SRC-TR-87-48

Towards Dynamic Planning

by

James C. Sanborn
James A.Hendler

Towards Dynamic Planning

James C. Sanborn
James A. Hendler

Computer Science Department & Systems Research Center
University of Maryland
College Park, MD 20742
(301)454-4148

Arpanet: sanborn@mimsy.umd.edu

Abstract

Many desirable application domains for planning are inherently *dynamic*: representation of the domain is ill-specified or incomplete, the existence or verity of information changes with time, and situations arise during plan execution that were not taken into account at plan generation time. Unfortunately, the traditional approach to planning and problem solving has concentrated most heavily on generating plans, with little attention given to problems associated with plan execution. As a result, these planners have not been generally successful when applied to many physical domains.

This paper introduces new directions in planning and acting for use in underconstrained and incomplete domains. In considering dynamic properties of several domains of interest, we define the properties of *dynamic domains.* The applicability of existing plan generation and execution techniques to dynamic domains is explored, illustrating the limitations inherent in assumptions these planners place on their application domains. In investigating extensions to the nonlinear planning paradigm, we show that nonlinear *plan interpreters* are better suited to less-static domains than are *plan compilers.* Further, we show that the plan interpreter approach is itself insufficient for dynamic domains. In examining recent approaches to handling dynamics, we present criteria necessary for a *dynamic planning* system based on integrating the notions of planning and acting. This approach marks a departure from traditional nonlinear planning though an ability to plan and act simultaneously, predict plan failure, and make short-term planning decisions based on information feedback during plan execution.

1. Introduction

Many desirable application domains for planning are dynamic in nature: representation of the domain is ill-specified or incomplete, the existence or verity of information changes with time, and situations arise during plan execution that were not taken into account at plan generation time. Unfortunately, the traditional approach to planning and problem solving has concentrated most heavily on generating plans, with little attention given to problems associated with plan execution. As a result, these planners have not been generally successful when applied to many physical (or "real world") domains. By a *dynamic domain*, we intend one which is

(1) changing continuously over time,
(2) incompletely specifiable at any point in time, and
(3) changing due in part to the actions of other agents.

As such, the blocks world is an inherently ***static*** domain: the locations of all objects are always known, only the planning agent moves blocks, and the planner always knows where it puts blocks. By a ***nonlinear planner***, we intend any problem solver that constructs a partial order on sequences of actions to be performed in order to attain a goal state from an initial state. Planners such as NOAH [Sacerdoti77], NONLIN [Tate77], NASL [McDermott77], DEVISER [Vere83], and TWEAK [Chapman85] fall into this catagory.[1]

This report discusses the applicability of nonlinear planning to dynamic domains. We consider the properties of two classes of nonlinear planners (section 2), restrictions placed on nonlinear planning application domains (section 3), a dynamic domain (section 4.1), extensions to nonlinear planning intended to incorporate dynamic domains (e.g., DEVISER and FORBIN [Miller85], section 4.2), and desiderata for a dynamic planner (section 4.3).

2. Plan Compiling and Interpreting

Following the terminology of [McDermott77], we distinguish two classes among the nonlinear planners. A planner simulating the effects of plan steps during plan generation, initiating plan execution only after a complete plan has been generated, is known as a *plan compiler* (denoted PC): NOAH, NONLIN, DEVISER, and TWEAK are PCs. A planner that plans each subtask explicitly, executing plan steps in the process, is termed a *plan interpreter* (denoted PI): NASL is a PI. To see the reason for this distinction, note that the PCs all generate plans from an *initial situation*: a snapshot of the world just before planning begins. PIs on the other hand, plan with an initial situation for each subtask as it is planned. Thus, PIs may examine the state of the world often during planning, while PCs work strictly from an initial situation obtained before any planning is done. Figure 1 shows the basic operation of PIs and PCs.

As a PC generates a plan, some portion of the current plan may lead to the failure of a condition the planner needs to preserve in order to satisfy a goal; this is referred to as a *protection violation* . In such a case, the planner is forced to backtrack to some previous node in its plan expansion and explore other alternatives. The portion of the previous plan backtracked over does not appear in the resulting plan, so the output plan is "optimal" in the sense that no steps are included which are later undone.

