# ABSTRACTION PLANNING IN REAL TIME

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Richard Washington

March 1994

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Barbara Hayes-Roth
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Nils J. Nilsson

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Yoav Shoham

Approved for the University Committee on Graduate Studies:

# Abstract

When a planning agent works in a complex, real-world domain, it is unable to plan for and store all possible contingencies and problem situations ahead of time. The agent needs to be able to fall back on an ability to construct plans at run time under time constraints. This thesis presents a method for planning at run time that incrementally builds up plans at multiple levels of abstraction. The plans are continually updated by information from the world, allowing the planner to adjust its plan to a changing world during the planning process. All the information is represented over intervals of time, allowing the planner to reason about durations, deadlines, and delays within its plan. In addition to the method, the thesis presents a formal model of the planning process and uses the model to investigate planning strategies. The method has been implemented, and experiments have been run to validate the overall approach and the theoretical model.

# Acknowledgements

I would like to thank my advisor, Barbara Hayes-Roth, for her support and guidance over the many years I have spent in her research group, and especially for her efforts to support my final push to graduate.

I would also like to thank my other committee members, Nils Nilsson and Yoav Shoham, for their incisive and insightful comments on my work. The work is clearer for their input.

Thanks also to the additional members of my defense committee, Russ Altman and Mark Cutkosky, for their time and interest. In particular I would like to thank Russ for making the defense as smooth a process as possible.

Without my family, none of this would have been possible. They have always offered their encouragement throughout the entire process. They can stop holding their breaths now.

It would fill an additional volume to name all the people whose paths have crossed mine during my nearly ten-year tenure in grad school. I appreciate every one.

Special thanks are due to my officemate, Michael, for managing to put up with me for so many years. His willingness to listen to and talk about everything from contorted mathematical analyses to the latest baseball scores made the long periods of frustration pass more quickly. And most of all, he made sure that I kept the proper attitude about the entire process.

I started my grad school career with Amy and Andy, and we wandered together down the long, winding path through the jungle of AI. They found their ways out well before I did, but their friendship and presence has accompanied me throughout the entire process, even after they physically left.

Thanks to Karen for her constant friendship and willingness to do random things on a moment's notice. She also performed above and beyond the call of duty by actually reading my thesis and discussing my work with me.

Janet, David, Lee, Vlad, the Philippes L. and M., Karl, Marko, John, Serdar, Jan Eric, and all the other BB1 group members over the years have listened to my many rambling talks and half-baked ideas presented as if they should be interesting and revolutionary. I appreciate their patience and feedback. And what would grad school have been without the weekly lunches and periodic Häagen-Dazs runs?

Thanks to all the other grad students, especially Pandu and Alon, who have listened to my ideas over the years and attempted to steer me back onto the path of reason.

Thanks to the folks at Interval Research, especially John and Ramin, for providing me with both the financial resources and the freedom that allowed me to finish my thesis.

And thanks to Philippe L. and Jean Paul for providing me with the incentive I needed to put in the final effort.

I owe special thanks to my former advisor Paul Rosenbloom for getting me started in the field, and for his interest and concern even after I was no longer working for him. The ideas I learned while working with him have influenced my work ever since.

I owe my interest in computer science to Joe O'Rourke, who served in the unofficial role of advisor during my undergrad days. His enthusiasm and inspired teaching started me down this path.

I would like to pay special tribute to the administrative staff at KSL, in particular Grace, Michelle, and Peche. They not only keep the creaky machinery of the lab working smoothly, but they also make the lab a friendlier and more fun place in general. Particular thanks to Peche for providing some of the real information abstracted into the grant proposal domain in the thesis.

I have also shared many years, as well as a house, with Larry and Dave. Thanks to them for everything from road trips to LA to burrito runs to Simpsons and football—in other words, for making sure I was living in the style to which I had become accustomed. Credit is also due to Michael (the other one), who moved into a house of terminal grad students and has managed to remain relatively sane.

I owe my involvement in biking to John, who got me started on what became a serious hobby and a great escape from the alternate reality of grad school. Thanks also to all the Lemmings, who put up with my twisted idea of fun. And particular thanks to the Stanford Cycling Team, which has allowed me to ride with them as everything from a squirrelly novice cat D to a crafty old cat D to a retired old geezer.

And thanks also to all the people over the years in the Stanford Symphony, my other escape from the pressures of grad school.

And finally, to all the people who have greeted me over the years with the question "Have you finished yet?" I would like to take this opportunity to say one thing. Yes!

This research was funded by the Defense Advanced Research Projects Agency, under NASA Grant NAG 2-581, and NIH 5P41 RR-00785, and also by Cimflex Teknowledge Corporation, Contract 71715-1 under ARPA contract DAAA21-92-C-0028.

# Contents

# List of Figures

# Chapter 1

# Introduction

When we as humans plan an activity, we draw upon a vast store of knowledge and experience. This knowledge and experience is accumulated over our lifetime, and in some cases is even built up over eons of evolutionary development. When confronted with a new activity to plan, we may find that we have done the same or a similar activity in the same or similar situations, and we can use that information to suggest what we should do in this new situation. Even when we are confronted by an unfamiliar activity or situation, we often can find aspects of the situation and the activity that will be familiar, so we can piece together a patchwork plan from bits of other plans.

Occasionally, we can find ourselves in situations that are so remote from our previous experience that we have no idea what to do at first. For instance, we can approach that state in games like chess, where as novices we learn the rules of the game, but we have no experience or knowledge to suggest appropriate courses of action. Over time, we develop the knowledge to be able to recognize familiar situations and recall the appropriate actions for them, but at first we are basically searching blindly. To handle new situations, we have the ability to build plans from scratch when our knowledge fails.

Now consider the plight of a computer agent dropped into the real world. It has neither experience nor evolution to guide it through the countless situations and activities that it will face in the world. The agent was likely programmed to handle some small space of situations, but the relative size of this small pre-programmed space compared to the vast space of possible situations it may encounter makes it much more likely that it will range outside of its store of knowledge and experience. As with humans, the computer agent must have a facility to fall back on that will allow it to reason in these foreign environments. And with the computer agent, this facility is even more important, since it will need to rely on these skills a large percentage of the time.

This thesis addresses the problem of how a computer agent builds plans in the case where it does not have pre-compiled information about how to achieve its goals. This is the skill that the agent will need to have so that it can fall back on this skill when its meager store of experiences is inadequate for its environment.

The field of planning has spawned a range of techniques that enable computers to construct plans to achieve specific goals. Unfortunately, the world is not as simple as these techniques would like to assume. The world is a complex web of interacting forces, many too many for the computer to account for. And on a smaller scale, it is even too hard to model single human (or other) agents in the world. So the computer agent can only imperfectly predict what the future holds. In particular, while achieving its own goals, it may find that actions do not have the precise results its internal model produces.

The agent may also need to monitor continuous events that coincide or overlap. It may need to respond with actions over a duration of time. The actions themselves may coincide or interact. So the agent must be able to represent and reason about these continuous temporal events.

The world waits for nobody, not even the planner. As the environment dynamically changes and moves forward, the agent faces time constraints within which it must take actions. If the agent removes itself from the outside world to think deeply about a problem it is trying to solve, it may awaken again only to find that the world has passed it by, and the problem has either disappeared, changed, or progressed to the point where the original solution is worthless. So the agent must be able to work within varying time bounds imposed from the outside environment. Furthermore, it must not sever its ties with the environment while it reasons, but rather should monitor the world and change its internal model of the world, and its resulting plan, accordingly.

We present a planning method that reasons about this dynamic world, continually updating its world model to agree with the information it receives from the outside world. The world model represents the temporal nature of events and actions. The planner incrementally builds plans to achieve its goals, and adapts these plans to the changing situation. The method as a whole provides an agent with the facility for building plans outside of its breadth of experience.

## 1.1 Existing planning work

Other researchers have delved deeply into specific aspects of the planning problem, exploring particular techniques or formalisms designed to work well within that specific niche. Little attention, however, has been directed towards finding approaches that are appropriate for the entire problem as we have described it. As is often the case, adding the additional requirements of the integrated approach complicates the details of the solution, but also constrains the approach to eliminate many candidate approaches. In a general sense this work follows directly from the constraints that building a real-world planner places on the process. In a real sense many of the details were dictated by these constraints.

The majority of current planning work concentrates on planning in a space of plans [Chapman, 1987; McAllester and Rosenblitt, 1991]. The operations within this space are adding ordering constraints between steps, adding equality constraints between variables, or adding steps to an existing partially-ordered plan. This general technique is not appropriate for real-time environments, since the partially-ordered plan does not necessarily contain actions that are applicable in the current world. Therefore, if an agent using this planning technique is forced to act before completing its plan, it would receive no guidance from the plan.

This thesis is based on an incremental planning technique. Other planners for resource-bounded situations have taken the approach of incrementally expanding a plan (see for instance [Drummond, 1989; Durfee and Lesser, 1986; Bratman *et al.*, 1988]), but none has addressed the range of issues this work does. Existing approaches are limited to static worlds, unchanging during the planning process, and static world descriptions, making it cumbersome or impossible to represent the continuous nature of the real world.

Another approach used to build a plan under resource bounds is to use a "quick and dirty" approach to build an approximate plan, then to improve the plan incrementally [Rymon *et al.*, 1992; Boddy and Dean, 1989]. The advantage is that often the rough approximation is an executable plan from the current state to the desired goal state. But the approximate plan may be arbitrarily bad. In addition, the approximate techniques again use simplified world descriptions that do not capture the dynamic nature of the real world.

Formal approaches to resource-bounded planning are designed to explore and analyze features of the planning process under specific and carefully crafted conditions [Horvitz, 1987; Dean and Boddy, 1988; Ginsberg, 1994]. These conditions are even more severely limiting than for the other resource-bounded planning approaches, and the assumptions hold for only a minuscule portion of the universe of planning situations where a planner might be needed.

The most popular technique for "fast" real-time planning has been to precompile plans for all or many possible world situations [Schoppers, 1987; Mulder and Braspenning, 1992]. For a limited domain that can be carefully controlled this is attractive. After all, this places the problem back into the region where the computer can rely on existing experience and knowledge to solve the problem, which we have said seems to be preferred at least by humans. The problems are twofold. The "experience" must be generated brute force rather than through experiential learning, so generating this knowledge is a formidable computational task, not to mention a daunting storage and retrieval task. And beyond the edges of its knowledge, the computer agent is lost, with not even a shred of information to tell it what to do. In a real-world domain, the storage and computational complexity would make it unrealistic to store all possibilities, or even a large portion of the space of possibilities, so the chance of the computer agent straying beyond the limits of its knowledge is dangerously, and we think unacceptably, high.

The use of abstraction in planners is a well-accepted technique to organize the planning process and make it more efficient. Existing work has focused on constructing and using abstraction in planners freed from the confines of resource bounds [Knoblock, 1991; Unruh, 1993]. But since one of abstraction's intended effects is for efficiency, it is natural to apply it to the problem of planning under time constraints.

Since we have argued for a world in which events and actions occur over intervals of time, our own work rests on work for representing temporal information. The research that focuses particularly on temporal representation is concerned more with the properties of the temporal formalisms than with their application in a resource-bounded environment [Allen, 1983; Dean, 1985; Vere, 1981; Penberthy, 1993]. Since we are building on the ideas, we use temporal representation more as a tool than an end in itself, so we are more concerned with the details of how to represent and reason about time in a way that will support our more general planning goals.

One of the fundamental distinctions of this work from much of the rest of the field is its explicit attention to the fact that the world is a dynamic and somewhat unpredictable environment. Thus any planning agent within the environment must be prepared to receive information at any time that changes its model of the world. Furthermore it must incorporate that information into its planning process, modifying its plan as necessary to account for this new information. Traditionally planning and replanning have been considered disjoint operations. We will argue that the two are inseparable in the changing real world, where information may change during the planning process.

## 1.2 What is real time?

The term "real time" has nearly as many definitions as there are researchers who investigate it [Laffey *et al.*, 1988]. A common definition is that a real-time system should respond within a fixed time bound. Although producing a response quickly is important, our emphasis is not on the initial response time, but rather on the incremental improvement of the response over time.

A fixed time deadline is important for applications where a single response is required within a hard deadline, after which the response is useless. For example, a system producing real-time video output would need to produce a new video image every 1/30th of a second.

In our work, we are more interested in situations in which the time deadlines are variable, even possibly changing after the agent receives them and starts to plan. Also, the deadlines may be too short for the agent to produce an optimal solution. Therefore, we are particularly concerned with the ability of our method to adapt to whatever time resources are available. The absolute amount of time is not of paramount importance, as it is in real-time control systems. Rather, the critical feature is the behavior of the agent under a range of time bounds, where the bounds themselves may change during planning.

## 1.3 Contributions

This thesis describes an approach to planning that constructs plans under time constraints while adapting to changes in the world. In particular, the planning method demonstrates the following abilities:

- The method operates within arbitrary and changing time bounds, building the best plan it can within the amount of time available. Given more time, it will produce a better, more complete plan.
- The method continually incorporates new information into its model of the world and adapts its plan accordingly.
- The method represents the dynamic and continuous nature of information and events in the real world.

The desired planning behavior is achieved by incrementally building plans at multiple levels of abstraction. In particular, the approach is realized in the following ways:

- The approach incrementally builds plans at multiple levels of abstraction. The plan at any given level of abstraction may be incomplete, but is still used as a partial framework around which the planner can build more specific plans. As the abstract plans deepen, they provide a more constraining and stable framework for the specific plans, but even in an incomplete state, they offer guidance to limit the search through the more specific space.

- The approach plans forward in time, starting from the current state and adding operators that lead towards the goals. This ensures that there is always a plan prefix that is applicable in the current situation, even if the planning process has not completed. If the plan has been expanded at the lowest level of abstraction, then the prefix at that level can be executed in the current situation.

- The approach merges planning and replanning into a seamless and inseparable process. A plan is a changing data structure, changed by the planner adding actions to it, by the world changing in unanticipated ways, and by the planner revising its plans to match the world model.

- The approach represents information temporally over intervals of time. It maintains a global record of its observations, expectations, and intentions, not just at a particular instant in time, but over all times.

## 1.4 A guide to the thesis

In this chapter, we describe the problem that has motivated the work in this thesis, and we broadly describe our work and how it extends current work in the field. In Chapter 2 we discuss an example that illustrates the abilities that a planning agent needs to work in a real-time, real-world domain. In Chapter 3 we explore our planning method in more detail, elaborating the particular techniques that endow the method with its particular behavior. In Chapter 4 we expand the discussion to include the implementation of the method in a working system. In Chapter 5 we present an analysis of the method using a mathematical model of the problem and approach. In Chapter 6 we see the results of examining the method empirically for an office robotics domain. In Chapter 7 we go into more detail on the existing work in the field and how our work compares to it. And in Chapter 8 we discuss open issues and conclude.

# Chapter 2

# An Example

Consider a robotic agent with the task of building communications towers out of various materials. For instance, the agent could be building towers using such materials as wood poles, metal poles, plastic poles, cement, or pontoons. These pieces would be put together into a base and a latticework for the tower (see Figure 2.1).

Suppose that the agent has descriptions of the domain at multiple levels of abstraction. At the highest level of abstraction it could have operators for building pieces of the structures, for instance a tower base or a segment of a tower. Adding each piece of a tower depends on previous pieces being there; for instance, the agent wouldn't plan to put the top on the tower before it planned to build a base.

At a lower level of abstraction the agent could have operators for more detailed operations: digging holes, constructing forms, mixing cement, pouring cement, assembling poles, etc. Again, there are dependencies among some of the operations. For instance, pouring the cement requires that the forms are constructed.

At even lower levels the agent could plan the use of tools and materials to implement the operations, and at the lowest level the agent could plan its executable motions.

One day, the agent receives two tasks: (1) build a 10-foot tower in one day, and (2) build a 30-foot tower in three days. The actual time to build the towers is just under a day for the smaller tower and two days for the larger tower, but because of delays the agent will

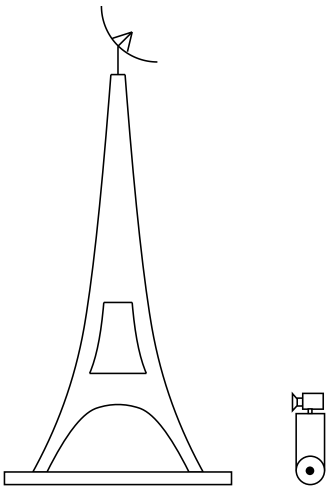

Figure 2.1: A robot building a communications tower.

want to interleave steps (see Figure 2.2). Given the initial goals, the agent gets to work putting together a plan to build the towers. Suppose it is on dry land, with a supply of metal poles, wooden poles, and cement available.

At the most abstract level, the available operators that are applicable in the current state are the operators to build the bases of the two towers. Building a higher level of a tower depends on having the lower levels built, so those operators are unavailable at first.

Since there is no dependency between the two towers, the agent decides to add both operators into the plan as a set (see Figure 2.3). Note however that the agent can be in only one place at a time, so the time intervals over which the towers can be built are constrained to avoid overcommitting the agent.

Because it only has access to certain base-building tools, for instance a cement mixer and post-hole digger, for a short time, the agent decides to leave the abstract plan unfinished while it expands out the base-building plan to an executable level. It will return to add more to its abstract plan once it has expanded out some more specific plans. It could even start executing the beginning of the plan before it has finished planning the complete task.

The agent expects that once it has executed the actions in service of building the bases of the towers, the bases will exist. Because of possible problems, it will wait until it gets confirmation of the bases existing before asserting the truth of the statement, but it can

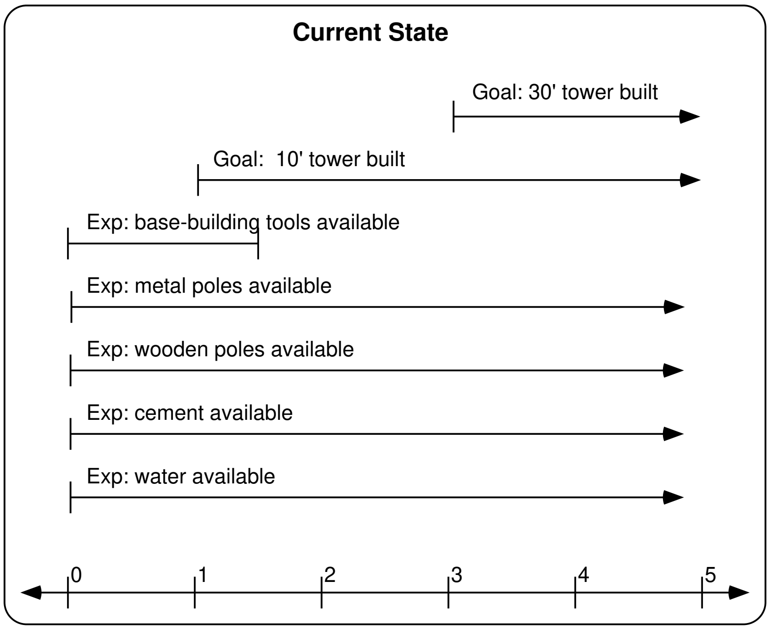


Figure 2.2: The initial information available to the robot. Its goals are to build towers. Its current expectations are that various building materials will be available for the foreseeable future and that some tools will be available for a limited time.

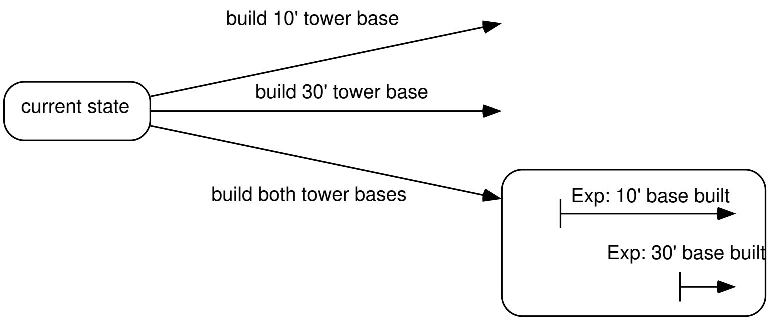


Figure 2.3: Abstract operators applicable given the current information.

