example, if only two or three tasks are involved in conflicts, it is reasonable to try rescheduling with an expanded $P_{k,i}$ for just those tasks before resorting to expanding the preferred state set of every task.

Another helpful factor to take into account might be the maximum fuel each task would require if it were scheduled to run in one of its currently preferred states—that is, the highest cost each task could contribute to the total plan cost at this iteration. For each task k in iteration i, let this cost be

$$C_{k,i} = \max\{\ f_k(s) : s \quad P_{k,i}\ \}$$

A task k which has one of the larger costs C_k at this iteration should *not* be asked to expand its list of preferred states. By definition, the states already on k's preferred list are its lowest cost states, so any states which k might add would only be adding more high-cost options for the scheduler. On the other hand, tasks that have a low C are relatively low-cost at this iteration. They might have still other states in which they would likewise be fairly inexpensive, making them good candidate tasks to ask for expanded preferred state sets.

To formalize this method, we consider

$$C_{med} = \text{median}\{\ C_{k,i}\ \}$$

C_{med}, the median worst-case cost for all tasks, offers a baseline for determining what is a low cost and what is a relatively high one for the current iteration. We modify each preferred state set $P_{k,i}$ as follows:

$$P_{k,i+1} = P_{k,i} \quad \{\ s : f_k(s) < C_{med}\ \}$$

Since half of the task costs $C_{k,i}$ at this iteration are already greater than C_{med}, we can consider fuel costs lower than C_{med} to be relatively inexpensive, and we expand each $P_{k,i}$ by all states that would generate a task execution cost below this acceptable limit.

To complete our discussion of this subcase, we wish to consider what value of c will be so high as to effectively make any transition impractical, effectively forcing all tasks to execute in the initial state regardless of their fuel function. Let T be the task set, $|T|$ the number of tasks, and F_{max} the maximum fuel usage value any fuel function takes on. If the highest fuel usage of any task in any state is F_{max}, then F_{max} is also an upper bound on the savings in fuel achievable by adding one more state and executing some of the tasks in the new state. If *all* of the tasks run more efficiently in the second state, the upper bound on the cost savings is $|T| * F_{max}$. (Of course, in this case, the first state could simply be eliminated entirely.) Thus, when $c > |T| * F_{max}$, it is clear

that any transition would be so expensive that no possible savings in task execution cost could balance it.

If we have more information about each fuel function, we can achieve a tighter upper bound on the possible savings that could result from adding another state. Assume, for example, that we know both the maximum and minimum value of every fuel function. Then for any given task, there can be no savings greater than the difference between the maximum and minimum value of that task's fuel function. The condition under which it is not worthwhile to transition at all is now

$$c > \sum_{k \in T} (\max(f_k) - \min(f_k))$$

where f_k is the fuel function for task k.

At the other extreme, $c = 0$ is the case where transitioning from state to state is free. In this situation, a very simple algorithm suffices. Let the ideal state for each task be the state at which that task's fuel function has a minimum. Find the ideal state s for each task, assign that task to s, and add s to the set S of scheduled states if it is not already there. Then, for every state s in S, transition to s and execute all tasks assigned to it. For a c that is low compared to the ranges of the fuel functions, transitioning is effectively free and a similar algorithm is be worth considering.

In general, neither of these cases will apply and c will be some value that will tend to push the search towards minimizing the number of state transitions in the schedule. Given that some task set T is currently scheduled to execute in state A, what are the conditions under which it would be advantageous to transition to state B and execute a subset of the tasks ($T_{moved} \subseteq T$) there?