This optimality is one reason PCs have been preferred over PIs. Since A PI merely looks for a means of achieving a goal, executes it, and checks to see if the goal has been established. If backtracking is involved, the world may have been manipulated such that several steps must be "undone" (through additional planning) before other alternatives may be explored. In this way, a PI performs *visible backtracking* when it has chosen the wrong means of achieving a goal. In static domains, this behavior gives the appearance of poor planning. However, note that a PI has the property that, upon successful termination, the physical domain is in a goal state, whereas a PC remains in the initial state until execution begins. So long as the physical domain is static,

[1] Chapman [Chapman85] has shown that nonlinear planners as a class share most significant properties; his TWEAK program illustrates this point. Thus, we have limited our discussion to some of the more well known examples.

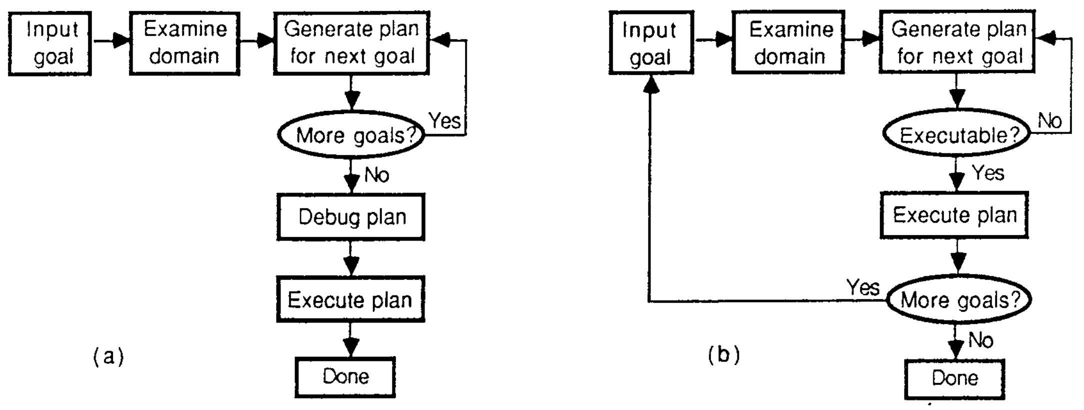

Figure 1 - *Plan compiling (a) and interpreting (b)*

the PC's plan will be just as viable as the PI's. However, as we shall see, the same behavior that makes PIs look bad in static domains gives them an advantage over PCs in non-static worlds.

Another reason for the concentration of attention on PCs in the past is that PIs have not traditionally incorporated *conjunctive goal* planning. In achieving a conjunct of goals, a PI plans to achieve each goal, the final situation is not checked to ensure that the conjunct holds. Adding this capability to a PI is a natural, relatively straightforward extension; see for example the SCRAPS planner [Hendler86].

3. The Static World Assumption

Present nonlinear planners assume that they operate on a static world satisfying various assumptions. We will refer to these assumptions collectively as the *static world assumption* (denoted SWA):

(1) the planner has complete knowledge of the world relevant to its task,

(2) change in the world is brought about only by the planner's executing primitive plan steps, this change may be modelled discretely, and the planner is completely aware of all effects of its actions.

(3) the planner acts alone in the world; there are no outside forces.

Consider the effects of the SWA on the properties of dynamic domains. Since the planner is considered to be (in some sense) omniscient with respect to the information relevant to its task, the only incompleteness permitted under the SWA is that which does not affect the planner's successful operation. In path planning for example, a planner needs to know the locations and dimensions of all objects it must avoid; it does not need to know the objects' identities, purposes, colors, etc. In dynamic situations, there is in general no way to capture all of the aspects of a domain that may be (or may have been) relevant to the planner, and no way to enumerate all the conditions under which an action will have a certain effect. Property (1) for dynamic domains violates SWA assumption (2), and property (3) violates assumption (3). While a nonlinear planner may manipulate several devices, there is a single *reasoner* controlling these devices in concert;

none acts autonomously.

Thus we see that planning under the SWA is restricted to inherently static domains. Due to the SWA, nonlinear planners have modelled action as a discrete operation on a static domain; planning is based on mapping discrete change onto a model of the world. This approach models action as **(before,after)** state change pairs; aspects of performing the action are not considered. Using this representation, the operations of planning and acting have become disjoint. For PCs, the process is one of generating a complete plan to achieve some (arbitrarily complex) goal, and only then executing the plan to effect changes in both the world model and the physical domain. By executing primitive plan steps during plan generation, PIs ***interleave*** planning with execution, but they rely on a similar model of action and discrete state change. The only difference here is that by explicitly planning to achieve its subgoals, a PI is able to update its model of the world during plan generation, keeping it closer to the state of the physical world as it plans. PCs are completely restricted by the SWA; no aspect of dynamics can be incorporated into a planner that does not at least periodically update its world model. However, as the following example shows, PIs are able to handle some aspects of dynamicity.