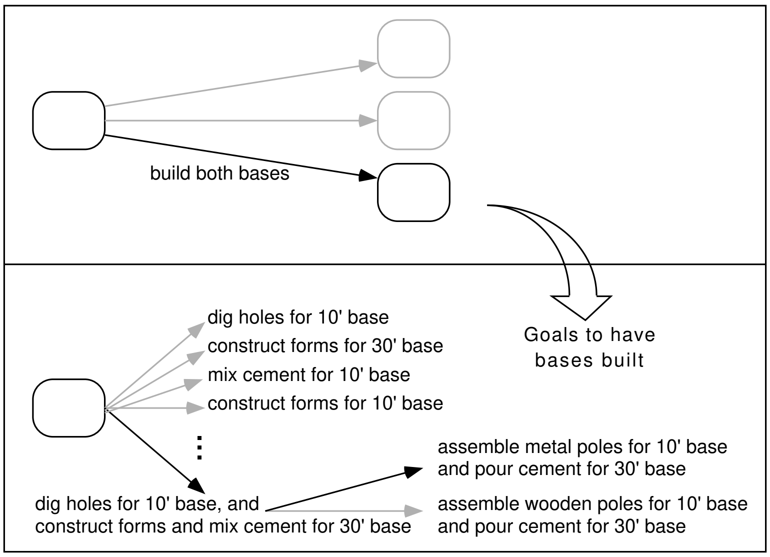


Figure 2.4: Abstract state information propagates to produce goals at the next lower level.

plan based on the expected information.

Since the action to build a base is much more abstract than an atomic action for the agent, the next step is to figure out how to realize the abstract action. To do that, the agent sets up some goals at the next lower level of abstraction. In particular, it will post goals that the bases exist within the expected time ranges (see Figure 2.4).

At the lower level of abstraction, the available operators are constructing forms, mixing cement, and digging holes. The agent also knows how to work with pontoons and plastic poles, but those are unavailable, so the operators that deal with them are also unavailable in the current state. So for each of the towers, the agent can construct forms, mix cement, or dig holes for that tower. Given that forms have been built and the cement mixed, the cement can be poured. Given that holes are dug, poles can be assembled. This gives the agent a number of possible action sequences to consider. For instance it may plan to construct forms, mix cement, and pour cement for one of the bases.

Because cement takes a while to dry, the agent prefers assembling metal or wooden poles for the smaller tower, since the deadline is short. In addition, the extra strength afforded by the metal poles is preferred over the wooden poles. For the taller tower, the added stability

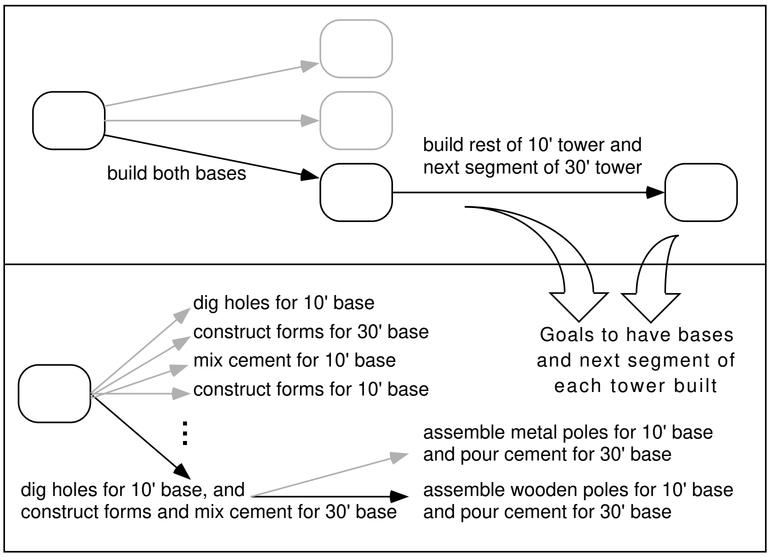


Figure 2.5: Further abstract search may constrain the lower-level search and may change the best plan at the lower level.

of the cement base is preferable, so cement is preferred for that. So the agent favors the plan that builds the base of the smaller tower out of wooden poles, and the base of the taller tower out of cement.

The agent then uses this partial plan to generate goals at the next lower of level of abstraction, where it will begin to plan the more detailed use of tools to implement the steps it has decided to use. The detailed plan itself is then used to generate goals for low-level executable robotic movements.

Now the agent returns to its abstract plan, adding another step. The next stage of each tower is planned. The short tower may be built in one more step, while the taller tower requires a few more steps. But once again, the two are independent, except for the shared resource of the agent, so they are added to the plan as a set. The abstract plan now has the agent building the entire short tower and the base and first segment of the second tower.

The new steps added to the abstract plan are now used as further goals for the lower level plan (see Figure 2.5). In particular, now that the tower structure is being planned more completely, the agent may notice that using metal poles for the bases is inadvisable, because a large number of metal poles may be needed for the towers themselves. So the additional goals constrain the choices available at the lower level. In particular, the agent will switch from metal to wooden poles for the smaller tower.

At this point, the agent learns that other construction agents in the area have depleted the water source, and that there won't be enough water for the cement for a day. This precludes the possibility of mixing cement for that time period. This drastically changes the part of the plan that deals with the larger tower, which now would not be finished in time given this delay. This narrows the choice of plan prefixes to the point where building the base with wooden poles is clearly preferred.

At this point the agent can commit to the plan using the wooden poles, considering the time available and the superiority of that possibility. So it now can begin executing the plan. The agent will build the base of the shorter tower because of its deadlines.

While the bases are being built, the agent may find a tool missing and may have to use a different tool to accomplish the same task, but the small change doesn't disturb the overall plan.

The agent continues on with its plan for building the remainder of the towers. While it is doing that, it receives word that the taller tower should really be only 20 feet tall. This changes the part of the plan for that tower while leaving the rest of plan unaffected.

The agent finishes the base of the first tower, and now moves on the the rest of that tower and the base of the second tower. Because of the deadline for using the post-hole digger, it decides to begin the base of the second tower.

However, one of the other construction agents accidentally backs into the already-constructed base of the shorter tower and dislodges some of the poles. So suddenly our agent's belief that the base is done has been contradicted by information it receives, and it will have to replan that part of its plan.

In this way the plan for constructing the towers gets developed incrementally and modified as new information arrives. The agent executes the beginning of the plan before it finishes expanding the plan completely, and continues to add to the plan while it is executing.

The construction agent exhibits the types of abilities that the work in this thesis is designed to produce. In particular:

- The agent builds plans incrementally at multiple levels of abstraction, allowing it to meet deadlines as they appear. The agent first builds an abstract plan that includes steps for building tower bases, then expands that to an executable level because of time constraints on the tools available for building bases.

- As the abstract plans are extended, they add constraints to the more specific plans. The additional information about how the upper segments of the towers will use metal poles constrains the agent to avoid using metal poles for the bases of the towers.

- The agent represents and reasons about temporal information, including deadlines, durations, and delays. The agent is given deadlines to begin with, and learns about new deadlines as it expands its plan. It receives information about the lack of water, and reasons about the delay that would add to the plan.

- The agent adapts its plan to fit the current state of the world. As it receives information, the agent incorporates it into the evolving plan. When the agent learns of the lack of water, or the damage to the shorter tower base, it modifies its plans according to the new information, and continues planning from that point.

# Chapter 3

# Abstraction planning in real time

We describe the approach by decomposing it into its representation, its method for plan construction, and its replanning behavior. The representation is the foundation on which the planner is built. Since the planner is designed to work with concepts and actions over time, the representation is designed around a temporal framework. The plan-construction method is described by first explaining in some detail how individual operators are selected and added to the plan. Then we step back to see how the plan as a whole develops at multiple levels of abstraction. We then see how replanning is integrated into the planning method, both in terms of individual operators and in terms of the overall behavior. Although many of the details have been influenced by what we learned from implementation, this discussion is restricted to those aspects that would appear in any implementation.

## 3.1 Representation

The basic representation for operators derives its general form from the classical planners such as Strips [Fikes and Nilsson, 1971], Nonlin [Tate, 1977], Sipe [Wilkins, 1988], and Deviser [Vere, 1981], although some details have changed to support the particular needs of this approach. The representation for states requires a general temporal representation, such as Time Maps [Dean, 1985]. Again, there are particular features that are adapted to this particular approach.

### 3.1.1 State representation

The state of a planner must be expressive enough to represent the information necessary to decide whether to plan and when to take particular actions. We choose for our state a representation that captures the internal mental state of the planning agent. The internal mental state captures not just the agent's beliefs about the current state of the world, but also its beliefs about the past and future. This is an extension of traditional planners, which view states as points in time. The expanded notion of state is because the choice of an action may depend not only on what the world state is at a particular point in time, but also on the state of the world before and after that time. For instance, if the agent is driving towards a cliff, it must anticipate that it expects to drive off the cliff enough ahead of time to be able to brake to a stop. Or an agent may decide that since a warning sensor has been on continuously for 60 seconds, that it should take an action to correct the problem. To handle these sorts of situations, the agent acts based on its complete mental state about the past, present, and future.

Thus our idea of state is a timeline of intervals[1] representing the mental state of the agent. We divide the timeline into three distinct types of information: occurred, expected, and intended (this is derived from [Ash and Hayes-Roth, 1990; Pardee *et al.*, 1989]). Occurred intervals represent readings from the sensors with minimal interpretation. Expected intervals represent the agent's derivations about the world. Information from the occurred intervals is copied to the expected intervals, along with inferences about the persistence of occurred intervals and predictions, and other inferences made by the planner or other computational processes within the agent. The intended intervals hold the goals of the agent—both the initial goals that the agent is trying to achieve, as well as intermediate goals generated within the planning process.

By distinguishing these three types of timelines, the agent can compare the information on them to derive information that will drive its planning behavior (see Figure 3.1). For instance, if an interval appears on the occurred timeline that conflicts with a corresponding

[1] We use the term *interval* to refer to a proposition (in particular, that a parameter satisfies a predicate) over an interval of time. We use the term *time interval* to refer to the interval of time.

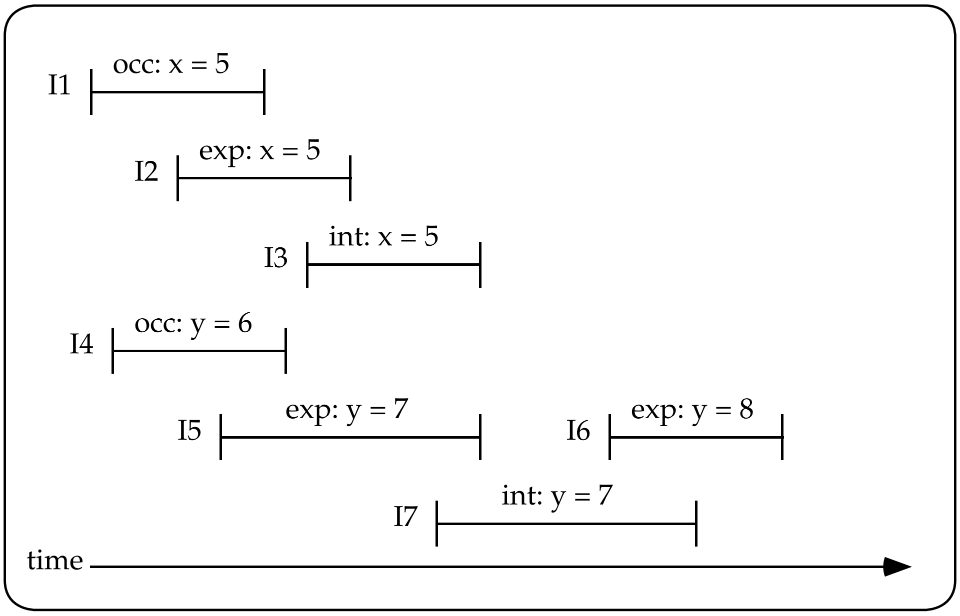


Figure 3.1: A planning state. Interval $I_1$, $I_2$, and $I_3$ are compatible. Interval $I_4$ mismatches with $I_5$, indicating that the expectation $I_5$ is not met. Interval $I_6$ mismatches with interval $I_7$, indicating that in the current state, the intentions are not expected to be met.

interval on the expected timeline, the agent can infer that its expectations may be incorrect; thus, any plan based on those expectations may need revising. If the agent generates an expected interval that conflicts with an intended interval, it is predicting that it will not achieve one of its goals; thus, it may be necessary to add to or change its plan to achieve that goal.

### Underlying temporal representation

The temporal representation necessary to support the planning method must capture a wide range of constraints among time points. All of the reasoning in the system is expressed in terms of time intervals, but constraints on the endpoints of the intervals suggest a point-based approach. The particular constraints that are generated are:

- $t_1 \; R \; t_2 + c$ for time points $t_1$ and $t_2$, an arithmetic relation $R$ (one of $<, \leq, =, \neq, \geq, >$), and a numeric constant $c$.

- $t\ R\ c$ for a time point $t$, an arithmetic relation $R$, and a numeric constant $c$.

This is similar to Reid Simmon's Quantity Lattice [Simmons, 1986].

Time intervals are built out of these time points. We make a careful distinction between open, closed, and half-open intervals, because they make an important difference when figuring out when and whether actions are applicable. For example, if an action is planned to execute as soon as a condition occurs, then the action will occur over an interval that is open at the start time of the condition. Without the distinction, it is impossible to correctly determine whether an action is applicable in a given situation, and unintended loops can result. So a time interval is of the form `(start-time-bound end-time-bound)` where a time bound is of the form `(relation time-point)`, where a relation is the appropriate arithmetic relation, in this case denoted as one of `:lt`, `:le`, `:eq`, `:ne`, `:ge`, `:gt`. For instance, to represent the half-open interval $[0, 10)$, the time-interval would be `((:ge 0) (:lt 10))`.

Given this representation, we also need to make a distinction between necessity and possibility. When looking up to see whether a time-point $t$ falls in an interval $i$, the system may find that $t$ is necessarily outside of $i$, possibly within $i$, or necessarily within $i$.

### 3.1.2 Operators

Operators, as in any state-based classical planner, represent a transition from one state to another. We generalize the traditional approach to allow a set of operators within a single state transition (as in Drummond's Situated Control Rules [Drummond, 1989]). We discuss sets of operators in more detail in Section 3.2 below. But a state transition, whether by one operator or a set of operators, will involve a change to the expectations in the world state. Consider the set of actions that an operator (or set of operators) generates. These actions, when executed in the world, have a set of expected effects. The effects are expectations because the system cannot necessarily predict with precision what effects the actions will produce in the real world. In many cases, sensor readings will confirm or deny the success of the actions at execution time, but at planning time the system can only post expectations about the actions' effects.

```
preconditions:
((robot-location (:ge 0) (:le 0) (≠ value :no-value) :assign
   rob-loc)
(grant-location (:ge 0) (:le 0) (= value rob-loc)))
resource conditions:
((robot-resource (:ge 0) (:lt 10))
(grant-resource (:ge 0) (:lt 10)))
actions:
((pickup-grant '(:ge 0) '(:lt 10)))
effects:
((grant-location '(:ge 10) '(:le 10) :at-robot :at-robot))
```

Figure 3.2: A sample operator to pick up a grant.

A single operator has preconditions and postconditions that are similar to a classical planning operator, but with added time intervals. The conditions describe the time intervals over which the conditions must be true for the operator to legally execute (see Figure 3.2). Each instance of an operator has associated with it an operator execution time variable $t$. The range of $t$ is those values for which each of the preconditions is true. Each precondition of an operator is of the form

```
(parameter-name start-bound end-bound value-test
:assign temp-variable)
```

The start-bound and end-bound describe a *relative time interval* over which the condition must hold. A relative time interval `(:ge` $a$`) (:le` $b$`)` describes the time interval $[t+a, t+b]$, where $t$ is the operator execution time. A precondition is true when its value-test is true everywhere within the interval $[t+a, t+b]$. The process of *triggering* an operator, or finding whether it is applicable in the current state, is the process of restricting the range of $t$ by matching successive preconditions against the state.

The value-test is the function that tests the condition, taking the parameter's value and returns T or NIL, and the temp-variable assignment (optional) stores the value of the parameter for a particular instance of the operator. Since the value-test may admit more than one value (for instance, a test that the value is less than 3 could match intervals where the value was, say, 1 and 1.6), the assignment is done for the value at the start of the interval. The temp-variable may be used in the value-tests of subsequent preconditions, as well as the actions of the operator. The assignment could be generalized to save the entire history of the parameter over the time interval, but at the cost of additional computational complexity in later conditions that inspected the value.

A parameter may appear in two conditions of an operator. For instance, the operator might look for an interval over which a parameter satisfied one relation followed by an interval over which the parameter satisfied another relation.

Additionally, an operator has resource conditions, which describe a common resource that multiple operators may share. A resource condition is of the form

```
(resource-name start-bound end-bound value-test)
```

with the bounds and value-test having the same meaning as in a precondition. For example, a blocks world operator to move a block with a single robot arm would use a resource condition on the robot arm so that other operators needing the arm would not be planned concurrently. They are particularly used with sets of operators, discussed in Section 3.2.

An operator also has binding conditions, which are meant simply to retrieve parameter values needed for the actions, but where the current value is not a condition on the legality of the operator. A binding condition is of the form

```
(parameter-name time-value :assign temp-variable)
```

For instance, a binding condition on a simple robot-motion operator could bind the actual position in the binding condition if the robot's motion is independent of its position (as in an open space). These are separated from preconditions because a change in a binding condition parameter will not require that an operator be re-evaluated.

The actions of an operator describe the executable actions implementing an operator, if the operator is executable. In particular, each action is of the form

```
(action-name start-bound-form end-bound-form
start-value-form end-value-form)
```

where all the forms are executed in the environment of all of the temp-variable bindings to produce the time-interval, start-value, and end-value for the action (the start-value and end-value allow the action to be parameterized).

The expected effects of an operator describe the expected intervals that result from executing the operator. They are each of the form

```
(parameter-name start-bound-form end-bound-form
start-value-form end-value-form)
```

where all the forms are executed in the environment of all of the temp-variable bindings to produce the time-interval, start-value, and end-value for that effect. If the intervals of the operator's effects start before the end of the last precondition interval, it would be possible to produce a sequence of operators that actually move successively back in time (see Figure 3.3). There is no prohibition against having overlapping preconditions and effects, though, since the backwards-firing operators are a degenerate case, and the added power of overlapping operators may prove useful in some domains. The only effect this has on the approach is that the default heuristic for deciding whether a goal cannot be achieved relies on forward-firing operators (described in Section 3.2.1).

Triggering an operator produces a range of time-points (actually, a time-point with a range of values) within which the operator can be legally executed. The operator's effects are then computed based on the legal operator times, along with the variables bound by the operator's conditions. The process of triggering an operator is described further in Section 3.2.

### 3.1.3 Hypothetical worlds

A triggered operator is not necessarily executed. It is merely added to the set of operators being considered for inclusion in the final plan, along with all the other possible operators. Thus its effects are hypothetical, and the current world state should not reflect the expected effects of a merely hypothesized operator. Therefore, as with a traditional classical planner, multiple states are maintained to hold the possible world states that the planner would find

Current state:

$x = 0$ `[10,20]`
$y = 0$ `[5,10)`
$z = 0$ `[0,5)`

Operator $o_1$

```
preconditions:
((x (:ge 0) (:le 10) (zerop value)))
effects:
((y '(:ge 0) '(:le 0) 0 0))
```

Operator $o_2$

```
preconditions:
((y (:ge 0) (:le 5) (zerop value)))
effects:
((z '(:ge 0) '(:le 0) 0 0))
```

Operator $o_3$

```
preconditions:
((z (:ge 0) (:le 5) (zerop value)))
effects:
((z '(:ge 0) '(:le 0) 1 1))
```

The sequence of operator executions will be:

$o_1$ `with operator execution time 10`
$o_2$ `with operator execution time 5`
$o_3$ `with operator execution time 0`

Figure 3.3: A set of operators that will appear to fire in reverse temporal order.

itself in (i.e., what expectations it would have) if it were to follow a particular plan. With the expanded notion of state, however, maintaining multiple states could be prohibitively expensive if done naively (this problem was acknowledged in Tom Dean's thesis [Dean, 1985]).