The cost of the transition is c. The savings gained from executing some of the tasks in state B instead of in state A is

$$C_{saved} = \sum_{k \in T_{moved}} (f_k(A) - f_k(B))$$

Presumably tasks only move to state B if they are strictly less costly in state B than in the original state A. If all tasks in A used the same amount of fuel as they would in state B, there is no reason to incur a transition cost at all. If $C_{saved} > c$, including this transition will lower the total schedule cost. If $C_{saved} \leq c$, the transition is not cost effective and should not be included. In the general case, the shortest path between two states is not necessarily the least expensive, and it would still be necessary to explore this transition because of the possibility it is part of a significantly less expensive path to some other necessary state. However, here we are considering only the subcase in which all transition costs are the same, and transitioning from A to B to C will never be less expensive than transitioning from A directly to C. Since any savings in the total task execution cost are outweighed by the cost of the transition, and since there is no possibility of this transition eventually helping to lower the total transition cost, this transition can be omitted from further consideration.

Case 3: General Transition Costs

In the general case, the cost of transitioning between two states will vary depending on the states instead of being a constant as we have been assuming. This complicates the search problem. A transition to a given state may now be worth including in the schedule not only because some tasks can run with lower cost in it, but also because it may be part of a lowest-cost path to some other state in which tasks can execute with significantly lower cost than they would require otherwise.

A possible approach to this problem is to begin by finding the lowest-cost Hamiltonian path through the state space, and then to schedule each task in its preferred state. This first stage would be followed by a repair process which takes into account the fact that, depending on specific transition costs and fuel function values, it may be more efficient to remove one or more states from the schedule if it means reassigning some tasks to less-preferred states.

Solution By Exhaustive Search

To give the reader an overview of the search process, we will describe the execution of an exhaustive search that we have implemented to solve the simplified problem example (no interactions between tasks) under the assumption that all transition costs have the constant value c. Abstractly, we wish to examine every possible combination of states and, for each combination, every possible assignment of tasks. Since the transition costs are all c, the order in which we visit the states is unimportant, so we fix them in an arbitrary sequence. Note that a schedule in which only state A appears will have the same total cost as a schedule in which both A and B appear but no tasks are executed in state B, minus one transition cost of c. This observation allows us to forgo calculating various combinations of internal states and instead work with all N states at once, with the convention that, in calculating the cost, we subtract the cost of transitioning to any states where no tasks are scheduled to execute.

State *A*	State *B*	State *C*	State *D*	State *E*
k1		*k2, k3, k4*	*k5*	

Table 1: A sample search state

Table 1 shows one search state which represents a partial schedule for a problem with five system states (*A*-*E*) and seven tasks (*k1*-*k7*, with *k6* and *k7* not yet assigned to a state). It is interpreted as, "Start in state *A* and execute *k1*, then transition to state *C* and execute *k2*, *k3*, and *k4*, then transition to state *D* and execute *k5*." States *B* and *E*, with no tasks assigned to them, are implicitly not entered in this schedule. The total cost of this schedule is

$$c\ (N_{nonempty} - 1) + \sum_{s} \sum_{k \text{ in } s} f_k(s)$$

The children of this search state are generated by the five possible locations where *t6* could be added:

Child	State *A*	State *B*	State *C*	State *D*	State *E*
1	*k1, k6*		*k2, k3, k4*	*k5*	
2	*k1*	*k6*	*k2, k3, k4*	*k5*	
3	*k1*		*k2, k3, k4, k6*	*k5*	
4	*k1*		*k2, k3, k4*	*k5, k6*	
5	*k1*		*k2, k3, k4*	*k5*	*k6*

Table 2: Child states of the search state in Table 1

The root of the search tree is the empty schedule with no tasks assigned to any of the states, and the first children generated would be the five possible assignments for *k1*. At the leaves of the tree are all possible assignments of tasks to states. This tree has a depth of $|T|$, the total number of tasks, since each state is expanded by scheduling a previously unassigned task. The branching factor is N, the number of possible states. The total number of nodes in the tree is therefore

$$\sum_{i=0}^{|T|} N^i = \frac{N^{|T|+1} - 1}{N - 1}$$

Clearly, this solution method is not feasible for realistic domains. However, its general structure provides a framework for understanding more sophisticated search algorithms, which will use heuristics to guide the search down the most promising branches of the state space tree rather than exploring it exhaustively.