3.1. Example 1: PCs vs. PIs

Suppose we relax the SWA by removing the restriction on multiple independent actors. We will allow these new actors to change the physical domain at any time, so long as their actions are carried out "relatively instantaneously" with respect to planning (and acting) speed. Figure 2 shows an initial situation in the blocks world with the planning goal to achieve (on A C).

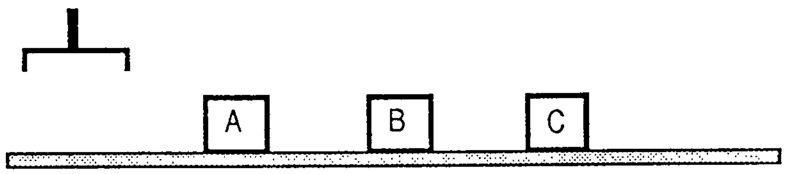

Figure 2 - *A Sample Blocks World*

From the initial situation and goal of Figure 2, a PC such as NOAH or TWEAK generates a plan of the form

1. (move-robot ROBOT-1 (location-of A))
2. (grasp-object ROBOT-1 A)
3. (move-robot ROBOT-1 (location-of C))
4. (release-object ROBOT-1 A).

Now suppose that, after the initial situation has been given to the planner, another actor puts block B on block C. Since the PC has no way of finding this out, the state of the *physical* domain when ROBOT-1 is finished will be as in Figure 3, whereas according to the planner, the goal will have been achieved. It is the PC's dependency on an initial "snapshot" of the world that leads to this discrepency.

Now consider a PI's attempt to achieve the same goal. We assume that the actor puts B on C sometime after the initial situation and before the the PI reaches the planning of (clear C). Thus, when (clear C) is planned, the PI finds that (on B C) is now true, causing it to plan (unstack ROBOT-1 B). This re-establishes (clear C), and the PI successfully achieves the goal state. It is the planning of each subtask, leading to an examination of the current state of the world at that point in the planning process, that leads to the PI's success in this case.

The advantage is clear: in non-static domains, re-examining the state of the world during plan generation allows greater flexibility than planning strictly from an initial situation. Unfortunately, this isn't enough; the PI isn't *that* much better than the PC. If our actor were to have

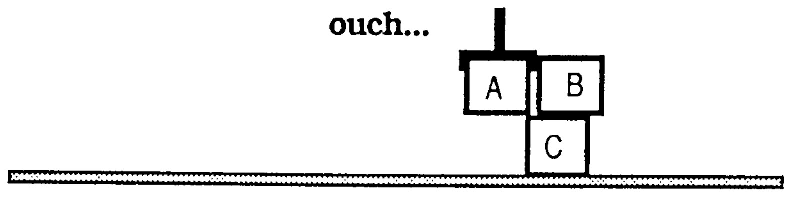

Figure 3 - *Adding multiple actors*

moved B onto C *after* the PI had successfully planned (clear C) (which would then have been true in the world), the PI winds up in the same situation as the PC.

4. Planning in Dynamic Domains

In this section, we introduce a simple dynamic domain in order to discuss proposed dynamic extensions to nonlinear planning. We show where plan interpreting, as well as extensions to plan compiling, break down in this domain and propose criteria for successful dynamic planning.

4.1. The Traffic World - A Dynamic Domain

For the study of dynamic planning, we have implemented the simulation environment pictured in Figure 4. The Traffic World consists of a straight stretch of a four-lane highway along which "cars" travel at various speeds. The cars make lane changes as they move according to their "personality types," as indicated in the figure. The goal of a planner in this domain is to get a robot travelling at a speed slow in relation to car speed safely across the road. The planner may obtain car speed, position, and direction information via instantaneous snapshots of the world, but is ignorant of car type.