Our approach maintains multiple hypothetical worlds by recording differences between subsequent hypothetical worlds. Each world can be considered a mask on the previous, supporting world(s), changing only those intervals specified in the difference. In particular, if a hypothetical world specifies a value for a parameter over the time interval $[t_1, t_2]$ but not over the time interval $[t_3, t_1)$ or $(t_2, t_4]$, then the value of the parameter over the time interval $[t_3, t_4]$ is the value of the parameter in the supporting world for the time intervals $[t_3, t_1)$ and $(t_2, t_4]$, and the value in the hypothetical world over $[t_1, t_2]$. Note that the lookup in the supporting world over the two time intervals may recur if the supporting world is itself hypothetical. This makes lookups more complicated and computationally complex, but it makes world updates more efficient (since effects don't need to be propagated throughout the entire state network), and it reduces the space complexity as well.

The fundamental behavior of the approach does not depend on having hypothetical worlds represented by differences, but it rather represents a decision about the time-space tradeoff. The difference-based representation does make it more straightforward to identify actions that are appropriate for a particular world, since the only intervals within the representation for a hypothetical world are those intervals that differ from the supporting world. Therefore, actions that are triggered from intervals in a hypothetical world are necessarily dependent on features particular to that world.

Each hypothetical world has one or more *founders*, which are the intervals that are axiomatic in that world. They describe the fundamental changes that distinguish a hypothetical world from its supporting world. All other differences between a hypothetical world and its supporting world are derivable from the founders added to the supporting world. In the case of planning, a set of instantiated operators is the set of founders for the hypothetical world in which the operators are executed. The operators' effects are derived from the instantiated operators, and appear in the hypothetical world.

```
preconditions:
((pole-state (:ge 0) (:le 15) (= value :bending))
(bolt-state (:ge 10) (:le 15) (= value :deforming)))
effects:
((failure-state '(:ge 0) '(:le 15) :deformation :deformation))
```

Figure 3.4: A sample abstraction rule.

### 3.1.4 Abstraction

From a representational point of view, abstraction is a way of reformulating information from one representation into a different, simpler representation and back again. The transformation rules for abstraction are represented similarly to the operators described above. The abstraction rules are represented temporally to allow an abstract proposition to reflect a pattern of specific intervals. For instance, the bending of a pole over an interval $[t, t+15]$ accompanied by the deformation of a bolt over the interval $[t+10, t+15]$ might represent a particular type of failure, say *deformation* failure, that is meaningful to the abstract plan (see Figure 3.4). If the specific state contains an interval of pole-bending over $[15, 50]$ and an interval of bolt-deformation over $[30, 50]$, then the abstraction rule will generate an abstract interval of deformation failure over the interval $[20, 50]$.

An abstraction transformation has preconditions and effects. The preconditions are of the form

```
(parameter-name start-bound end-bound value-test
:assign temp-variable)
```

which is exactly the same as operators. The effects are of the form

```
(parameter-name start-bound-form end-bound-form
start-value-form end-value-form)
```

which should look equally familiar.

When the preconditions are true, the effects describe the transformed representation of the same information (this works equally well for propagating information to more abstract and to more specific levels of abstraction). The difference from operators is in the fact that whereas triggering an operator produces a legal time interval over which the operator could possibly execute (so it produces a time point within that range), an abstraction rule produces a time interval over which the transformation necessarily holds (so it produces a time interval describing the range).

Note that we make no assumptions about the actual form of the transformation, other than that we are able to transform information both from less abstract to more abstract and from more abstract to less abstract levels of description. The actual transformation rules are dependent on the structuring of the abstraction hierarchy and must be appropriate for the operators at the corresponding levels of abstraction. If, for instance, one were to adopt the representational scheme used in ABSTRIPS, then the transformation to a more abstract level would consist of dropping conditions in the final goal—conditions with a criticality ignored by the more abstract level—and of dropping no information in the states (since the states do not change, but only the operators and the goals). The transformation downward would also delete nothing, since the intermediate goals are of the same representation in the abstract level as they are in the specific level, just with some conditions unspecified. In a GPS style of abstraction, the transformations would completely reformulate the information into the representation of the other layer.

## 3.2 Planning

Planning is the process of finding a set of operators that will lead from the current state to a state in which the goals are achieved, or at least are expected to be achieved. For plans at multiple levels of abstraction, these plans are built at each level, with information flowing between adjacent levels. For a planner under time constraints, these plans may not in fact achieve all the goals or may achieve some of them suboptimally, since there may not be time to complete the planning process.

We first detail what happens within one level of abstraction, in terms of triggering operators and adding them to the plan, and then step back to see how the different levels

```
preconditions:
((x (:ge 0) (:le 10) (= value 3)))
effects:
((x '(:gt 10) '(:le 20) 4 4))
```

Figure 3.5: A simple operator that adds one to the parameter $x$.

of abstraction interact to form the overall plan.

### 3.2.1 Planning at one level of abstraction

Planning within one level of abstraction involves finding a legal set of operators for each state, and adding those operators to the plan in an order most likely to lead quickly to the goal. We describe each of the components of that process in turn. First, we describe how a legal operator is found. Then we describe how it is added to the plan. Finally, we discuss the control used to choose the best operators to add to the plan.

#### Triggering operators

Each interval added to a state may allow an operator to execute in that state. It may be enough by itself to trigger an operator, or it may be just the missing piece of a larger puzzle. For instance, an interval where $x = 3$ over the time interval $[5, 15]$ will clearly trigger the operator in Figure 3.5. But suppose an interval instead asserts $x = 3$ over the interval $[9, 10]$, where the state already contains $x = 3$ over the intervals $[5, 9)$ and $(10, 15]$. The new interval would again make it possible to execute the operator, but in this case it is only partly responsible for triggering the operator. Therefore, each operator must be considered over the range of possible times that the new interval could assist with, not just be entirely responsible for itself.

Given an interval over the time interval $[t_1, t_2]$ and a corresponding operator precondition over the relative time interval $[t_3^r, t_4^r]$, the time interval that must be considered for the operator execution time is $[t_1 - t_4^r, t_2 - t_3^r]$. We mean by operator execution time the time point $t$ with respect to which all of the conditions of the operator will hold, that is, for each

```
preconditions:
((x (:ge 0) (:le 10) (numberp value) :assign temp-x)
(y (:ge 5) (:le 15) (numberp value) :assign temp-y)
(z (:ge 10) (:le 20) (and (numberp value) (= value (+ temp-x
   temp-y)))))
effects:
((z '(:ge 25) '(:le 35) (+ temp-x temp-y 2) (+ temp-x temp-y 2)))
```

Figure 3.6: An operator for illustrating operator triggering. It triggers when a length-10 interval of $z$ is the sum of $x$ 10 time units earlier and $y$ 5 time units earlier.

condition $C$ with a relative time interval $[t^r_s, t^r_e]$, $C$ is true over the time interval $[t+t^r_s, t+t^r_e]$. Thus the operator execution time range is a time point whose value is constrained to be within the time interval $[t_1 - t^r_4, t_2 - t^r_3]$. Given the interval over $[t_1, t_2]$ and the precondition over $[t^r_3, t^r_4]$, note that if the start bound of the interval is open, or either of the bounds of the precondition is open, the resulting operator execution time range is open at the left. Similarly, if the end bound of the interval is open, or either of the precondition bounds is open, the resulting operator execution time range is open at the right.

This operator execution time range represents the broadest possible range of times for operator execution. Once this range of times is determined, then the preconditions of the operator are checked to narrow the time range down to one or more that satisfy the conditions. Each condition is checked in turn. If the current estimate of the operator execution time range is $[t_1, t_2]$ and the precondition relative time interval is $[t^r_3, t^r_4]$, then all time intervals from $[t_1 + t^r_3, t_1 + t^r_4]$ through $[t_2 + t^r_3, t_2 + t^r_4]$ are checked to see whether they satisfy the value test. This is done by collecting the values over the time interval $[t_1 + t^r_3, t_2 + t^r_4]$, separating off the sub-intervals in which the value test is satisfied, and then returning those sub-intervals that are long enough to contain the time interval $[c + t^r_3, c + t^r_4]$ for some $c$. Note that this may return a set of sub-intervals, and in this case the operator triggering procedure splits and tries to instantiate the operator further for each of the sub-intervals.

An example may make this clearer. Consider the operator in Figure 3.6. Suppose the current state contains the following intervals:

$x = 5$ `over [0,20]`

$x = 3$ `over [25,40]`

$y = 8$ `over [3,13)`

$z = 13$ `over [0,25]`

$z = 11$ `over [26,49]`

Now suppose a new interval is added, either from expected plan effects or from the environment, asserting $y = 8$ over [13,41]. The steps to trigger the operator given that interval are the following:

1. The bounds on possible execution times for the operator are determined from the new interval. By the formula above, the operator execution time range is constrained to the time interval [-2,36].

2. The time range [-2,36] is now checked against the first precondition. The precondition relative time interval [0,10] suggests a time interval of [-2,46] in which to find time intervals where $x$ is a number. There are two such time intervals, [0,20] and [25,40], which, when transformed back via the precondition relative time interval [0,10], split the execution time range into two ranges: [0,10] and [25,30]. The variable temp-x is bound to 5 for the first range and to 3 for the second range.

3. The execution time range [0,10] is now checked against the second condition. The precondition relative time interval [5,15] suggests a time interval of [5,25] in which to find time intervals where $y$ is a number. [3,41] is the interval where $y$ is a number, and the portion of this within the checked time interval is the time interval [5,25], which, when transformed back via the precondition relative time interval [5,15], produces the execution time range [0,10], unchanged. In addition, the variable temp-y is bound to 8.

4. Finally, the time range [0,10] is checked against the third condition. The precondition relative time interval [10,20] suggests a time interval of [10,30] in which to find time

intervals where $z = x+y = 5+8 = 13$. The condition holds over the interval [0,25], and the portion of that within the checked time interval is the time interval [10,25], which, when transformed back via the precondition relative time interval [10,20], produces the execution time range [0,5]. This is one final execution time range.

5. The execution time range [25,30] is now checked against the second condition. The precondition relative time interval [5,15] suggests a time interval of [30,45] in which to find time intervals where $y$ is a number. [3,41] is the interval where $y$ is a number, and the portion of this within the checked time interval is the time interval [30,41], which, when transformed back via the precondition relative time interval [5,15], produces the execution time range [25,26]. In addition, the variable temp-y is bound to 8.

6. Finally, the time range [25,26] is checked against the third condition. The precondition interval [10,20] suggests a time interval of [35,46] in which to find time intervals where $z = x + y = 3 + 8 = 11$. The condition holds over the interval [26,49], and the portion of that within the checked time interval is the time interval [35,46], which, when transformed back via the precondition interval [10,20], produces the execution time range [25,26], which is another final execution time range.

The result of this procedure is the determination that the operator can be legally executed in the time interval [0,5] or the time interval [25,26].

The procedure is similar with open and half-open intervals, the only change being that since the transformations are not necessarily invertible, the retransformed execution time range must be checked to make sure it still lies within the original time range. For instance, the time range (0,5] when combined with a precondition interval (0,2], produces a time interval of (0,7]. But the time range [0,7] combined with a precondition interval (0,2] produces the identical time interval of (0,7]. So when the reverse transformation is done, the algorithm first transforms the (0,7] back to the more generous [0,5], and then clips to the original time range if necessary. This ensures that the time ranges are as inclusive as possible without generating incorrect triggering ranges.

**Adding operators to the plan**

The set of operators that trigger within a state is the set of operators that are applicable in that state. Once an applicable operator has been found for a state, the agent may add it to the plan to generate further states. Given an operator execution time range $t \in [t_1, t_2]$ and an operator effect with a relative time interval $[t_3^r, t_4^r]$, a new world is produced based on this operator. Since the operator's execution within the time range $t$ is hypothesized by the agent, it is axiomatic in the new world and is the founder of that world. The hypothetical world additionally contains the expected effect of the operator over the time interval $[t + t_3^r, t + t_4^r]$.

The set of operators legally executed in a state is restricted to those dependent on some interval of that state, and not dependent solely on intervals of preceding states. In other words, given two independent operators $o_1$ and $o_2$ executable in state $s_1$, then if $o_1$, executed in $s_1$, produces state $s_2$, then operator $o_2$ is not executable in $s_2$. This avoids the problem in simple state-based planners of having to consider all possible total orders of independent operators. But to get the same functionality, our approach allows executing a set of operators in a state. In our example, our approach allows executing $o_1$, $o_2$, or the set $\{o_1, o_2\}$. This is also potentially a large set of possibilities, but it is of order $O(2^n)$ instead of $O(n!)$, and it more accurately reflects the independence of the operators.

Allowing sets of operators introduces some additional complexity into the problem of triggering, because of possible interactions among the operators in the set. For instance, two operators may both use the same resource, so they must have their execution time ranges adjusted so that they cannot be executed in a way that would cause a resource conflict. Also, if one operator's effects would affect another operator's preconditions, then the execution time ranges must be modified to avoid those times that would cause a conflict.

Operators may be independent even when they have potential resource conflicts or precondition-effect interactions. Note first of all that all operators legal in a state are triggered independently of all other operators' effects (in fact, the operators' effects do not appear within this state, but rather in subsequent hypothetical states). And second, note that we are restricting the execution time ranges of operators within the set precisely so that there are no resource conflicts or precondition-effect interactions. The operators' remaining execution time ranges are such that the operators are truly independent when executed within those time ranges. Note that independence does not imply concurrency. An operator executed at time 0 may be independent of an operator executed at time 100, so we may include them in a set of operators. All independence means is that there is no cause-effect relation between any pair of operators within the set.

To restrict the operators within a set so that they are truly independent, the planning agent must, as we have said, restrict the execution time ranges so that there are no resource conflicts and no precondition-effect interactions. For every pair of operators within the set of operators, the execution time range of the two operators in the set is restricted to avoid these conflicts, and that restricted time range is used in further pairwise comparisons.

To restrict the execution time range of a pair of operators $o_1$ and $o_2$ using resources, constraints are constructed based on the operators' shared resources. Suppose for such a resource, operator $o_1$ has a resource relative time interval $[t^r_{11}, t^r_{12}]$ and an execution time range $p_1$. Suppose similarly that operator $o_2$ has a resource relative time interval for the same resource of $[t^r_{21}, t^r_{22}]$ and an execution time range $p_2$. Then $p_1$ and $p_2$ are restricted by adding the constraint:

$$[p_1 + t^r_{11}, p_1 + t^r_{12}] \cap [p_2 + t^r_{21}, p_2 + t^r_{22}] = \emptyset$$

To restrict the execution time range of a pair of operators $o_1$ and $o_2$ so that their preconditions and effects do not interact, constraints are constructed based on parameters that are mentioned in the effects of one operator and the preconditions of another. Suppose that $o_1$ has an effect relative time interval $[t^r_{11}, t^r_{12}]$ (including persistence) and an execution time range $p_1$. Suppose similarly that operator $o_2$ has a precondition relative time interval for the same parameter of $[t^r_{21}, t^r_{22}]$ and an execution time range $p_2$. Then $p_1$ and $p_2$ are restricted by adding the constraint:

$$[p_1 + t^r_{11}, p_1 + t^r_{12}] \cap [p_2 + t^r_{21}, p_2 + t^r_{22}] = \emptyset .$$

The constraints based on resources and precondition-effect pairs are the most taxing for a temporal representation. We discuss alternative ways of approaching the problem with a simpler representation when we discuss the implementation in Chapter 4.

### Control within one level of abstraction

Even within a single level of abstraction, the search space for planning is still prohibitively large. That is, after all, why abstraction is needed in the first place. Once intermediate goals are added from more abstract levels of abstraction, the search space becomes more manageable, but it is still large, and plans that achieve the intermediate goals should be favored over ones that do not. This fits into the model of best-first search, so that is used to control the search within a single level of abstraction. The default evaluation function computes the extent to which the current plan expectations achieve all the goals. States that more completely achieve the goals are favored over states that achieve them to a lesser extent. For finite goals and expectations, the measure is simply the percentage of the goal interval where the expectation agrees/disagrees with the goal. For 0-length goals, the measure is all-or-none. For infinite goals, the theoretically correct measure is again all-or-none, depending on whether the expectation is infinite and completely covers the goal. But potentially troublesome cases appear when both are infinite but the expectation covers all but a finite amount of the goal (this looks the same to the evaluation as an expectation that actually covers the entire goal), or when the expectation covers a large, but finite amount of the goal (this is indistinguishable from an expectation that achieves none of the goal). The practical, although not as clean solution to this problem, is to set an upper bound on the length of an interval for the purposes of evaluation, so there will be a small, but measurable difference in the cases described above, while preserving the major differences that are used in most cases.

A penalty may be placed on states in which goals cannot be achieved. Since we do not know how to weight this appropriately in relation to the evaluation above, we do not include this in the default evaluation function. But we consider it a potentially useful heuristic, so we describe it here. A goal cannot be achieved over a time interval in a state when all of the founders of the state start after the end of the time interval. As long as one founder starts earlier than the end of the time interval, there may be a sequence of operators that achieves the goal (see Section 3.1.2 for necessary conditions on the operators for this to be true in all cases).

### Commitment

The expansion of operators and hypothetical states may provide useful information about what effects we can expect from different courses of action. But without committing to these plans, the agent will not take any action, since none of these operators or effects exist in the "current" state of the planner. Remember that the plans may not be complete, but may represent the best partial set of operators known for making progress towards the goals.

The agent decides to commit to a plan under a number of different conditions. It may be running short on the time available until its best plan will need to be executed. It may find that a plan is good enough with respect to the goals that it is willing to ignore other alternatives and commit to that plan. It may find that one plan, although not outstanding on its own, far outshines the competing plans, so that work on the other plans would be unproductive.

The agent will commit to any portion of a plan that satisfies the conditions for commitment. The agent continues extending the plan after committing to a plan prefix, but that prefix is considered fixed.

By committing to a plan, the agent effectively reduces its planning search space by pruning alternative branches from the search space. All of the operators of the chosen plan are added into the current state, along with their expectations. At the executable level, any actions produced by the operators are then scheduled for execution by an independent daemon—a simple sort of execution module—that compares the current time versus the intended execution time of each action, and sends the actions to the appropriate effectors when their time has come.

For the agent to commit to a plan, that plan must be the best one at that level of abstraction. Although the agent may consider many alternative plans while expanding its search space, it always keeps track of the best plan known so far. As the plan is expanded, the agent compares the expanding plan against its record of the best plan, and updates the best plan as necessary to maintain current and correct information.

This model of commitment paints the picture of an agent that searches through a space of hypothetical operators and states to find which operators it should add to the current plan. In traditional planning this would be restricting the possible plan modifications to operator additions. It also closely resembles the basic approach of the SOAR architecture [Laird *et al.*, 1987], which performs automatic subgoaling to choose which of a set of operators to execute in its current state, with the search continuing from that state. Our framework can commit to multiple steps at a time, which resembles SOAR with its chunking mechanism.