Future Work

In the near future we intend to implement reasoning about state-dependent action costs as an extension to SHOP2, a planner written in Lisp that follows a total-order, forward-chaining strategy (Nau et al. 2001). Because of this strategy, SHOP2 always knows the current world state, which considerably simplifies planning with state-dependent action costs.

We are interested in exploring ways of including these state-dependent costs as part of the planner's heuristics for guiding search. In particular, one idea that may hold promise for guiding the planner is the realization that the expected variance and extremes of each fuel function may suggest which task assignments it is safe to commit to early in the search process. Realistically, it is likely that some tasks will have a flat or nearly flat fuel function: it makes little difference what state they execute in. These tasks allow the scheduler the most flexibility, since they can be moved from state to state to resolve scheduling conflicts without significantly affecting plan cost. On the other hand, some tasks may have a sharp dip somewhere in their fuel function, such that one state is highly preferred and scheduling the task in any other state will cause a large increase in total plan cost. If scheduling conflicts occur, it may be worthwhile to shuffle some other tasks to less preferred states in order to accommodate these tasks.

It would be very useful to be able to automatically categorize tasks according to how strongly their fuel functions favor a small number of particular states.

- *State sensitive* tasks have a small standard deviation with a few large minimal outliers. These tasks will cause a significant rise in total plan cost if they are not scheduled to execute in their preferred states.
- *State invariant* tasks have a fuel function characterized by a small standard deviation and no outliers. The choice of state for these tasks does not substantially affect the total task execution cost.
- *Random* tasks have a large standard deviation. Little can be determined about what states they should be executed in.

Since these calculations are already doing work linear in the number of states for each fuel function, they should also take the opportunity to annotate any tasks with especially *high* outliers—states that are particularly worth avoiding for that task. This statistical precomputation could be done once offline for any number of runs of the scheduling algorithm afterwards.

Clearly, it benefits the scheduler to schedule the state sensitive tasks first. As much as possible, these tasks should stay in their preferred states during any future backtracking. Since the number of preferred states for each state sensitive task is assumed to be very small, scheduling these tasks first should significantly pare down the search space. It is, of course, still possible that two state sensitive tasks will somehow conflict with each other. Even in this case, however, it should be a simple matter to evaluate which of the tasks will have the least impact on the total plan cost if it has to be moved to a non-preferred state. Furthermore, by definition a state sensitive task has a small standard deviation. If it is not scheduled in one of its preferred states, it makes little difference which of the non-preferred states it is assigned to. In fact, at that point it has essentially become a state invariant task. Therefore, the conflicting state sensitive task that could not be accommodated can be removed from the schedule entirely for rescheduling at a later time with the state invariant tasks.

After the state sensitive tasks are scheduled, the scheduler works on finding effective assignments for the random tasks. Most of the state space search takes place here, possibly using the algorithm detailed previously which repeatedly increases each random task's set of preferred states until they can all be scheduled. Alternately, another, more sophisticated algorithm might be used instead. Either way, the hope is that scheduling the state sensitive tasks early has pruned the search space enough to substantially reduce the heavy state space search work in this phase.

In the final stage, the state invariant tasks are scheduled. Since these tasks cost about the same in any state, the scheduler need not do extensive search. In the interests of efficiency it can simply fill in the loosely bound tasks wherever there is space for them.