We claim that this domain is dynamic. It incorporates multiple actors effecting continuous change in the environment independent of the planner's actions. The planner can only "see" to the end of the visible stretch of the road, so it must deal with the appearance of new (moving)

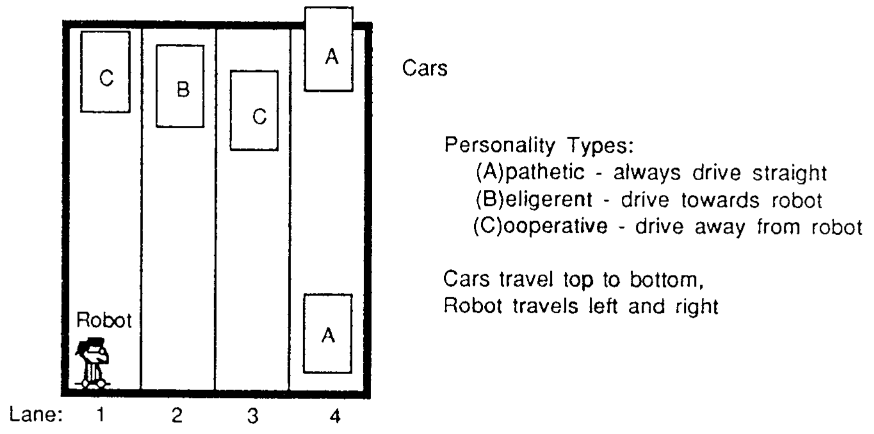
C
B
C
A
Cars
Personality Types:
(A)pathetic - always drive straight
(B)eligerent - drive towards robot
(C)ooperative - drive away from robot
Cars travel top to bottom,
Robot travels left and right
Robot
A
Lane: 1 2 3 4

Figure 4 - *The Traffic World*

objects in the domain. Since car type information is witheld from the planner's knowledge, the domain is incompletely specified.

A nonlinear approach to planning in the traffic world begins with a planning operator of the form

```
(to-do (cross-road robot)
  (do (cross-lane robot (lane ONE))
      (cross-lane robot (lane TWO))
      (cross-lane robot (lane THREE))
      (cross-lane robot (lane FOUR))))
```

From the discussion in Example 1, we see that, barring random successes, no PC can be effective in the traffic world. Since the domain changes continuously, the PC's expected initial situation may not persist at execution time. However, we can show that the PC approach fails even if plans can be generated very quickly. Suppose that the cross-lane operator is

```
(to-do (cross-lane robot (lane num))
  (achieve (at robot (left-side (lane num)))
           (clear-distance (lane num)
                           (* (speed robot) (width (lane num)))))
  (do (move-robot robot (left-side (lane num)) (right-side (lane num)))))
```

so that the planner checks that some area of the lane near the robot is clear before crossing. Since a PC only enforces these conditions before execution, by the time (at robot (left-side (lane TWO))) becomes true, the crossing of the first lane will have used up the time over which the clear-distance condition for lane TWO was expected to hold (ditto for lanes THREE and FOUR). An example of this problem is shown in Figure 5(a).

As in Example 1, since a PI checks the world before crossing each lane, it performs better than a PC. However, we can show that the PI approach also fails in the traffic world. There are two reasons for this, based on the PI's disjoint planning and executing operations. Figure 5(b) shows the robot being hit while stopped between lanes; here the PI is in the process of planning its next move. The PI approach fails in this instance due to its need to "stop and think" in a dynamic domain. A PI will also fail, as will a PC, in situations where some (beligerent) car moves into the robot's lane as it is moving, after the clear-distance precondition has been established.

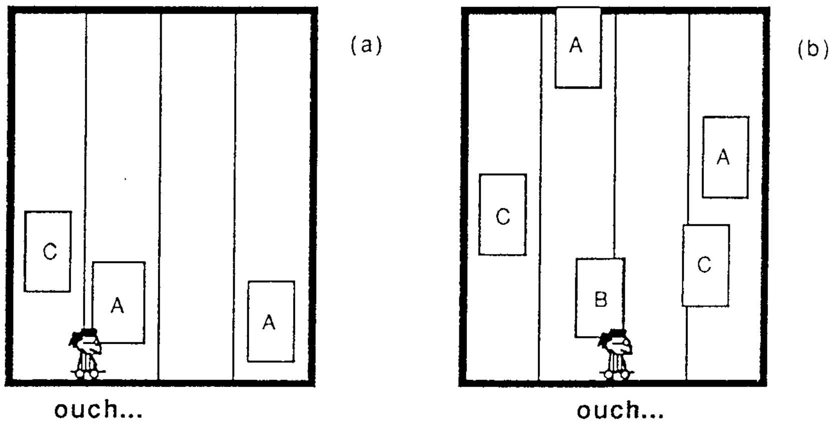

Figure 5 - *How PCs and PIs fail in dynamic domains*

4.2. Past Work

Now that we have seen the effect of the SWA on nonlinear planning's prospects for handling dynamic domains, we consider some extensions to nonlinear planning that attempt to relax the SWA. These extensions incorporate a temporal reasoning component into a nonlinear planner.