### 3.2.2 Planning at multiple levels of abstraction

Although the planning techniques described above form an interesting method in their own right, such a method is, as are all single-level planning methods, too inefficient to solve complex problems in any reasonable amount of time. Certainly we cannot expect the techniques to provide by themselves a real-time planner that will be of much use, since they face a daunting search space both for initial planning and for recovering from errors or unexpected changes in the world. To allow the planner to operate effectively in the real world, we add in abstraction to help both in building the plan and in revising the plan as more information is received. In this section we concentrate on the initial plan construction. We discuss replanning in Section 3.3.

#### Propagating world information

Information about the outside world is received by sensors and posted in the current world at the appropriate level of abstraction. Sensors are normally associated with the lowest, executable level of abstraction, but there is no prohibition in our model against sensors at arbitrary levels of abstraction.

The sensor information is posted in the current world on the "occurred" timeline. This

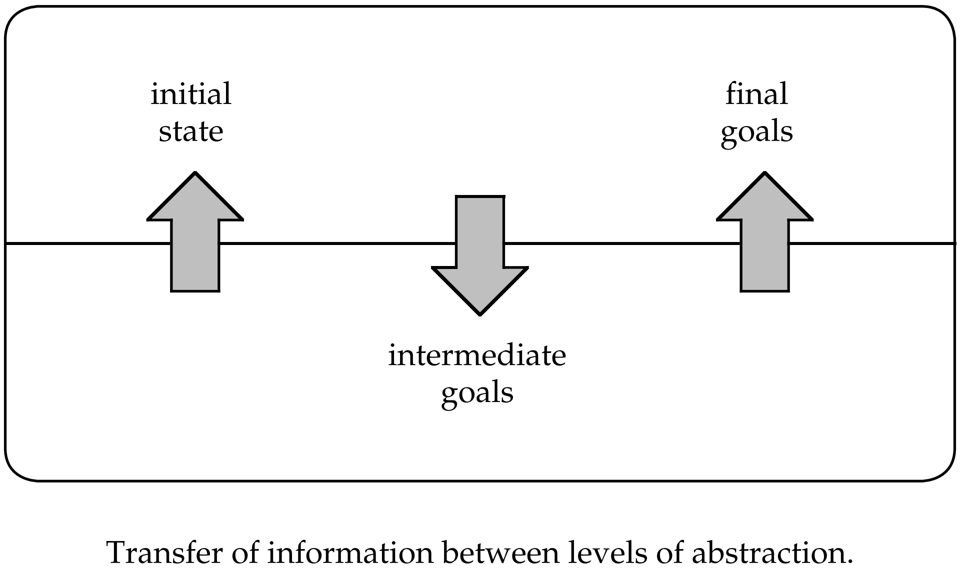


Figure 3.7: Transfer of information between adjacent levels of abstraction.

information is checked against the current expectations (we discuss in Section 3.3 how this affects the planning process), and it is also propagated to higher levels of abstraction to ensure that their model of the world is correct and that their plans are useful for the currently known state of the world (see Figure 3.7).

Recall from Section 3.1 that information is propagated across levels of abstraction using abstraction rules, which resemble operators. As information arrives about one level of abstraction, the abstraction rules are used to transform the information and post it in the current state of the next higher level of abstraction.

### Propagating initial goal information

The initial goals may be specified at multiple levels of abstraction. Since the goals are part of the state, represented as intended intervals, they can be treated similarly to world information. As with world information, the information about the initial goals is propagated throughout the abstraction levels to ensure that the abstract plans will be solving the appropriate (albeit more abstract) problem. The process to abstract the initial goals is precisely the same as for propagating world information: abstraction rules transform the

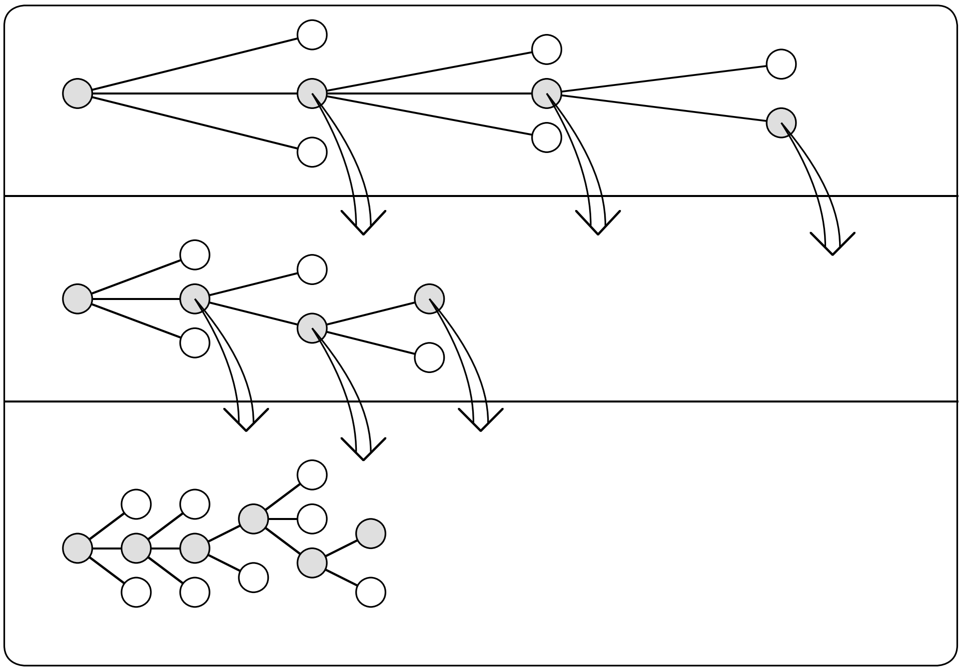

Figure 3.8: Operator effects in an abstract plan are propagated to create intermediate goals at the next lower level of abstraction.

goals and post them in higher levels of abstraction, from where they are propagated further.

### Propagating intermediate goal information

When an operator is added to the best plan at any but the lowest level of abstraction, the operator and its effects are added to a hypothetical state at that level. Since the system has no way to directly execute the operator—unlike the executable level, where actions may be sent to the effectors—it must figure out how to actually achieve those effects. So the effects are propagated to the next lower level of abstraction as goals to be achieved (see Figure 3.8). This type of goal propagation can also be found in Knoblock's thesis [Knoblock, 1991]. One can view this technique as posting generic actions which are then instantiated by the lower levels. For instance, a higher level action might be to travel from one point to another, leaving it for the lower level to choose the most appropriate form of transportation.

Unlike other approaches, we make no assumptions about a strict ordering among the

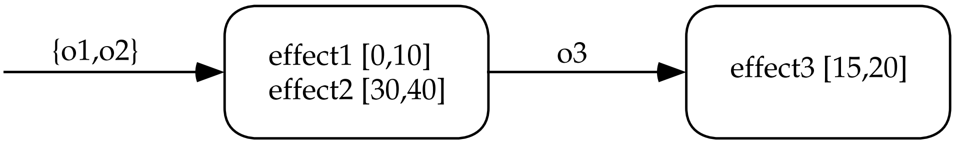


Figure 3.9: Operators may produce effects that are "out of order" with respect to the order of plan construction. The resulting intermediate goals at the lower levels of abstraction will be similarly ordered.

intermediate goals inherited from more abstract planning. Nor do we assume goal independence, which many people who assume strict ordering also assume. For instance, Knoblock's results rest on his assumption of goal independence, which even with strict goal ordering does not hold for even simple classic problems such as Sussman's anomaly or register swapping.

If the agent were to assume goal independence, it could treat goals as "milestones" to be achieved one at a time, ignoring all other goals. This is not reasonable in the case of interacting goals, so for complex, real-world problems this is not a feasible approach.

Even the assumption of a total order of goals is unreasonable when working with temporal information. For instance, at the abstract level one operator may have effects over the time interval [0,10], which triggers another operator that produces effects over the time interval [5,15]. When these goals are propagated to the next more specific level, the intermediate goals overlap.

Furthermore, since sets of independent operators may carry the search from one state to another, there is not even a guarantee that steps at the abstract level will have any particular temporal relation to earlier steps. For instance, an agent may have a set of two operators, $o_1$ producing effects over the time interval [0,10] and $o_2$ producing effects over the time interval [30,40] (see Figure 3.9). Suppose an operator $o_3$ is triggered by the effects of $o_1$ and produces effects over the range [15,20]. Operator $o_3$ is "after" $o_1$ and $o_2$ in the plan-construction process, but its effects actually appear before those of $o_2$. So when we propagate these effects to a lower level of abstraction, the temporal order of the intermediate goals will not mirror the order of the plan-construction process.

So what can a planning agent do with this potential jumble of goals? It takes them as a conjunction of goals and gauges its progress in relation to the entire conjunction. This can be performed more efficiently than it might seem at first glance. All of the goals are intervals on the timeline. If intervals are indexed by parameter and time, then as expectations are added to the timeline, they can be checked against those goal intervals that share the parameter and overlap in time. This restricted check is sufficient for determining goal satisfaction, and is an operation local to a small portion of the total set of timeline intervals. In this way, updating of goal achievement can be computed locally.

**Recovery from subgoal failure**

When an intermediate goal is found to be unachievable, the agent should propagate information about that back up to the level from which the goal was derived. In particular, a failed intermediate goal in our approach corresponds to the possible failure of an operator's expected effect at the more abstract level. The abstract operator's expected effect is overridden by the information from the specific level. Any operators, and more generally any parts of the plan, that depend on that effect will have to be modified or replaced. The basic approach here is similar to how the agent replans given sensor reading mismatches—the details of that and the replanning process appear in Section 3.3.

Note then that a goal failure does not necessitate replanning at the more abstract level. If, despite this failure, the abstract plan is still determined to be the best available, then the abstract plan remains untouched. In this case, the intermediate goal at the lower level is removed, and the planner continues without that goal. Depending on the criticality of the goal to the overall plan, the agent may decide either to ignore the goal or to replan to compensate for its failure. For example, if the agent decides at the abstract level to build a tower on a base of metal poles, but finds that it has no spare metal poles but instead has a base already made of wooden poles, then it may ignore the "metal" goal and continue with its plan, building the rest of the tower on the wooden base.

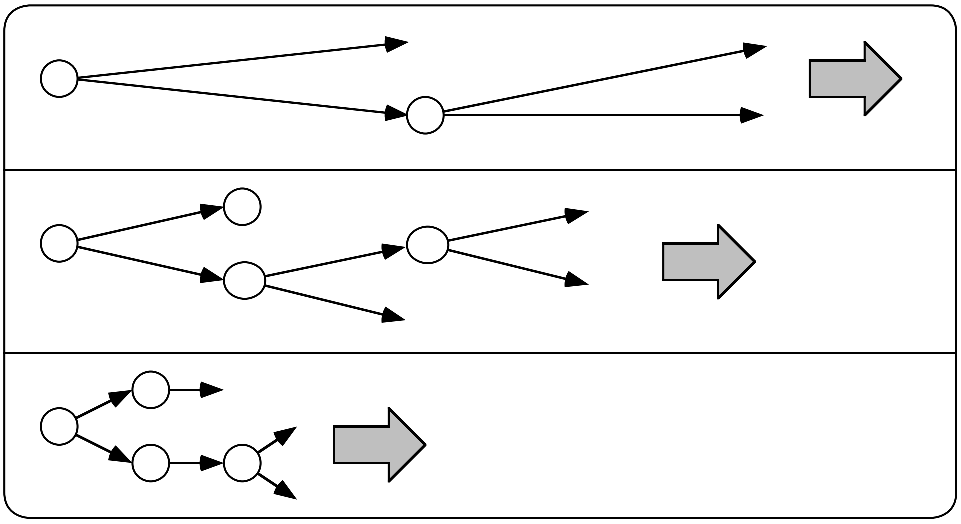

Figure 3.10: Plans may be expanded at any of multiple levels of abstraction.

### Control over multiple levels of abstraction

The agent interleaves the incremental expansion of its plan at multiple levels of abstraction. Because of this, the agent is faced with the task of deciding how much planning to perform at each level at any given time (see Figure 3.10). If all the agent's energy were applied to the most abstract level, it would mimic the behavior of systems such as ABSTRIPS, expanding each level completely before continuing on to the next level of abstraction. The problem with this approach is that it requires that the optimal plan is found at every non-executable level before the first step is planned at the executable level. This is unreasonable in a real-time environment, where the agent may not have the resources to find the optimal plan at any level of abstraction, much less almost every level, before it begins to act.

Suppose instead that the agent took the opposite approach, and planned a minimal amount at each abstraction level, perhaps just enough to stay ahead of the specific level's plan. This also is likely to be suboptimal. The abstract plan has a high chance of error, since it is only minimally expanded and might change substantially when expanded. Also, if the intermediate goals interact, the agent will discover the interactions only when the goals are all present. If the goals are added after the agent has planned a significant amount of the specific plan, the specific plan might well prove less optimal than if the complete set of interacting goals were available while the agent built the specific plan.

The appropriate plan-expansion strategy falls somewhere in between these two extremes. The general problem of formulating an ideal strategy depends on many characteristics of the problem, such as the branching factor of the plan, the variability of the world, and the depth of the plan, so there is no universally optimal approach. In Chapter 5 we describe how to find strategies that are optimal for some important aspects of the problem. Here we discuss the tradeoffs inherent in the choice of a strategy.

Expanding a plan at any level of abstraction is a search process, and as such, may be expensive. Abstract plans often do not become as deep as specific plans, but they still carry a fair overhead. On the other hand, abstract plans provide intermediate goals to the more specific plans, so they reduce the search at that more specific level. If an abstract plan is producing goals that do not reduce the specific search more than the cost of the abstract search (for instance, the goals are too far in the future, or they are too vague to limit the specific possibilities significantly), then it is a net loss in search time to expand the abstract level instead of the specific plan.

Emphasizing one of the points mentioned above, we see that the amount of aid an abstract plan gives to a specific plan depends on the usefulness of its goals. So the approach needs to be sensitive to the ability of the goals generated by the abstract plan to reduce the specific search. If the abstract plan is expanded too far, then its goals are likely to be less useful to the specific plan.

A longer plan is more likely to approximate the optimal plan for a given level of abstraction. In particular, since abstract plans form a framework within which the specific plans are built, longer abstract plans offer a more stable overall plan. However, as shown in the extreme case, a specific plan, or at least a partial plan, must be generated in the amount of time available. In addition, the specific plan may be used to react quickly if the time available is reduced dramatically and the system is forced to execute something. Thus there is a tradeoff between reactivity to changing resources and the stability of the overall plan.

In a similar vein, a longer abstract plan is more likely to give rise to a more optimal overall plan. The longer abstract plan is more likely to be correct, and this is used as a framework around which lower level plans are built. Therefore the overall plan is more likely to be correct when more work is done at abstract levels.

If an abstract plan is too uncertain, then it may be worth postponing further planning work until the agent receives more information from the world that will constrain the abstract plans. Otherwise expanding the plan will offer vanishingly small amounts of useful aid to the lower-level plans.

A planning agent also must not ignore the amount of time available until an action is required. With a short deadline, the agent might not be assured of finding an executable action in time if it ignores the deadline, so this also must be taken into account when deciding how much effort to expend at each level of abstraction. With shorter deadlines, more relative effort must be spent at the executable level than some of the other tradeoffs might suggest.

### Control of plan execution

As the agent commits to steps of its plan, the corresponding executable actions are scheduled for execution. In particular, the action intervals describe a schedule for execution, and this schedule is sent to the effectors. Depending on the sophistication of the effectors, the actions are either sent along with time information when they should be executed, or a daemon within the agent relays them at the appropriate time.

In the extreme time-pressured case, we might reach a deadline before arriving at a single executable action. Although the control strategy should prevent this from happening in all but the most extreme cases, our approach can produce an action quickly if necessary. First, remember that the approach always maintains the complete list of actions that it has triggered for each state. This includes the current state at the executable level. Although the initial goal is likely to be of little use, an obvious first cut would be to find the operator rated highest by the search control with respect to that goal. But given a bit more time, we can follow the following procedure: Find the lowest level of abstraction at which a plan exists. Propagate the effects from that plan to goals at the next lower level of abstraction.

Find the best first step with respect to those goals at the lower level of abstraction. That in turn will produce a set of intermediate goals at a still lower level of abstraction, and the procedure is repeated until the executable level is reached. The rationale behind this is that the abstract levels produce goals that are close enough to the current state to provide meaningful guidance to the lower level plans, while expending a minimum of effort to supply the guidance.

## 3.3 Replanning

An important, but oft-neglected feature of real-world domains is that the world does not always conform to the internal model that the agent maintains. The planning agent is not necessarily the only agent acting on the world, so events may occur that are unexpected by the planner. Some of the predictions made by the planning agent may not be completely accurate in a dynamic environment; in particular, the planned actions might not have the effects that the planning agent has envisioned.

To function in such an environment, the planner must be able to accept unexpected information, incorporate it into its model of the world, and adjust its plan to conform to this new model. Within a single level of abstraction, the new information may force some operators to be removed from the plan, possibly requiring the plan to be revised. Operators may also be triggered by the additional information, adding steps and branches to the plan. The new information may cause replanning at multiple levels of abstraction.

Since our approach is designed to work with partial plans at multiple levels of abstraction, replanning actually uses the same mechanism as planning in the first place. Once the world model and plan have been adjusted to fit the new information from the world, partial plans remain at the various abstraction levels. They are possibly less complete than they were before the information arrived, but they are still just as usable by the planning method (see Figure 3.11).

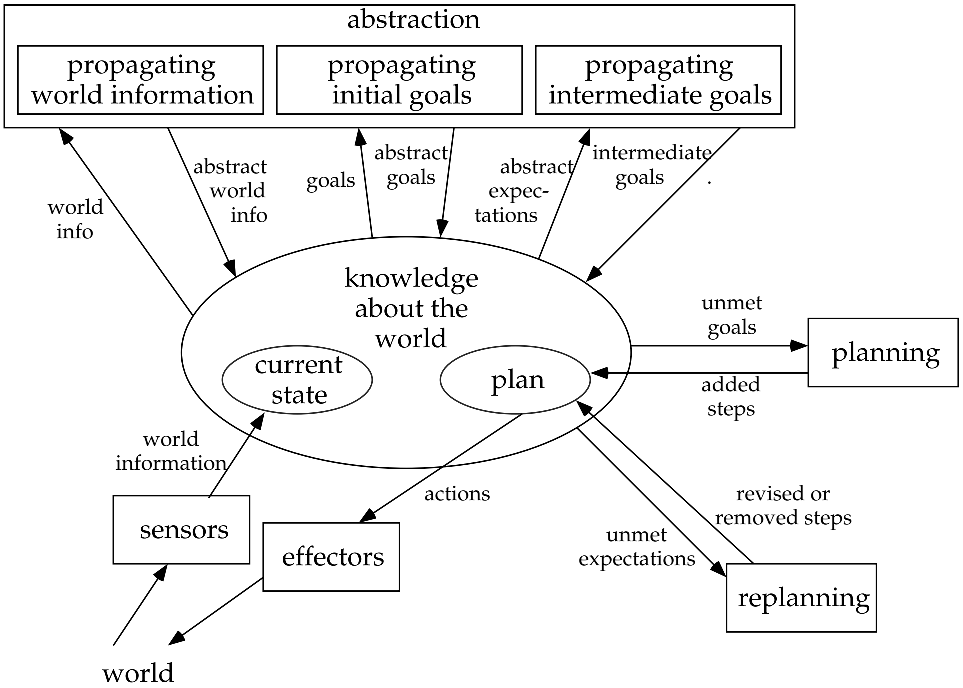


Figure 3.11: Planning may be viewed as a number of processes modifying a global data structure (the plan) and reacting to changes in it.

### 3.3.1 Replanning at a single level of abstraction

When a sensor records an event in the world, an interval is placed on the occurred timeline at the appropriate level of abstraction. That information is then propagated to the expected timeline. If the new information triggers new operators, the operation of the planner is exactly as described in Section 3.2.1 above: the possible range of execution times is computed for the operators, and they are added, singly and in combination with any other operators applicable in that state, to the space of operators that the planner is searching. For further information about the process of triggering operators and adding them to the plan, see Section 3.2.1.