This algorithm could be improved by not only assigning each task a category, but also ranking the tasks within each category. Thus, a tightly bound task with only one preferred state would be scheduled before a tightly bound task with three preferred states. The random tasks could be ranked by their standard deviation. Even more worthwhile would be to prioritize scheduling the tasks that were both very variable and, on the average, quite expensive. This may be useful if, for example, there is some preset time limit on the run of the scheduling algorithm, or if the scheduler is meant to be an anytime algorithm where the user can request a "best current guess" at any point in the search process. In this case, it would be necessary for the scheduler to prioritize its efforts by working first on scheduling the tasks whose exact placement is most likely to make a large difference in the final plan cost. In case of a time limit cutoff, some tasks may have to be scheduled much less carefully than would be ideal. It may make a great difference to the quality of the solution if the scheduler can arrange that these shortchanged tasks are only ones whose exact scheduling has little affect on overall schedule cost anyway.

We have been assuming that it is a simple matter to analyze the complete range of values that each fuel function takes on. This need not be the case. It is possible that the fuel function is an arbitrary function of selected features of a state, and it may be very expensive indeed to generate all possible world states in order to find the value of the fuel function at each of them. However, even if this is so, the expense may still be acceptable as a one-time preprocessing cost. Failing that, it may be feasible to categorize each task by random sampling of the fuel function values. Further analysis of the fuel functions of real-world tasks will help clarify this issue.

Conclusion

We have begun a study of scheduling with state-dependent resource requirements, motivated by the increasing need to conserve fuel resources and follow smooth motion paths in space-based observation craft. We have surveyed the various facets of the scheduling problem, many of which could be incorporated into our research in the future to create a powerful planner capable of reasoning about state-dependent action costs as one aspect of complex, many-attribute tasks. In order to open discussion of this little-studied topic, we have introduced concepts and notation that can be used to discuss the simplified problem in which the only concern is handling task costs that are dependent on the internal state of the data gathering system, without the added complications of task interactions or external world state.

We have discussed algorithms to solve this simplified problem for the case where transition costs are constant as well as for the general case of arbitrary transition cost between any two states. We have described a solution by exhaustive search of all possible schedules as a framework on which more refined heuristic strategies can be based. Finally, we have presented the beginnings of work on a promising algorithm which may allow considerable pruning of the search space and a significant reduction in backtracking by carefully choosing which tasks to schedule first. With further work we hope to extend the conceptual understanding of planning with state-based resource costs as well as to integrate this knowledge into powerful and sophisticated algorithms that can offer previously unachieved effectiveness and flexibility in mission planning.

References

Atkins, E., Abdelzaher, T., Shin, K., and Durfee, E. 2001. Planning and Resource Allocation for Hard Real-time, Fault-Tolerant Plan Execution. *Journal of Autonomous Agents and Multi-Agent Systems* 4(1-2):57-78.

Dean, T., Kaelbling, L.P., Kirman, J., and Nicholson, A. 1995. Planning under Time Constraints in Stochastic Domains. *Artificial Intelligence* 76(1-2):5-74.

Garvey, A.J., and Lesser, V.R. Design-to-time real-time scheduling. 1993. *IEEE Transactions on Systems, Man and Cybernetics* 23(6):1491-1502.

Krishna, C.M., and Shin, K.G. 1996. *Real-Time Systems.* McGraw-Hill Series in Computer Science. McGraw-Hill.

Milam, M.B., Petit, N., and Murray, R.M. 2001. Constrained Trajectory Generation for Microsatellite Formation Flying. In Proceedings of the 2001 AIAA Guidance, Navigation and Control Conference.

Miller, G. 1995. Planning and Scheduling for the Hubble Space Telescope. *Robotic Telescopes.* ASP Conference Series 79. Henry, G.W. and Eaton, J.A. eds. San Francisco: ASP. 173-183.

Musliner, D.J., Durfee, E.H., Shin, K.G. 1995. World Modeling for the Dynamic Construction of Real-Time Control Plans. *Artificial Intelligence* 74(1):83-127.

Nau, D., Cao, Y., Lotem, A., and Munoz-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-99).

Nau, D., Cao, Y., Lotem, A., Munoz-Avila, H., and Mitchell, S. 2001. Total-Order Planning with Partially Ordered Subtasks. In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001).