DEVISER [Vere83] is a PC that schedules actions and events according to start windows and durations expressed as absolute dates.[2] External events and actions are permitted only at scheduled times, so that the planner is aware of them during plan generation (thus enforcing an only slightly relaxed version of the SWA). Plans fail during execution if expected conditions fail to hold. Thus DEVISER also fails to "notice" the occurance of (on B C) in the non-static domain of Example 1.

More recently, the work of Dean [Dean86] and Miller [Miller85] has been applied to several non-static domains. The FORBIN planner schedules tasks in a job-shop domain subject to time, cost, and deadline constraints; here we consider its temporal planning system based on Dean's Time Map Manager (TMM). The TMM is a database manager for tracking the validity of facts and the occurance of event in time. Dean describes a hybrid planner, based on NASL, using breadth-first subtask expansion until all planned tasks are primitive, at which point execution may begin. As such, this planner is a PC. An important feature of this system is its use of *monitor* tasks (c.f. [McDermott77], [Dean86]) used to gather information during plan execution. This information may affect the execution status of tasks in the current plan as well as the future planning goals, so that these plans have a degree of conditionality not found in previous work.

As we showed in section 3, examining the world during plan *generation* leads to greater flexibility in dynamic domains. Since the SWA does not hold in dynamic domains, we cannot expect a planner to generate plans with a priori conditionals for arbitrary exception situations which may present themselves during execution. Hence, the FORBIN system is limited in its adaptability to the actions of independent actors. Using our traffic world example, suppose that the robot can plan to travel either left or right, with monitors causing it to reverse direction if a crash will occur should the robot continue in its current travel direction. The situation in Figure 6 occurs because (the apathetic) car A's path causes the robot to start moving left, and (the beligerent) car B's path causes the robot to start moving right, etc.

The problem with this approach is its lack of effective *situation projection*. In order to avoid the collision in Figure 6, the robot needs a reasonable expectation of the intended headings of each car. This information is only available through an analysis of the previous behavior of each car. Making decisions in dynamic situations needs to be based on "what have you seen lately" rather than "what's happening now" - by the time an exception situation presents itself, it may already too late for the planner to avoid failure.

4.3. Towards Dynamic Planning

We have shown how attempts to impose the SWA on dynamic domains have been unsuccessful in application. In this section, we argue that no planning system treating plan generation and execution as disjoint operations can be successful in dynamic domains. We propose that dynamic planning be based on integrating these two activities, and outline a reasoning system taking this approach.

4.3.1. Making and Maintaining Assumptions

It is clear that in dynamic domains a planner must make assumptions as to what the world will look like as its plans are executed. These assumptions are defeasible inferences made through examining the world; they may be withdrawn only as the world is re-examined. In order for a dynamic planner (DP) to maintain an accurate "world model," it must constantly update the

[2]These times may be expressed as functions of other time variables which DEVISER attempts to instantiate; when the plan is complete, these variables must have numeric values.