If, however, the new information contradicts information that an existing operator relies on, then the operator is revised. The planner maintains dependency information about which intervals are used to trigger each operator, so when an interval is overridden, it is a simple matter to find the set of affected operators. If the new information only partly obviates some condition of the operator, then the range of possible execution time ranges is narrowed to match the remaining condition that does match, using the same procedure as operator triggering to determine the new range of possible execution times.

Once an operator's execution time range has been changed, the time range of the effects will also change. Note that this is a change to a set of expected intervals. So the same procedure that we just described, revising operators based on changes to expected intervals, now applies to these new intervals. This procedure will iteratively update the plan and propagate the changes throughout the affected portion of the plan.

The agent is left with the portion of the plan that is applicable in the new state of the world, as the agent now knows it. Since the basic planning process takes a partial plan as input and expands it, this new plan is suitable input to the planning process. So the agent can continue expanding the plan from this new state. Basically, we can view the agent as reacting to changes in the plan. It reacts in the same way to its own changes to the plan and changes imposed by the world.

### 3.3.2 Replanning at multiple levels of abstraction

When a plan is being constructed at multiple levels of abstraction, then unexpected changes in the world may affect more than just the particular level of abstraction to which they are reported by the sensors. So the additional problem presents itself of finding the affected plans at all levels of abstraction. Here we can adopt an approach dating back to NOAH [Sacerdoti, 1977], in which the change is abstracted up through the abstraction hierarchy until the abstraction transformation produces data that do not change the world at some level of abstraction.

This behavior of abstracting the change up through the levels of abstraction actually falls out of our existing method without any additional mechanism. We describe in Section 3.2.2 how state information is propagated up through the levels of abstraction. And we describe in Section 3.3.1 how world changes are represented as new state information. Therefore the changes are propagated automatically by the mechanisms for recording changes and propagating state information.

Moreover, the desired behavior at each level of abstraction also falls out automatically from the single-level replanning as described in Section 3.3.1. At each level, the plan is revised to match the new information.

Note that this may have effects at more than a single level. As the plans are revised, the intermediate goals propagated to lower levels of abstraction may change. So as the changes propagate throughout the layers of the plan, the intermediate goals may change also, which may change the idea of the best plan at a level. That may in turn change the goals propagated to the next lower level, and so forth. But the important point of all of this is that this behavior is all covered by the existing planning mechanisms.

## 3.4 Summary

We have described an approach to constructing partial plans simultaneously at multiple levels of abstraction. These plans are built based on the planner's state of information about the past, present, and future. The plans are expanded incrementally as long as the planner has time available. In addition, the planner remains sensitive to the dynamic environment, revising the plan to match new information as it arrives. This process of replanning fits within the model of building the plan initially, so that planning and replanning are integrated together into a seamless process that incrementally molds the plan into a increasingly optimal form.

# Chapter 4

# Implementation

The planning approach in this thesis has been implemented within the BB1 blackboard problem-solving architecture [Hayes-Roth, 1985]. We will discuss the aspects of the architecture that are relevant to the planning system, and then discuss details of the implementation as they augment or differ from the method as discussed in Chapter 3.

## 4.1 BB1

BB1 is a blackboard-based problem-solving architecture. For a complete treatment of BB1, see one of [Hayes-Roth, 1985; Hayes-Roth *et al.*, 1987]. Here we give an overview of the architecture and then present the details that are particularly relevant for the work in this thesis.

BB1 is a reasoning architecture that supports opportunistic reasoning. The blackboard is a global data structure that serves as a communication medium for multiple reasoning components. The reasoning components are implemented as set of *knowledge sources*, which are activated by changes in the contents of the blackboard and execute to produce more changes. Opportunistic reasoning is enabled by this data-driven knowledge-source activation—when something changes on the blackboard, a knowledge source may be activated automatically. Contrast this with a typical programming language, where procedure calls are made in accordance with a fixed control structure, rather than as a direct response to changes in the data. This much of BB1 closely resembles production systems as seen in [Laird *et al.*, 1987; Nilsson, 1992].

An additional feature of BB1 is its explicit and declarative representation of control. Many production-style systems retain a fixed control structure, allowing opportunistic reasoning within this fixed control. But BB1 has a control plan that is declarative and modifiable by reasoning actions. In particular, a knowledge source may have as its results a change in the overall control of the system.

### 4.1.1 The concept hierarchy

All information in BB1 is stored as part of a concept hierarchy, which is similar to a semantic network or conceptual graph [Sowa, 1984]. Objects are part of a type hierarchy, but also have named links to other objects, expressing relationships among them. All objects in the BB1 system, including knowledge sources, control objects, and reasoning data structures, are objects in the hierarchy and may be inspected and reasoned about by the system.

### 4.1.2 The basic architecture

The basic reasoning cycle of BB1 is shown in Figure 4.1. We start at the bottom, with events that record changes in the blackboard. The agenda manager checks each available knowledge source against each event, and all triggered knowledge source instantiations (KSARs) are placed on the *agenda* (along with the unexecuted KSARs from previous cycles). The scheduler then uses the information from the control plan to choose the best KSAR to execute. The executor executes the KSAR, and its actions create events that start the cycle over again. The reasoning continues as long as there are KSARs to be executed.

The control plan provides global coherence to the reasoning process by favoring KSARs that are relevant for the goals of the reasoning system. Given appropriate control knowledge, the system may notice important KSARs from other reasoning modules and modify the control to execute the new KSARs and follow that line of reasoning.

The architecture is designed to support multiple reasoning components concurrently. The information from each component appears on the blackboard and may be used by other

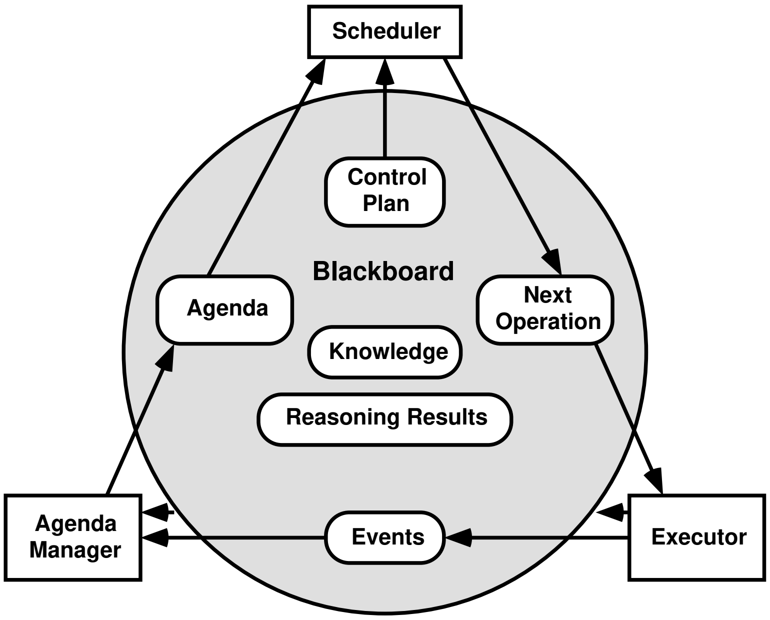


Figure 4.1: The basic BB1 reasoning cycle.

reasoning components. The planner we describe in this thesis may be considered one of many reasoning components within a larger reasoning agent. For instance, the basic persistence module could be supplanted by a component that follows a more knowledge-based approach to prediction. In addition, these other components may modify the blackboard contents and consequently affect the plan much in the way that an external environment might. When we discuss the interactions of the planner with the external environment, we also include in our view any other reasoning modules that might be affecting the plan, or in other words, anything external to the planner itself.

### 4.1.3 Interaction with the environment

BB1 has been extended to interact with an external environment. In Figure 4.2, the basic control structure has been augmented with a communication channel to sensors and effectors. The basic cycle is unchanged, except that now external events may be generated by sensors and used to trigger reasoning operations, and the execution of KSARs may generate external actions in the effectors. The external communication with the outside world

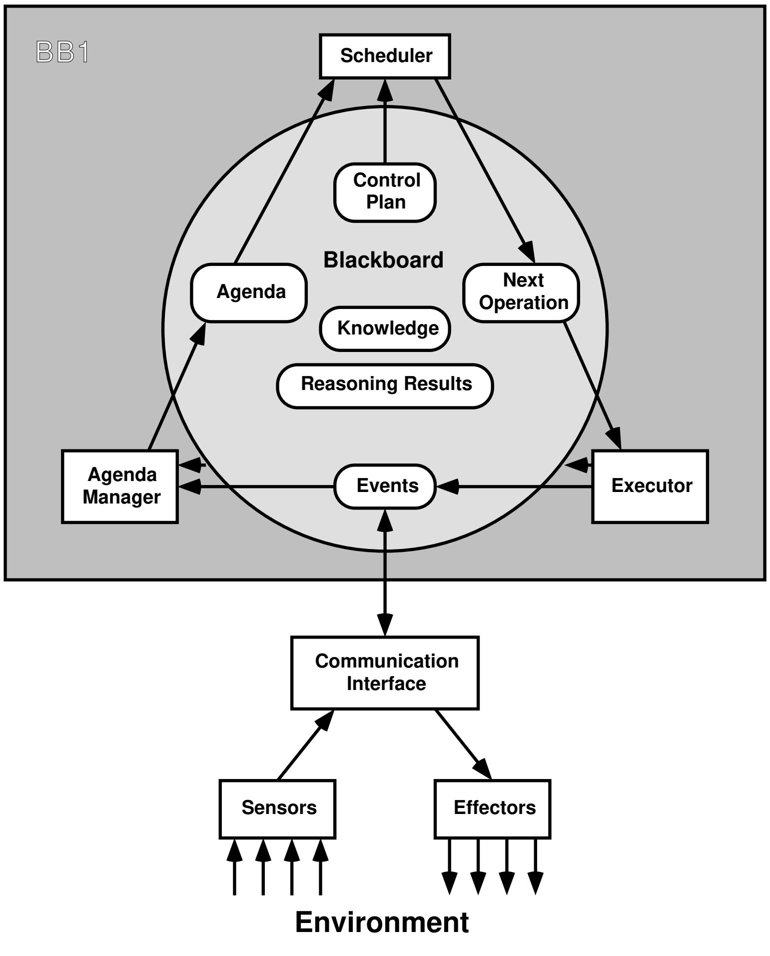


Figure 4.2: The BB1 cycle augmented to communicate with an external environment.

is asynchronous and controlled by the reasoning system (BB1) as well as by the communication channel to avoid overwhelming the system with data [Washington and Hayes-Roth, 1989].

The data that do arrive at the system can trigger KSARs, so that with this added communication channel, reasoning can be opportunistic with respect to the external environment. The interaction of the environment with internal reasoning is critical to our approach, both in the initial planning and in changing the plan in ways that require replanning.

### 4.1.4 BB1 languages

The basic BB1 knowledge-source language has been augmented with a higher-level description language that is designed to enhance understandability and also internal control of the reasoning process. Instead of arbitrary lisp-syntax knowledge sources, the conditions and actions of knowledge sources are expressed in a simple template grammar, where the objects in the grammar are themselves part of the concept hierarchy. This way a knowledge source can produce a specific effect in the blackboard that will trigger a knowledge source with a more general condition. In planning, for instance, changing the start bound of an operator may trigger a knowledge source that looks for any change to an operator.

We have extended the basic BB1 languages to implement our method, allowing multiple language actions in a single knowledge source, and allowing hierarchical implementation of an action (where a single language action may be executed by executing many subsidiary language actions). This was necessary to implement the sometimes complex plan-modification operators.

## 4.2 Timeline

The temporal representation has been realized as the Timeline component. The Timeline component is a program of 18000 lines of Common Lisp and BB1 code. The Timeline component's basic algorithms and data structures reside in Lisp for efficiency purposes.

The Timeline maintains and updates a parallel model in BB1 for reasoning purposes. This separation allows it to perform its innermost data structure handling at least an order of magnitude faster than BB1 operations. The results of the Lisp manipulations are then propagated to BB1 once they have been optimized to minimize the BB1 operations. The BB1 operations cause changes in the reasoning that the system performs, so in this way we can use the power of BB1 while maintaining the speed of a standard programming language.

### 4.2.1 Data structures

The approach as described in Chapter 3 represents world information using the basic unit of an *interval*, which represents a proposition over a time interval. The implementation breaks this down further, to sub-intervals, interval segments, and time points.

Underneath everything lies a network of time points. This network maintains a strictly-ordered timeline of time points for each variable in the system. This loses the complete generality that we need to express the general constraints of Section 3.1.1, but it limits the computational complexity of reasoning about time points. In retrospect, the difference in speed between Lisp and BB1 would suggest that this computational complexity is probably acceptable, so we might prefer a richer temporal representation.

The time points serve as endpoints of the intervals that represent information in the timeline. The intervals themselves are subdivided into smaller pieces for reasoning purposes. If multiple intervals intersect and disagree over a time interval, the system has to choose a single interval to be true over that time interval for the sake of consistency. The interval that is chosen to be true by the system is called *active* within that time interval. In Figure 4.3, if C overrides B, which overrides A, then in the time interval [10,15], A and B intersect, and B is the active interval.

If the currently active interval is removed from the timeline, one of the other intervals will become active over the time interval. This leads to a stack-like behavior, where the topmost in the stack is active within the time interval. To maintain that behavior, the timeline is subdivided into *interval segments*. An interval segment is a time interval over which a set of intervals intersects. In Figure 4.3, the time interval [0,10] is an interval

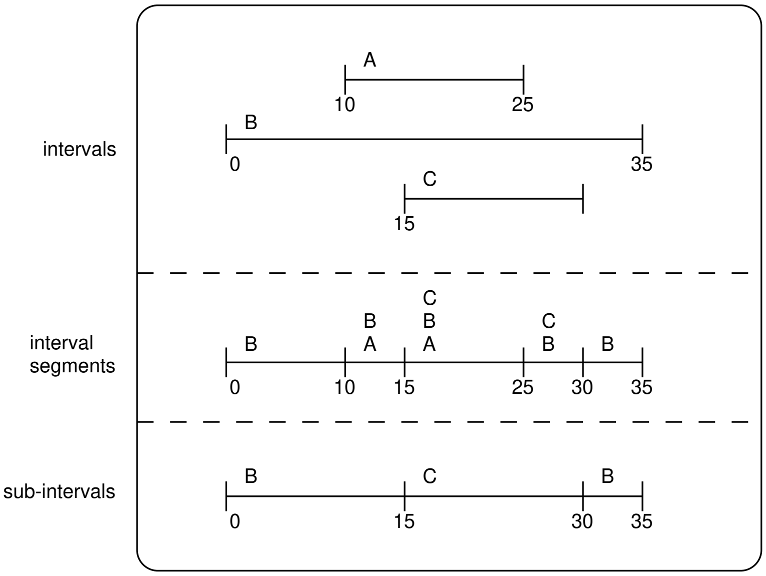


Figure 4.3: The representation of intervals in the implementation. Intervals are subdivided into interval segments and sub-intervals. In this example, interval C overrides B, which overrides A.

segment, as are [10,15], [15,25], etc. Each interval segment maintains a stack of intervals that intersect over the interval segment. Adjacent interval segments have different sets of intervals; in other words, an interval segment is the maximal time interval included in the same set of intervals.

The planner and other reasoning components focus on the active interval for each interval segment. For reasoning purposes, the set of inactive intervals is uninteresting. Therefore the timeline maintains a set of *sub-intervals*, where each sub-interval is a time interval that contains a set of interval segments with the same active interval. In Figure 4.3, the time interval [0,15] is a sub-interval over which B is active, and it contains the interval segments [0,10] and [10,15]. Adjacent sub-intervals have different active intervals; that is, a sub-interval is the maximal time interval over which an interval is active.

Note that there is a many-to-one mapping from interval segments to sub-intervals. Since a sub-interval is a maximal contiguous set of interval segments with the same active interval, each interval segment will belong to one sub-interval.

Note also that there is a many-to-one mapping from sub-intervals to intervals. An interval may be active over multiple time intervals because of "holes" caused by other intervals that are active within the time intervals of the holes.

As intervals on the blackboard are added, modified, and deleted, the underlying data structures are updated automatically to reflect their changes. BB1 data structures are maintained for sub-intervals and intervals, so the reasoning system may react to changes to either of those levels of description.

### 4.2.2 BB1 blackboard structures

The BB1 data structures are maintained in parallel with the Lisp data structures. As the Lisp data structures are updated, models of the BB1 objects are updated simultaneously. Once all of the modifications have been made to the Lisp structures, the BB1 object additions, modifications, and deletions are propagated to the BB1 blackboard. This is primarily an efficiency consideration, but there are also other reasons as well. Consider splitting a sub-interval into two sub-intervals, where one will point to a new interval. Using normal programming techniques, the sub-interval will be duplicated, and then the pointers from the second sub-interval will be changed. If the system were to propagate the changes as they occurred, there would be a period where the blackboard information was inconsistent. By propagating the changes as a whole, the system can make a single addition of a now-consistent object to the blackboard.

Even some of the efficiency gains will be substantial. Consider the effect of adding an interval, then deleting it. Perhaps that interval will be deemed important enough to completely change the control plan of the BB1 system. Suppose that the interval was merely an intermediate step in some other interval operation. Then by storing the results of the data structure manipulations, the BB1 object would end up never being created in the first place. By avoiding this object creation, the system can also avoid the potentially expensive and useless changes to its plan that the object would cause.

### 4.2.3 Hypothetical worlds

As described in Chapter 3, hypothetical worlds are represented as deltas from their supporting worlds. Therefore, operations that look up information within hypothetical worlds must return information not only about the intervals that are found, but also the time intervals within which these intervals are used in the hypothetical world (since an interval in the supporting world may be masked by an interval in the hypothetical world). The lookup routines actually return a set of sub-interval objects (since intervals may not be active over a single, contiguous time interval, but sub-intervals will be). These sub-intervals are paired with time-intervals over which they contribute to the lookup.

The relations among worlds, world-founders, timelines, and intervals are all represented in the Lisp data structures and mirrored in the BB1 object network. This way reasoning may be performed about the BB1 objects and links, while retaining the efficiency of working at the Lisp level for internal timeline processing.

## 4.3 Planning

Planning is much as described in Chapter 3. The planning component comprises an additional 11000 lines of Common Lisp and BB1 code on top of the Timeline code. Intervals added to the timeline trigger planning knowledge sources. These knowledge sources are instantiated as KSARs for particular operators and triggering intervals. The KSARs' actions will add those operators to the plan.

The parameters that appear in an operator's conditions are also objects in the BB1 object network. Links are strung from the operator (which is also a BB1 object) to the parameter. As an interval is added to the timeline, the interval's parameter is checked to see whether any operators could be triggered by the new interval. That condition will trigger the knowledge source that does the actual planning work of triggering the planning operator. If the planning operator's conditions are satisfied by the new interval and the other information on the timeline, an *operator possibility* object is added, which records the variable bindings generated by the operator triggering and the time intervals over which the operator could be executed.

As multiple operator possibilities appear for a world, they will trigger a knowledge source for combining the possibilities into operator sets. The combination KSARs then check for resource and precondition-action conflicts as described in Chapter 3. Because of the simpler temporal representation, the resulting operator combinations may end up as multiple worlds, each of which represents one part of the complex constraints that resolve the conflict. For instance, given a constraint that $x \neq y$ for two time points $x$ and $y$, two worlds will be created, one in which $x > y$, and one in which $x < y$.

## 4.4 Replanning

Replanning, as described in Chapter 3, is a natural extension of our planning method. Indeed, the very events that cause new planning operators to be added to the plan may also cause the loss of information on which a plan step depends. In particular, as operators are added to the plan, links are formed from the plan steps to the timeline intervals on which they depend. As these intervals are modified or deleted, the plan steps are rechecked and revised as necessary to maintain consistency with the new information.

If an interval changes enough that an operator is no longer applicable in the world in which it appears, that operator is removed from the plan, and its world and all worlds that depend on it are removed as well. The planning method then expands the newly-shrunken plan.

The same interval may also trigger new operators that were not applicable before, or expand their potential execution interval. This also changes the possible plan expansions available to the planner, which continues on from that situation.

# Chapter 5

# Analytical results

We have investigated the effects of abstraction on planning search analytically. Korf [Korf, 1987] shows that abstraction can reduce an exponential search problem to linear in a macro-operator network. The macro-operator model differs significantly from our own. The states themselves are no more abstract at higher levels of abstraction; rather, an abstract space contains a subset of the states at lower levels of abstraction. The only states that can be abstracted are those states shared with the next more abstract level, so there is a search within a level of abstraction to find a state that can be abstracted. And the abstract search completely obviates the need for lower-level search.

Knoblock [Knoblock, 1991] uses a more traditional planning search space. He shows that abstraction can reduce an exponential planning search to linear with appropriate abstractions, but that result only holds under a set of strong assumptions. We take a different and weaker set of assumptions that are more reasonable for a temporally-represented domain. We cannot achieve the exponential-to-linear gains under our model, but we can show the following:

- how much abstraction is appropriate when the environment may change;
- the optimal amount of abstraction for any given problem;
- the change in the optimal amount of abstraction as the plan lengthens.

First we review the assumptions that underlie the analysis and discuss how the planning process operates under these assumptions. Then we address each of the three points above.

## 5.1 Basic assumptions

We assume blind search at each level of abstraction, with a constant branching factor $b$ across all levels. As described in Chapter 3, the states of an abstract search are translated into goals for the next more specific level of search. The reason for this is that the agent is trying to implement the abstract plan at the lower level by achieving each of the states that the abstract plan reaches.

Our approach to planning only enforces dependencies among subsequent steps in a plan at a given level of abstraction. If there is a sequence of states $s_0, s_1, s_2$ in the plan, where a set of operators $O_1 = \{o_{11}, o_{12}, ...o_{1m}\}$ produces $s_1$ from $s_0$, and a set of operators $O_2 = \{o_{21}, o_{22}, \ldots, o_{2n}\}$ produces $s_2$ from $s_1$, then each $o_{2i}$ depends on the results of some $o_{1j}$. Each such $o_{2i}$ does not have to strictly follow the effects of the $o_{1j}$, but may overlap. Hence the effects of an operator $o_{2i}$ may overlap with the effects of the operator $o_{1j}$ on which it depends, and may be completely temporally independent of all other operators in the set $O_1$. When these steps are propagated to become goals at a lower level of abstraction, subsequent intermediate goals may overlap temporally or even be "out of sequence."

Because of this lack of strict ordering among intermediate goals, the search at the specific level cannot use the goals as "milestones" in the sense of [Knoblock, 1991]. Knoblock assumes that the search can achieve one intermediate goal completely independently from all previous and subsequent goals. This is a problem even without our lack of strict temporal sequence (for instance, the Sussman anomaly or register swapping are examples where this assumption does not hold), but in our domain it is completely untenable.

Instead of viewing the set of intermediate goals as milestones, we see the goals, taken as a set, as constraining the search at the lower level of abstraction. We express this by taking the probability of an arbitrary goal ruling out an arbitrary state (i.e., if the system reaches the state, the goal will never be achieved) as $p$. For $g$ goals, the probability that any single state is not ruled out by a goal is $(1-p)^g$. For a search of branching factor $b$, the fringe at depth 1 will be $b(1-p)^g$, the fringe at depth 2 will be $((b(1-p)^g)b)(1-p)^g = (b(1-p)^g)^2$, and in general the fringe at depth $i$ will be $(b(1-p)^g)^i$. Therefore, where a completely blind search of branching factor $b$ and search depth $d$ expands

$$\frac{b^{d+1}-1}{b-1}$$

nodes, adding $g$ goals will reduce the search space to

$$\frac{(b(1-p)^g)^{d+1}-1}{b(1-p)^g-1}$$

nodes.

Given $n$ levels of abstraction, where level 1 is the most specific and level $n$ is the most abstract, and plans expanded to depth $d_i$ at each level $i$, we assume that there is some $k \geq 1$ such that $d_1 = d_2k = \ldots = d_nk^{n-1}$. That is, for any two adjacent levels of abstraction, the ratio of their plan lengths is $k$. Note that for any specific domain there would also be an upper bound on $k$ corresponding to the ratio of steps at one level to steps at the next more abstract level; otherwise the abstract search could lag behind the specific search. Our analyses are valid with or without this upper bound.

When an abstract search propagates a goal to a specific level, the existing search tree at the specific level is first revised using the new goal, then extended from the new (smaller) fringe. This incurs some overhead for revising the existing search tree, but remember that the fringe of a full search tree is of the same order as the entire search tree, so this overhead is balanced by the savings in extending the search. Since the new search will be from the smaller fringe, the savings will be at least as high an order as the revision work.

## 5.2 Abstraction in a changing environment

The problem we address in this section is how much abstraction is desirable when the world can change. We assume a blind search to some depth $d$ at the executable level. Given different abstraction depths and different probabilities of world changes affecting the abstraction levels, we show how much work is lost. We normalize across the different problem situations by analyzing the work lost as a percentage of the total work.

A blind search of branching factor $b$ and search depth $d$ expands

$$\frac{b^{d+1}-1}{b-1}$$

nodes. Recall that we assume that given $n$ abstraction levels, for $n > 1$, where level 1 is the most specific, level $n$ is the most abstract, and the plans are expanded to depths $d_1, \ldots, d_n$, $d_1 = d_2 k = \ldots = d_n k^{n-1}$ for some $k$.

We define $w_i = \frac{b^{d_i+1}-1}{b-1}$ to be the amount of work done at level $i$.

Given a change in the external environment, we denote by $p_i$ the probability that the change will affect level $i$, causing it to be replanned. We assume that $1 > p_1 > p_2 > \ldots > p_n > 0$, which indicates that more abstract plans are more immune to noisy or changing data. We assume strict inequality, discarding the case $p_1 = p_2 = \ldots = p_n$, since in that case we can derive that the percentage of work lost is constant (namely $p_i$ for any $i$) independent of any other factor. The case where the $p_i$s are equal not only makes the analysis unrevealing in terms of how much abstraction is appropriate, but it is also, we believe, an unrealistic assumption.

To compute the amount of work lost, first we note that when level $k$ is affected by a world change, all levels $< k$ will also need to be replanned. So the absolute amount of work expected to be redone for $n$ levels of abstraction can be represented by the value of $W(n)$ for the following recurrence relation:

$$\begin{aligned} W(i) &= p_i \textstyle\sum_{j=1}^{i} w_j + (1-p_i)W(i-1), \text{for } i > 1 \\ W(1) &= p_1 w_1 \end{aligned}$$

If we define $\phi_i = p_i \prod_{j=i+1}^{n}(1-p_j)$for $i < n$ and $\phi_n = p_n$, we can restate $W(n)$ in the form:

$$W(n) = \sum_{i=1}^{n} \phi_i \sum_{j=1}^{i} w_j$$

The total amount of work done is:

$$\sum_{m=1}^{n} w_m$$

Hence the percentage of work lost is the ratio of the two, or:

$$\frac{\sum_{i=1}^{n} \phi_i \sum_{j=1}^{i} w_j}{\sum_{m=1}^{n} w_m}$$

To determine how different amounts of abstract search affect the amount of work lost, we observe the result of varying $k$, the ratio of search depths between adjacent levels of abstraction. In particular, consider taking $k$ and $k'$, where $k > k'$. Consider the case where the total amount of work done in the two cases is equal, so that

$$\sum_{i=1}^{n} w_i = \sum_{j=1}^{n} w_j'$$

In general, we will use $\theta'$ to refer to the value of $\theta$ when using $k'$ as the ratio between levels of abstraction.

Thus in comparing the percentage of work lost, we only need to compare $W(n)$ to $W'(n)$. In particular, we will determine the sign of $W(n) - W'(n)$.

Note that

$$\begin{aligned} W(n) &= \textstyle\sum_{i=1}^{n} \phi_i \sum_{j=1}^{i} w_j \\ &= \textstyle\sum_{i=1}^{n} w_i \sum_{j=i}^{n} \phi_j \end{aligned}$$

We define $c_i = \sum_{j=i}^{n} \phi_j$ and restate the formula above as

$$W(n) = \sum_{i=1}^{n} w_i c_i$$

Similarly for the case where the ratio is $k'$, we define

$$W'(n) = \sum_{i=1}^{n} w_i' c_i$$

Now recall that $d_1 > \ldots > d_n$ and $d_1' > \ldots > d_n'$. From this it follows that $w_1 > \ldots > w_n$ and $w_1' > \ldots > w_n'$. Furthermore, since $k > k'$, it follows that $w_1 > w_1'$ and $w_n < w_n'$. Therefore, there exists a value $l$ such that $w_i \geq w_i', 1 \leq i \leq l$ and $w_i < w_i', l < i \leq n$.

Also observe that $1 > c_1 > \ldots > c_n > 0$, which follows from the definition of the $c_i$s.

We are trying to determine the sign of $W(n) - W'(n)$:

$$\begin{aligned} W(n) - W'(n) &= (c_1w_1 + \ldots + c_nw_n) - (c_1w_1' + \ldots + c_nw_n') \\ &= c_1(w_1 - w_1') + \ldots + c_n(w_n - w_n') \\ &= [c_1(w_1 - w_1') + \ldots + c_l(w_l - w_l')] - \\ &\quad [c_{l+1}(w_{l+1}' - w_{l+1}) + \ldots + c_n(w_n' - w_n)] \\ &> c_l[(w_1 - w_1') + \ldots + (w_l - w_l')] - \\ &\quad c_{l+1}[(w_{l+1}' - w_{l+1}) + \ldots + (w_n' - w_n)] \\ &> c_l[\textstyle\sum_{i=1}^n w_i - \sum_{j=1}^n w_j'] = 0 \end{aligned}$$

Therefore, when $k > k'$, $W(n) > W'(n)$. In other words, a smaller amount of abstraction leads to a greater amount of lost work. We can derive from this result that in a changing environment, doing more abstract work is better, because less work needs to be redone.

## 5.3 The optimal amount of abstraction

The next problem we will address is finding the optimal amount of abstract work to do for a given problem situation. Given a partially-expanded plan to depth $d_1$ at the executable level, we would like to know how much work to do at each level of abstraction to extend the plan $s_1$ steps further. In particular, given the assumption stated earlier that the search depths are related by the relation $d_1 = d_2k = \ldots = d_nk^{n-1}$, and similarly $s_1 = s_2k = \ldots = s_nk^{n-1}$, we would like to know the value of $k$ that minimizes the work to extend the search.

The variables involved in the equation will be:

$n$: the number of levels of abstraction,

$d$: $(= d_1)$ the number of steps so far at the executable level,

$s$: $(= s_1)$ the number of steps to extend the plan,

$p$: the probability that a goal rules out a possible step in the plan,

$b$: the branching factor at each level of abstraction.

The model of search we use is that the plan is extended in the following order:

- For $k = 0, \ldots n-2$:
  - $s_{n-k}$ steps are taken at the $n-k$th level of abstraction.
  - The additional $s_{n-k}$ states are propagated to goals at the next lower level of abstraction.
  - These additional goals are used to narrow the existing plan space at the $n-k-1$st level—this revision of the existing plan incurs some overhead, but is quickly outweighed by the savings in extending the space.
- Finally, the plan at level 1 is extended $s_1$ steps.

If we denote by $R(b,d,p,k,n,i)$ the amount of work necessary to revise a plan at the $i$th level of abstraction, and by $W(b,d,p,s,k,n,i)$ the amount of work necessary to extend a plan $s$ steps at the $i$th level of abstraction, then we can see that the total amount of work $T(b,d,p,s,k,n)$ necessary to extend a plan $s_1$ steps at the executable level is:

$$T(b,d,p,s,k,n) = \begin{cases} (\sum_{i=1}^{n-1} R(b,d,p,s,k,n,i) + W(b,d,p,s,k,n,i)) + \\ \qquad\qquad W(b,d,p,s,k,n,n), & \text{for } n > 1 \\ W(b,d,p,s,k,n,n), & \text{for } n = 1 \end{cases}$$

Note that the revision work is done at all but the highest level of abstraction.

The amount of work necessary to revise a plan at a given level of abstraction is merely the size of the search space so far. The overhead of this revision is considerable, but when we considers that the fringe of the search space is of the same order as the entire search space, and that the size of the search space to extend a plan multiplies the size of the fringe by the (exponential) extension search space, then the overhead is more than compensated for by the savings when extending the search space. The actual amount of work necessary to revise the existing plan at the $i$th level of abstraction is the size of the search space for the first $d_i$ steps when constrained by the extra $s_{i+1}$ goals:

$$R(b,d,p,k,n,i) = \frac{[b(1-p)^{\frac{d+s}{k^i}}]^{\frac{d}{k^{i-1}}+1} - 1}{[b(1-p)^{\frac{d+s}{k^i}}] - 1}$$

The amount of work necessary to extend a plan $s$ steps at the $i$th level of abstraction differs when $i < n$ and $i = n$. When $i = n$, the highest level of abstraction, the search is not narrowed by any goals from a higher level of abstraction. At every other level, goals from the next higher level of abstraction narrow the search.

First, we define $A(b, d', p, g', s')$ to be the amount of work necessary to extend a search with branching factor $b$, at depth $d'$, with probability $p$ of a goal ruling out a step in the plan, with the number of goals $g'$, and the number of steps to extend the plan $s'$:

$$A(b, d', p, g', s') = [b(1-p)^{g'}]^{d'} \left[ \frac{[b(1-p)^{g'}]^{s'+1} - 1}{[b(1-p)^{g'}] - 1} - 1 \right]$$

Then we can define $W$ in terms of $A$:

$$W(b, d, p, s, k, n, i) = \begin{cases} A(b, \frac{d}{k^{i-1}}, p, \frac{d+s}{k^i}, \frac{s}{k^{i-1}}), & \text{for } i < n \\ A(b, \frac{d}{k^{i-1}}, p, 0, \frac{s}{k^{i-1}}), & \text{for } i = n \end{cases}$$

We can expand this out completely to get a full definition of $T$:

$$T(b, d, p, s, k, n) = \begin{cases} \Big( \sum_{i=1}^{n-1} \frac{[b(1-p)^{\frac{d+s}{k^i}}]^{\frac{d}{k^{i-1}}+1} - 1}{[b(1-p)^{\frac{d+s}{k^i}}] - 1} + \\ \quad [b(1-p)^{\frac{d+s}{k^i}}]^{\frac{d}{k^{i-1}}} \left[ \frac{[b(1-p)^{\frac{d+s}{k^i}}]^{\frac{s}{k^{i-1}}+1} - 1}{[b(1-p)^{\frac{d+s}{k^i}}] - 1} - 1 \right] \Big) + \\ \quad b^{\frac{d}{k^{n-1}}} \left[ \frac{b^{\frac{s}{k^{n-1}}+1} - 1}{b-1} - 1 \right], & \text{for } n > 1 \\ b^d \left[ \frac{b^{s+1} - 1}{b-1} - 1 \right], & \text{for } n = 1 \end{cases}$$

We can see from this that for $n > 1$, as $k \to \infty$,

$$T(b, d, p, s, k, n) \to \frac{b^{d+1} - 1}{b - 1} + b^d [\frac{b^{s+1} - 1}{b - 1} - 1] > T(b, d, p, s, k, 1)$$

Since $T(b, d, p, s, k, 1)$ is blind search, we see that the total work exceeds blind search slightly as the search approaches infinity (and the amount of excess is due to revising the existing search space, which is not necessary when $k = \infty$, since there will be no change at any level $> 1$). The value of $T$ as $k \to \infty$ corresponds to an infinitesimal amount of work at abstract levels, so it makes sense that it corresponds to pure blind search.

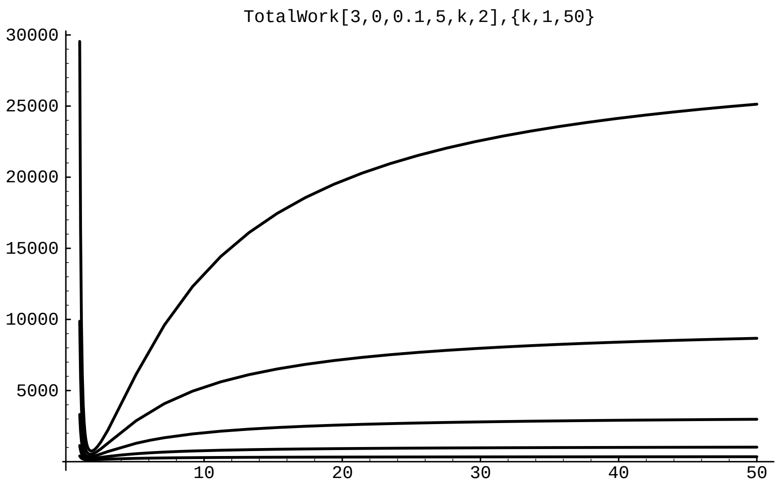


Figure 5.1: The amount of total work as a function of $k$, the ratio between levels of abstraction. The different curves result from different search depths.

We also can see that when $n > 1$ and $k = 1$,

$$\begin{aligned} T(b,d,p,s,k,n) &= \left(\sum_{i=1}^{n-1} \frac{[b(1-p)^{d+s}]^{d+1}-1}{[b(1-p)^{d+s}]-1} + [b(1-p)^{d+s}]^d \left[\frac{[b(1-p)^{d+s}]^{s+1}-1}{[b(1-p)^{d+s}]-1} - 1\right]\right) + \\ &\quad b^d \left[\frac{b^{s+1}-1}{b-1} - 1\right] \\ &> T(b,d,p,s,k,1) \end{aligned}$$

The reason that this is more work than blind search is that the search space at the highest level of abstraction is exploring the same size search space as a blind search when $k = 1$. Note that the highest level of abstraction is the only level not constrained by goals from a higher level of abstraction, so its search is not narrowed by a deeper abstract search. As long as the search at abstract levels is significantly shallower than at specific levels, the overhead of this unconstrained search will be outweighed by the savings in narrowing lower search spaces. But at $k = 1$, the overhead is already as large as blind search, so it is guaranteed to take more work than blind search.

The general shape of the curve is as shown in Figure 5.1. The curve drops sharply as the amount of abstraction decreases below the point where the overhead is the dominant term. Then it rises again and asymptotically approaches blind search again. The optimal value of $k$ is the value for which the amount of work is minimized.

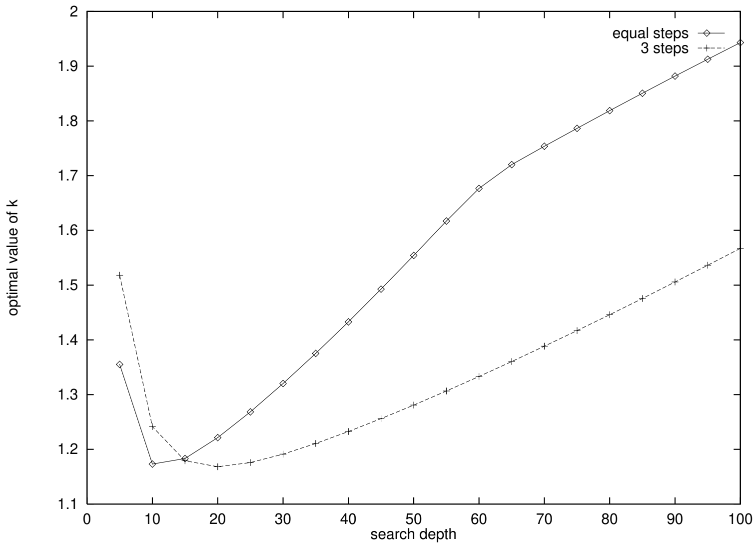


Figure 5.2: The optimal value of $k$ increases with search depth.