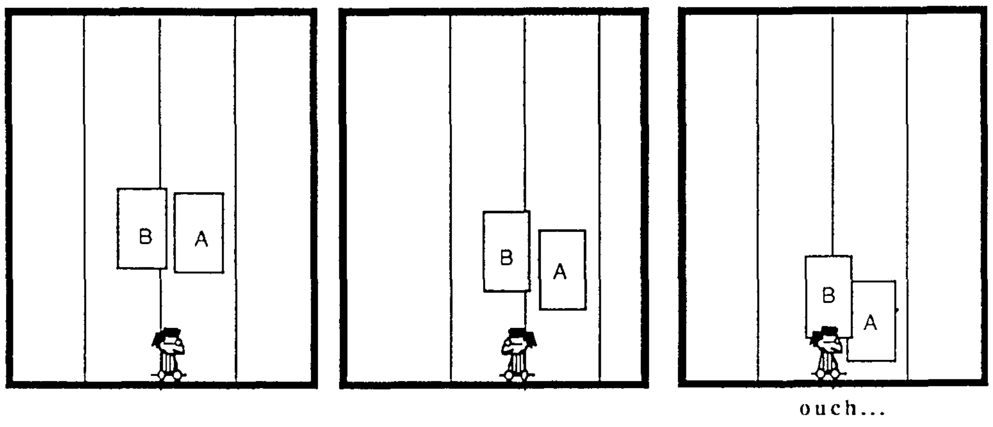

Figure 6 - *Using monitor-based contitionals during plan execution*

validity of the assumptions it has made. As shown in the previous section, it is insufficient for an assumption's validity to simply be negated by new information. Rather, the planner needs to know when an assumption *may be failing,* not simply that one has failed. The time-map provides a formalism for detecting failed assumptions, but enforces a discrete truth value on the facts and events it maps. It but does not permit reasoning about the impact of continuous change on assumption validity.

4.3.2. Observations

The DP we envision will operate in a domain changing radically over short periods of time, such as the traffic world. We make the following observations on operation in such a domain:

(1) In any given dynamic situation, the space of possible actions is small, being governed both by the short-term (safety) and longer-term goals of the DP.

(2) Failure conditions noted after the fact are in general insufficent for executing dynamic plans. Since most failures occur, as do actions, over time, we may look for conditions that may lead to failures and hence predict plan failure.

(3) It is in general impossible to enumerate the various execution-time conditions that may result in plan failure; however, it is *always* possible to specify *success criteria* for any given plan based on the preconditions of its operators. These criteria may be imposed as assumptions on the future of the domain and monitored for potential failure thereafter. For example, one success criterion for a cross-lane event in the traffic world is

(clear <lane> (<from-time> <to-time>)).

(4) There is a fundamental difference between plan generation (in the abstract) and plan execution (in an application domain). While *plan scripts* may be generated offline under the SWA (with appropriate attention paid to assumptions on the domain), action in a dynamic domain is governed by an *instinctual reaction* mechanism. Such a mechanism selects actions to perform from a store, based on the impact of changes in the world on the DP's short-term goals (ususally, continued successful operation).

These observations provide the basis for our approach to dynamic planning. Since we expect the near-term future to bound the space of possible actions, we envision a situation projecting system making plan selection decisions based on observations mapped into the predicted future. Longer-term goals of the DP may be planned offline using a hypothetical (and optimistic) world model; these plans influence plan selection in the online reaction system.

5. Conclusions

We have introduced the properties of dynamic domains, and showed how the assumptions on the static-ness of nonlinear planning domains limit the use of such planners in dynamic applications. We have shown that plan interpreting holds promise for use in dynamic domains for some of the same reasons it has been overlooked in planning for static domains. We have introduced the dynamic domain of the traffic world and propose it as a new "toy" domain for dynamic planners. In examining the approaches to extending nonlinear planning to dynamic applications, we have presented observations on the nature of dynamic domains as well as criteria for successful dynamic planning. Our research in dynamic planning continuing with the implementation of a dynamic planner operating in the traffic world based on the desiderata of section 4.2.2.

6. References

[Chapman85] Chapman, David *Planning for Conjunctive Goals,* Masters Thesis, Dept. of Computer Science, MIT, 1985.

[Dean86] Dean, Thomas *Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving,* Ph.D. Thesis, Dept. of Computer Science, Yale University, 1986.

[Hendler86] Hendler, James *Integrating Marker-Passing and Problem Solving: A Spreading Activation Approach to Improved Choice in Planning,* Ph.D. Thesis, Dept. of Computer Science, Brown University, 1986.

[McDermott77] McDermott, Drew "Flexibility and efficiency in a computer program for designing circuits," Technical Report #402, MIT AI Laboratory, 1977.

[Miller85] Miller, David *Planning by Search Through Simulations,* Ph.D. Thesis, Dept. of Computer Science, Yale University, 1985.

[Sacerdoti77] Sacerdoti, Earl *A Structure for Plans and Behavior,* Elsevier, North-Holland, 1977.

[Tate77] Tate, Austin "Generating Project Networks," *Proc. IJCAI* 5, pp. 888-893, 1977.

[Vere83] Vere, Stephen "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* V. PAMI-5, #3, pp. 246-267, 1983.