Although $T'$ does not yield to analysis, we can use numerical methods to find the optimal value of $k$ for any given problem situation. In fact, because of the steepness of the curve, numerical methods converge quickly on the approximate minimum.

## 5.4 The optimal amount of abstraction at different depths

Given the analysis of the previous section, we can compute the optimal value of $k$, hence the optimal amount of abstraction, for any given problem situation. For instance, we can examine the behavior of the optimum as the depth of the search increases.

Figure 5.2 shows this for one specific (but representative) case. Here we assume that we have searched to depth $d$ at the most specific level of abstraction (level 1). We also assume that to extend the search $s$ steps, we first search $\frac{s}{k^{n-1}}$ steps at level $n$, then $\frac{s}{k^{n-2}}$ steps at level $n-1$, using the goals inherited from level $n$, then so forth until we search $s$ steps at level 1, using the goals inherited from level 2. The two lines on the graph are the optimum for doubling the search depth versus extending it a small constant amount (in this case 3 steps). Note that in both cases, as the search depth increases, the optimal value of $k$ increases, which means that the optimal amount of abstraction decreases.

Because we are using numerical methods to find the optimal value of $k$, this trend can only be explained qualitatively. Recall that the most abstract search is unconstrained by any goals. So as the search deepens, the cost of taking more steps at the most abstract level increases. At the same time, the lower searches are becoming more and more constrained by the goals from the more abstract plans. After a point, the incremental constraint of adding another goal is outweighed by the cost at the most abstract level of extending the search, so the optimum shifts in the direction of less abstract search.

## 5.5 Summary

Starting with a reasonable set of assumptions, we have shown the following:

- When the environment is changing, more abstraction leads to a more stable plan.
- For any given problem situation, we can determine through numerical methods the optimal ratio of work between adjacent levels of abstraction.
- As the depth of the plan increases, the optimal amount of abstraction decreases, as the cost of extending the abstract plan outweighs the added benefit of the additional goals.

# Chapter 6

# Empirical results

To further validate the method beyond the analysis presented in Chapter 5, we tested it empirically by applying it to a specific domain. The results were produced using a simulation of the planning system. The implemented system runs on smaller examples, but the simulation was necessary to generate the wider range of empirical data that we needed. We investigated the overall performance of the method to see whether it had the desired behavior of graceful degradation under resource constraints. In addition, we repeated a part of the analysis to see whether its predictions would hold for a real problem domain.

## 6.1 The domain

We chose to apply our method to the domain of an office robot shuttling papers around for grant proposal preparation. Within the domain of grant-proposal preparation, there are operators to carry documents in various stages of preparation to other people and places for them to be further processed. The abstract operators are more concerned with the status of the grant as a whole and less with the robot's position relative to the paperwork.

The search space of the specific operators is shown in Figure 6.1. Since the arcs between the states are of varying length, the actual search tree followed by the planner may visit a single state several different times. The actual search space with the added dimension of time is thus unlimited in size, even though the number of states (independent of time) is

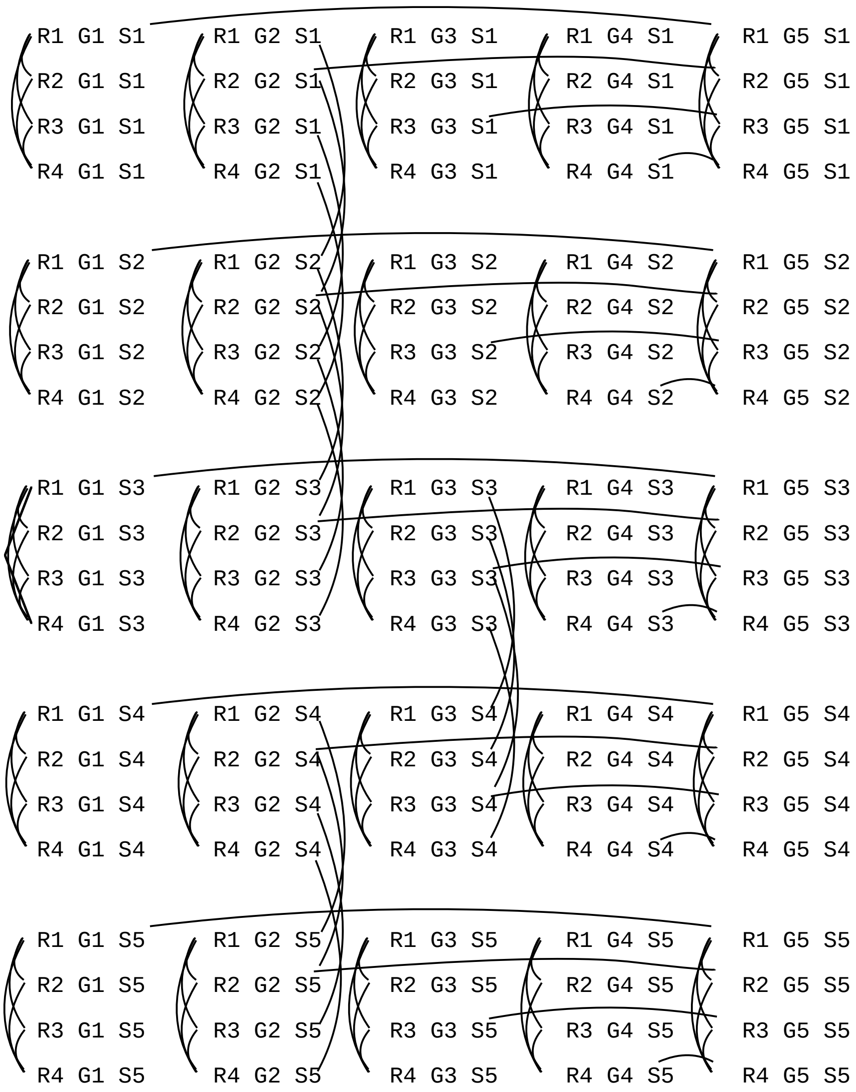


Figure 6.1: The space of states possibly visited in the grant-proposal domain. Each state is a triple of robot location (R), grant location (G), and grant status (S). The arcs represent the possible transitions between states.

limited.

The experiments we performed fall under the following two categories:

1. Overall performance. We explored the behavior of the method under varying resources and search strategies. The aim here was to see whether the method would actually exhibit the desired behavior of graceful degradation under time stress.

2. Finding optimal strategies. We examined how much search work was actually required to expand the plan to various depths under different search strategies. The aim here was to see whether the analytical results about optimal search strategies could be replicated in an actual problem domain.

All of the experiments are the result of applying the planning method with two levels of abstraction to a problem that requires a search depth of 16 steps at the specific level, and a search depth of 8 steps at the abstract level.

## 6.2 Overall performance

The first category of experiments measures how performance is affected by varying the resources available to the planner. We measured the quality of the evolving plan during plan construction, where quality represents the percentage of the goals actually achieved if the steps in the plan were actually executed (see 3.2.1 for the precise definition). We measured this over a number of search strategies.

Each search strategy requires a different amount of search to fully expand the plan. So combining the results from multiple search strategies can be done either by just using the absolute search size of the searches for each strategy, or by normalizing the searches to be a percentage of the search required to fully expand the plan. We present results from both of those possibilities.

First, consider the absolute search effort expended. In Figure 6.2 that the quality of the plan rises as the amount of search work increases. This particular result is swamped by the worst search strategies, however. The worst search strategies will require the most search

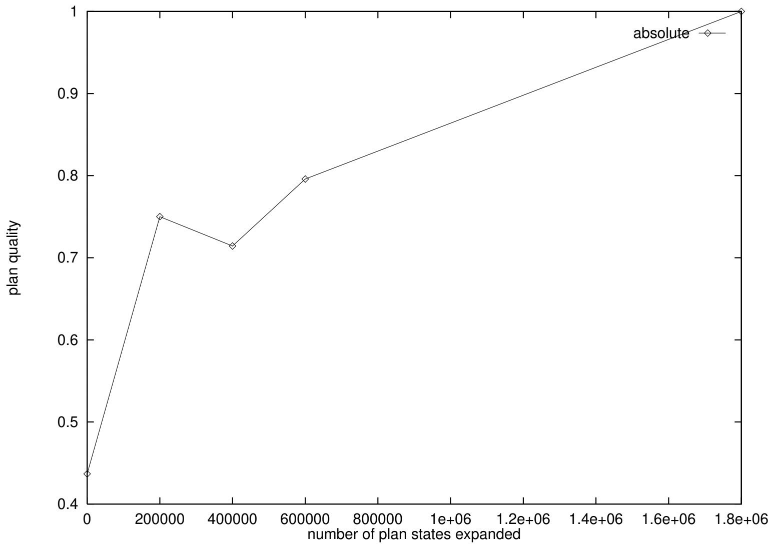


Figure 6.2: The quality of the plan in relation to the search effort expended. The data are averaged to determine each graphed point.

effort, so the data points contributing to the graph at the upper end of the search effort scale are almost solely from the worst search strategies.

To inspect more closely how the curve behaves in the areas affected by all search strategies, we restrict our focus to the lower end of the search-effort spectrum. See Figure 6.3 for the results within that range. Although there is more variation, we still get the kind of incremental improvement we are looking for.

Now consider the case where we normalize the results so that the total search work to expand the plan fully is the same for every search strategy. See Figure 6.4 for the graph of these results. The resulting curve again shows the desired behavior.

Note in all the curves that, as we might expect, we gain a fair amount of improvement early, but require much more effort to improve the plan to a near-optimal level. This supports the idea that with limited resources, we can still generate useful, albeit potentially suboptimal plans.

To see what improvement abstraction offers, we compare the amount of search work done with the optimal abstraction strategy versus search solely at the base level. In Figure 6.5 we see that the time to reach an optimal plan with abstraction is significantly shorter

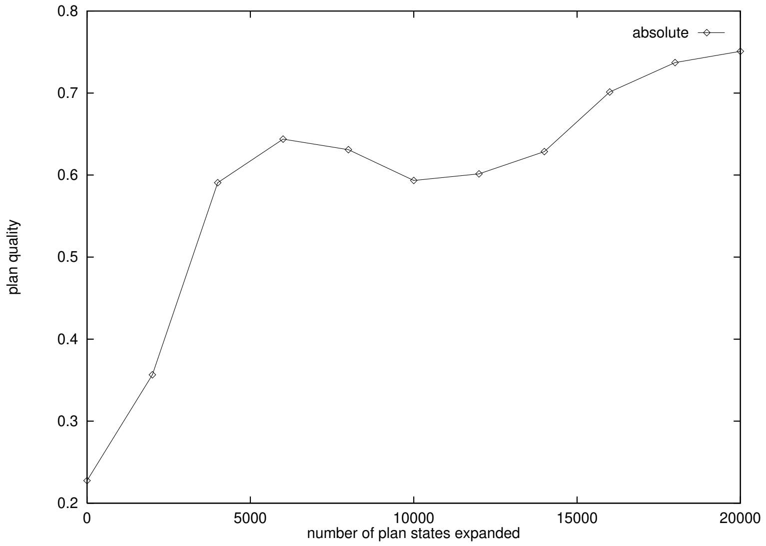


Figure 6.3: The quality of the plan in relation to the search effort expended, restricted to the lower end of the search-effort spectrum.

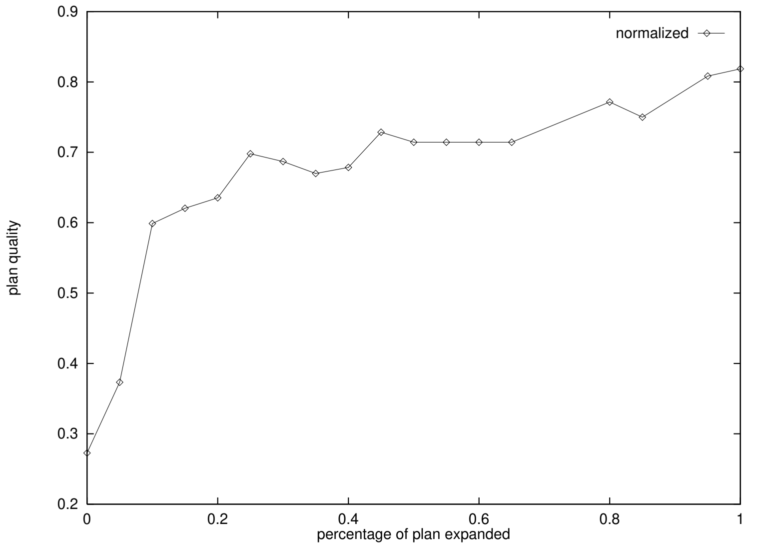


Figure 6.4: The quality of the plan in relation to the search effort expended, when the search spaces are normalized across search strategies.

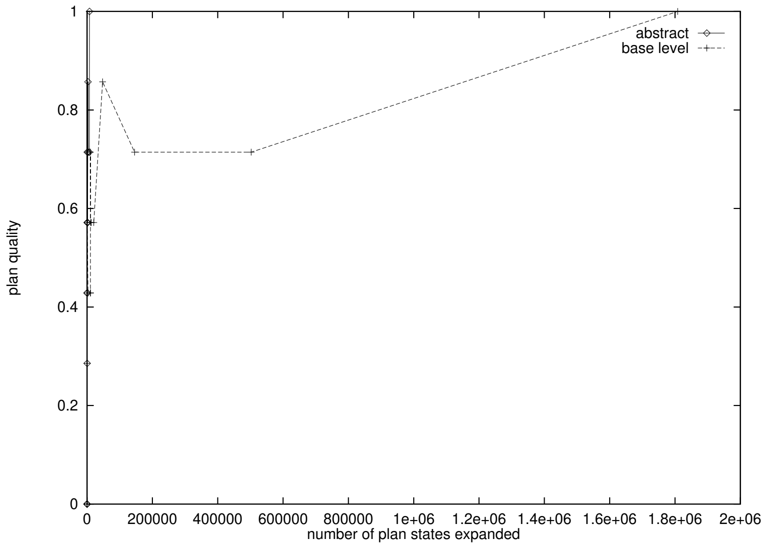


Figure 6.5: The quality of the plan in relation to the search effort expended, with abstraction and without. The abstract plan reaches the optimum near the left edge of the graph.

than search without abstraction.

Since the graph is not overly informative about the abstract search, we further inspect the relation between abstraction and base-level search by concentrating on the section of the search where both abstraction and base-level search are present. See Figure 6.6 for that portion of the graph. We see that search with abstraction is significantly better for the same amount of search than search without abstraction.

In summary, the empirical results support the basic underpinnings of the work: producing plans with graceful degradation under time stress, and producing the plans more efficiently than with only base-level planning.

## 6.3 Finding optimal strategies

Next we explore the repeatability of the analytical results on finding optimal strategies. For this experiment, we varied the search depth and the search strategies to see which strategies were best for each search depth.

Recall from the analytical results in Chapter 5 that a search strategy is changed by

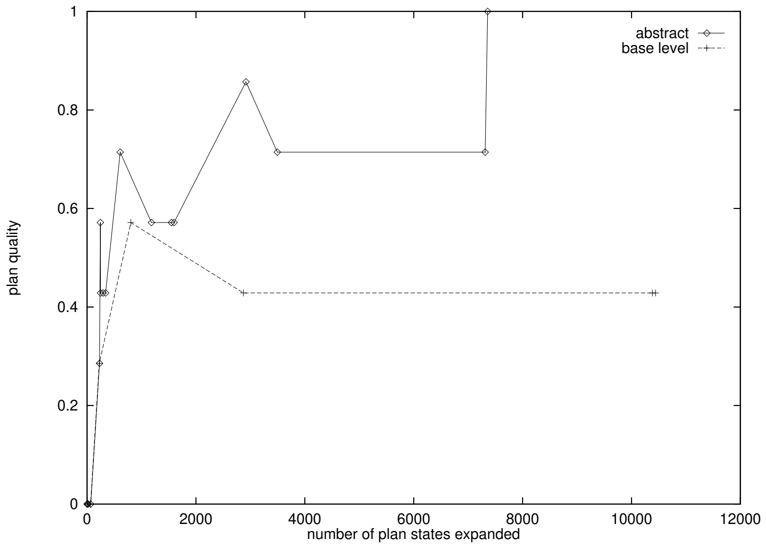


Figure 6.6: The quality of the plan in relation to the search effort expended, with abstraction and without, restricted to lower amounts of search effort.

varying the ratio of the work done at successive levels of abstraction. So a ratio of 2 reflects that the search depth at the less abstract level is twice the search depth at the more abstract level. This is in terms of the number of steps, since we expect the more abstract search to extend further in time than the less abstract search.

In the actual problem domain, the abstract search reached its goal in 8 steps. Because of that, a search depth of 8 and a ratio of 1 would reach the abstract goal. Any deeper search would fall outside of the analytical model, so we restricted our experiment to search depths of less than 8. Search depths of 0–4 were basically flat, because the states within that range are unaffected by the abstract goals. We include search depth 4 to reflect those depths, but we focus on search depths of 5–8 at the base level.

Figure 6.7 shows results similar to those from a purely analytical setting. Note that in this case, the limiting value of the search work as the ratio increases past the optimal is higher than the amount of search work required with a search ratio of 1. This differs from the original analytical results because the abstract search space in the experimental domain has a lower branching factor than the specific level. In the analytic model, the branching factor is the same across levels, so the work required when the ratio is 1, where

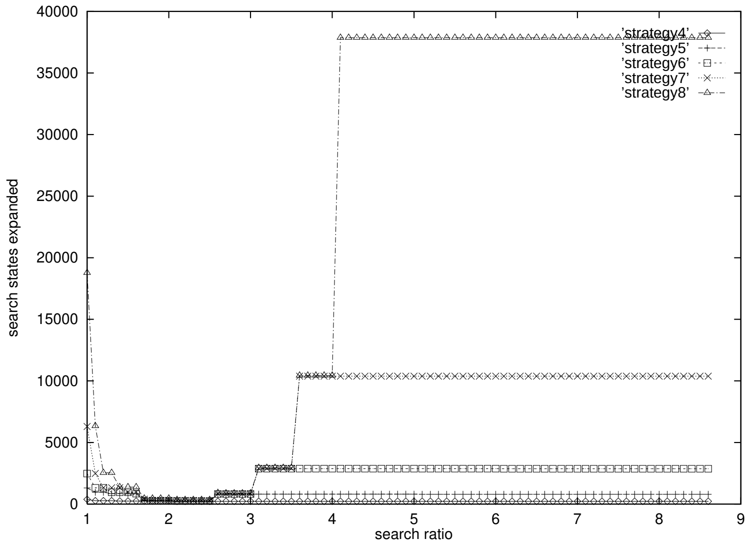


Figure 6.7: A plot of total search work required versus the search ratio of base-level steps to abstract steps, for search depths 4–8.

the majority of the work is due to the abstract level, is equivalent to the work required as the ratio approaches $\infty$, where the search is entirely at the base level. Note also in the figure that the work required levels off rather than asymptotically approaching the limit. This is because for these particular problem descriptions, the abstract level performs no work after the point where the curve levels off (a point which is only asymptotically approached in the analytical model).

One feature of the state space explored by the planner is that there is a great deal of duplication: the same state appears in multiple positions in the search tree. We have discussed how the states are augmented with temporal information, so there are more states possible than the set shown in Figure 6.1. However, even with the additional temporal information, there are multiple equivalent states within the search tree.

The implemented system does not look for repeated states, since that is a potentially computationally complex operation. But to see whether eliminating repeated states would make a significant difference for finding optimal search strategies, we eliminated repeated states from the experimental results. The results of that are shown in Figure 6.8. The optimal search strategies are shifted somewhat from the original case, but overall the influence

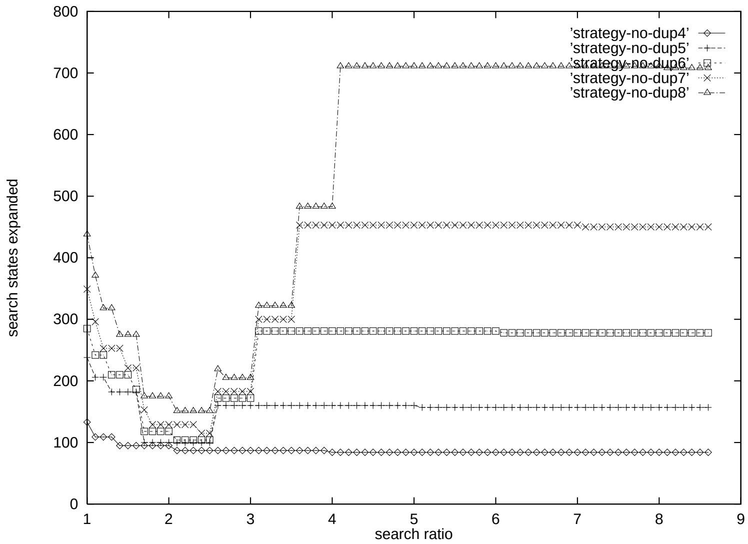


Figure 6.8: A plot of total search work required versus the search ratio of base-level steps to abstract steps, for search depths 4–8. Duplicate states are eliminated.

of the search strategies is quite similar.

A further test we can perform is to regenerate the analytical data for the particular problem we are using for the experiments. Although the analytical model is somewhat simplified, so that some parameters of the model aren't easily characterized for a particular problem (such as the probability of a goal affecting a state), we modeled the domain as accurately as possible. The results of regenerating the analysis for various depths are shown in Figure 6.9.

A few differences appear between the analytical data and the empirical data. First and most obvious is the smoothness of the analytical data. This is because the analysis does not model the discrete nature of the planning search—for instance, there is no way to expand the search 0.3 steps. For the purposes of comparison, we can discretize the analytical data. Doing that produces the graph shown in Figure 6.10. The graph now looks more similar to the empirical results, but still some differences remain.

The optimal value for the search ratio is a bit different in the analytical model, ranging from 1.52 at a depth of 4 to 1.68 at a depth of 8. This compares to an optimal ratio of 1.7–2.6 for the actual domain. The discrepancy is due to the differences in the actual domain

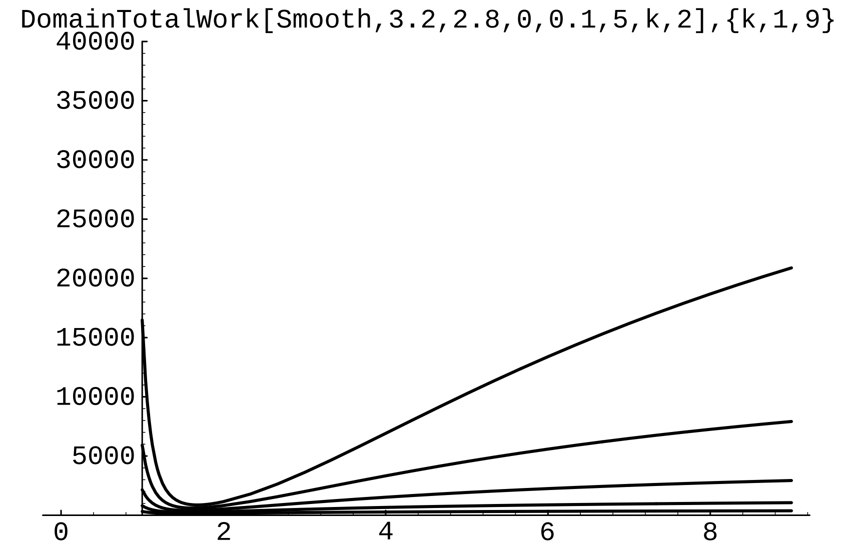


Figure 6.9: The results of the analytical model applied to the experimental problem for search depths 4–8.

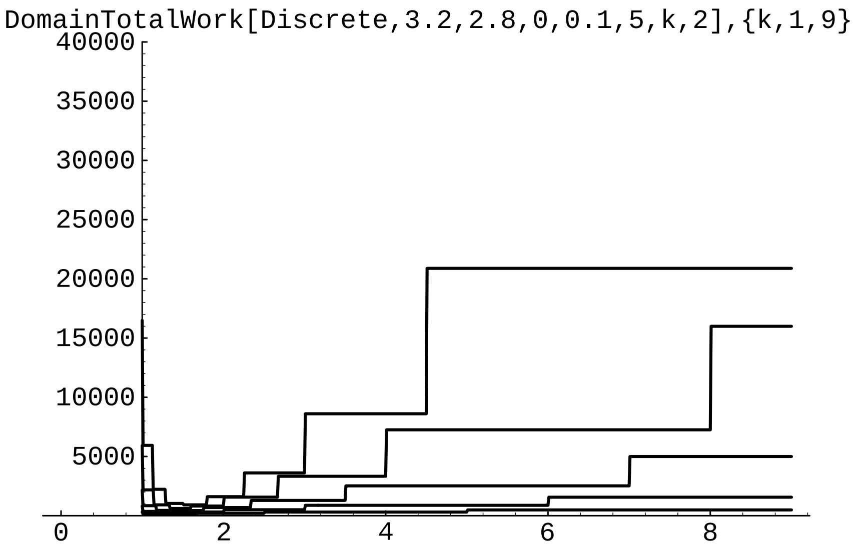


Figure 6.10: The results of discretizing the analytical model for the experimental problem.

versus the model that we constructed for the analysis. But it is important to note that the optimal search strategy for the analysis is close enough to optimal for the empirical case to make the analytical model a useful approximation for the real domain.

The other difference is the behavior of the analysis for large values of the search ratio. Because of the particular behavior of the operators in the experimental problem, the total work required levels off after a certain depth. The analytical model doesn't account for such behavior, so it continues to grow slowly as the search ratio increases. This is again a minor difference in qualitative terms, as it makes no difference in the area where we are interested.

These results show that the analytical model provides a reasonable approximation to the data from an actual problem. The techniques described in Chapter 5, applicable to the analytical model, may thus be used in actual problems.

# Chapter 7

# Related work

This work has roots and relations spread over a wide range of the planning and reasoning literature. In this chapter we will show how our ideas are similar and different from other work in planning and other related areas. We will organize our discussion around some of the major themes that appear in the work: resource-bounded planning, abstraction, temporal reasoning, replanning, blackboard-based planning, and integrated execution and planning. Some of the cited works may fall in more than one category, or only fit crudely within one of the boxes we have drawn; we will point out these as they appear. The discussion is not meant to be exhaustive, but rather to cover the major ideas in the fields.

## 7.1 Resource-bounded planning

A growing body of work is devoted to the problem of resource-bounded planning. There are a few major themes within the field. One group of researchers finds an imperfect but complete plan and then incrementally improves it. Another group incrementally adds steps to a partially-developed plan. Others find a single plan using static information about the problem and resource constraints. And finally some researchers abandon run-time planning altogether, opting for a precomputed plan.

### 7.1.1 Improving imperfect plans

One approach to incrementally building a plan is to build an approximate plan first, then to improve it incrementally. The attractive aspect of this approach is that the system always has a complete plan reaching from the start state to the goal, so the plan may be improved up until execution time. The drawbacks of this approach are that building the first approximate plan may take a significant amount of time itself, and the quality of that first plan may be arbitrarily bad.

All of the approaches discussed here assume that planning occurs before run-time. The major effect of this is that there is no mechanism for incorporating new information into an existing plan, and no way to repair a plan when it becomes outdated.

In addition, all of the approaches operate within a single level of abstraction.

#### Progressive horizon planning

Rymon, Webber, and Clarke [Rymon *et al.*, 1992] present an algorithm they call *progressive horizon planning*. The basic idea is that a simple polynomial algorithm is used to produce as complete a plan as possible. Then the beginning of the plan is explored exhaustively: for each possible alternative plan prefix, the simple algorithm is run to add a more-or-less complete plan from the end of the prefix to the goal. This complete plan is then evaluated to compare the competing plan prefixes.

The reason that the plan prefixes are more carefully explored is that the first few steps are the most important in a real-time, unpredictable domain, and the world may change and make later steps inapplicable by the time they are reached anyway. The prefixes are restricted to a fixed size, which is 1 in the described implementation. This size, or any fixed size, may not be appropriate for all problems or resource limitations, and may lead to horizon effects. Also, the initial plan may be arbitrarily bad, so optimizing the first step or first few steps of an arbitrarily bad plan may not get you very far.

The reason that complete plans are evaluated is to provide some global perspective on the evaluation. But again, the completions of the plan prefixes may have no relation to the best possible plan completion, so evaluating the results of the quick and dirty plan completion algorithm may have no relation to the actual worth of the plan prefix.

### Boddy and Dean

Boddy and Dean [Boddy and Dean, 1989] present a continuation of their work on anytime algorithms [Dean and Boddy, 1988], focusing here on ways to build and improve plans in an anytime fashion. Their paper actually falls within both the realm of improving imperfect plans and extending incomplete plans.

Their basic idea is to construct an incremental planner, specifically a path planner, by combining an incremental best-first search with an incremental edge-swapping route-improvement algorithm. Within their limited domain, they found that combining the two incremental techniques provided more benefit than either one in isolation. To reach this conclusion, they make the assumption that all steps of the plan are of equal cost, so the result is of questionable generality.

If one considers the operations of adding a step to the plan and swapping two edges of the plan to be the basic plan modification operations, then the algorithm becomes a best-first search through the space of plan modifications, which is a standard planning technique, in this case exhibiting strict anytime behavior.

### Reaction-first search

The work on reaction-first search [Drummond *et al.*, 1993] falls best under the heading of improving incomplete plans, although it is actually restricting a search space by incrementally extending a simulation of a reactive system. The approach assumes an underlying reactive system that is simple enough to simulate. A planning system on the side is allowed a certain amount of time to suggest plan constraints to the reactor to help it improve its plan.

The planning system is told that the reactor's job will be to achieve a goal or set of goals. The planning system then simulates the reactor's program faithfully, including any probabilistic choice points. When it finds a dead end, it backs that information out and revises its plan to eliminate the dead end from consideration. The evolving plan is guaranteed to improve monotonically, since at worst the planner is simulating the best plan known, and at best is improving it. So the expected value of the plan improves monotonically over time. When the deadline for action arrives, the planner sends the reactor the revisions to its original plan in the form of advice not to take certain branches of the plan, namely the ones that are guaranteed to lead to dead ends.

For a complex domain, the assumption that the reactor's search space can be effectively explored is at best a questionable one. Since the plan only improves when dead ends are reached, then these dead ends must actually be found, which may take a large amount of search. In addition, a domain with reversible actions will completely defeat the method.

### 7.1.2 Extending incomplete plans

The planning method described in this thesis relies on incrementally extending the plan at multiple levels of abstraction. A few other researchers have also adopted the approach of extending an incomplete plan. By building a plan incrementally in this way, the planner always has an executable plan prefix that it can use when deadlines arrive.

For this approach to be useful, the planning must be done left to right so that there is a usable plan prefix. Also, the quality of the plan prefix depends on the search control. With good search control, the plan prefix will closely approximate the optimal plan. With search control of lower quality, the plan prefix may diverge arbitrarily far from the optimal plan.

#### Real-time $A^*$

Korf [Korf, 1990] describes a heuristic search technique for incrementally extending a search in best-first fashion. This is a weak-method search, in that it can be used anywhere a standard search technique is used. It relies completely on its search control, since it operates at a single level of abstraction. It also uses the classical notion of static and discrete states and operators, so it is of limited use in dynamic domains. The basic technique is useful as a basis for other algorithms, however.

### Bratman, Israel, and Pollack

Bratman, Israel, and Pollack [Bratman *et al.*, 1988] offer a general discussion of the issues involved with combining means-ends reasoning, deliberation among alternatives, and resource-bounded reasoning. The architecture they present is general, and vague on details, but the general idea is that unsatisfied "intentions" are the important aspects of the plan (as opposed to "desires"). Means-ends reasoning is used to build plans for those intentions. Other gaps in the plans are filled in with means-ends reasoning as resources allow. The deliberation mechanism is left undescribed.

The major drawback with this approach is that by committing to build plans for the intentions, the planner is required to do a potentially large amount of work before the resource-dependent improvements are made.

### Durfee and Lesser

Durfee and Lesser [Durfee and Lesser, 1986] present a blackboard-based approach that uses multiple levels of abstraction. Within the domain of vehicle control, the system plots tracks at high levels of abstraction, which then provide intermediate goals to the lower levels of abstraction.

The generalities sound similar to the work in this thesis, but there are many differences. First, the plans at the higher levels of abstraction are completely expanded—only the lowest level of abstraction is expanded incrementally. In contrast, our approach is uniform across all levels of abstraction, so that the plans are expanded incrementally at all levels. The states are annotated with times, but they are merely annotations about the expected position of the vehicle at particular times, as with as air-traffic control system. There is no general treatment of time. The states and goals are considered to be "milestones" by the lower levels, which try to to achieve them in sequence. We have argued that this is unrealistic for complex domains.

#### Situated control rules

Some of the basic underlying ideas of this thesis share some similar features with situated control rules [Drummond, 1989]. The ideas of situated control rules are relatively simple: A plan is built incrementally from the current state forward in time, allowing sets of parallel operators as well as individual operators. A best-first search is performed within this space of operators and operator sets, and this may be terminated at any time. In addition, new information about the environment will move the current state to a different point in the search space, where the search will continue or restart.

The simplicity of the approach is also its weakness. There is no treatment of time in situated control rules, and the states are discrete and static. There also is no treatment of abstractions and how they affect the search.

### 7.1.3 Formally-based approaches

In this section we discuss approaches to resource-bounded planning that are concerned mainly with the problem of having a clear mathematical formulation of the planning problem and solution. In doing that, they of necessity give up some of the power that more informal solutions provide, trading that off for mathematical elegance and simplicity. For example, abstraction is not an easily-formalized technique, so is missing from these approaches. Also, although temporal information has been formalized by a number of researchers, temporally-based resource-bounded approaches have not yielded to much mathematical analysis.

The formal approaches can also suffer from the problem that real domains often do not behave the way that formal models require. Real domains may not have context-independent utilities, or at least not any that are easy to determine. They also just may not follow any easily-determined rules that would be necessary to carry out the analysis required by the approaches.

### Decision theory

Horvitz [Horvitz, 1987] is the name most closely associated with resource-bounded planning using decision theory. His approach shows how one could find the optimal planning step to take given enough utility measures. Although the approach is attractive in that it assures an optimal solution given the resource constraints, it also requires utilities of an order that are rarely available, or would require immense effort to catalog even approximately.

Dean and Boddy's original work on anytime planning [Dean and Boddy, 1988] also falls reasonably within the category of decision-theoretic approaches, since it is designed to produce a plan of increasing utility over time. Again, it suffers from the problem of determining utility, and also from the lack of complex algorithms that strictly satisfy the definition. It puts a number of restrictive requirements on the problem domains that are not easily realized in real domains. For instance, actions must occur at exactly the expected time.

### Approximate planning

Ginsberg [Ginsberg, 1994] tackles the problem of resource-bounded planning by building an "approximate" plan. An approximate plan is a nonlinear plan that is approximately correct, in the sense that any exceptions to the plan—things that would make the plan fail—are of measure 0 in the plan. For instance, if there are an infinite number of possible variable assignments, a plan with a variable assigned is of measure 0 in the plan without an assignment. Basically, the idea is that there are infinitely more ways for the plan to succeed than to fail, so the probability of the plan failing is 0.

One obvious problem with this approach is that it needs an infinite domain. A finite domain will have no plans of measure 0. There is no distinction between low and high probability exceptions in a finite domain, so a complete plan would have to be built for all finite domains.

Even for infinite domains, this approach still requires a fixed amount of time no matter what the resource bounds, so there is no adaptation to various resource allocations.

### 7.1.4 Reactive planning

An extreme approach to resource-bounded planning is to precompute the optimal action for all possible situations. In that case "planning" is reduced to matching the current situation, via a table lookup or discrimination net, to the set of precomputed situation-action pairs, and then performing the associated action.

The prototype of this approach is the universal plans approach [Schoppers, 1987]. A universal plan is given a goal or set of goals, and it uses a simple backward-chaining planner to find all the possible states that can reach the goal via a sequence of actions. For each such state and action sequence, the first action of that sequence is associated with the situation, and the process continues.

The obvious advantage of this approach is speed, since it reduces the planning problem to a simple match problem. Given a complex problem, the number of situations may make the match process nontrivial, but it is still potentially much faster than planning de novo.

Another advantage of the approach is that replanning is immediate and integrated into the approach. In fact, that is one of the initial reasons for the rise of this approach. Since all the situations are precomputed, then the planner never assumes that it is in the state that a classical planner would have predicted it to end up in. Instead, each new situation is matched against its situation-action set, and the appropriate action is executed. This way another agent may affect the world, and the planner will be able to take the appropriate action for the changed situation with no extra effort.

The obvious disadvantage of this approach is space. A complex domain may require an immense, if not infinite, amount of space to capture the entire possible search space. See [Ginsberg, 1989] for a in-depth discussion of the problems inherent in the approach.

If a less extreme position is taken, where only some of the plan is precomputed, then the question that arises is what happens when a situation is reached for which no action has been precomputed. This reverts to the problem of planning, and by itself the reactive planning approach has no advice to offer when planning is required at run time. Our approach takes the opposite approach—since we acknowledge that some planning must be done at run time, we are trying to figure out how to solve that part of the problem. If you can run even 90% of the time, but don't know how to walk, the remaining 10% of the time you will be stuck. If you know how to walk 100% of the time, you may be slower 90% of the time, but you won't get stuck.

Mulder and Braspenning [Mulder and Braspenning, 1992] offer a compromise from the strict reactive approach by building a planner that puts together pieces of reactive plans into a larger plan. The problem here is that the initial reactive plans are built only by carefully analyzing the domain to choose appropriate partial plans for the problem at hand. Again, the appropriate reactive plans must be on hand in order for the planner to be able to compose them into a larger plan.

## 7.2 Abstraction

Abstraction is used in our approach both to provide global search control for the planning process and to construct a framework around which replanning can operate. Neither of these notions is particularly revolutionary, although allowing partial abstract plans is not common in the abstraction literature.

### 7.2.1 Early uses of abstraction

Abstraction in planning dates back to Planning GPS[Newell and Simon, 1972]. In that system, a logic theorem was transformed into a more abstract formulation, where it was then solved, and the abstract solution was used to guide the lower-level solution.

ABSTRIPS [Sacerdoti, 1974] was the first general-purpose abstraction-based planner. Using abstractions generated semi-automatically based on criticality ratings, ABSTRIPS generated a plan for the most abstract level, then tried to fill in the missing steps at the next more specific level, and so forth. The language at each level is the same, except that preconditions below the criticality rating for that abstraction level are dropped. Each level of abstraction was planned completely before the next level.

NOAH [Sacerdoti, 1977] and its derivatives take a different view of abstraction that is more like procedure calling. An "abstract" operator is merely a placeholder for a sequence of lower-level goals, where the lower-level goals are defined ahead of time. In NOAH, this expansion into lower-level goals was hidden in Lisp code, but in more recent planners of this sort like SIPE [Wilkins, 1988], the expansion is declarative and represented as part of the abstract operator. Theoretically there could be multiple expansions of an abstract operator, but the planners of this sort either could not backtrack over operator expansions or at best discouraged this, since the search space of operator expansions is orders of magnitude larger than the base-level search space [Christensen, 1989].

### 7.2.2 Alpine

Knoblock formalized the ABSTRIPS approach in ALPINE [Knoblock, 1991]. He also analyzed the effects of abstraction and showed that under a particular set of assumptions abstraction could reduce the overall effort of a planner from exponential to linear.

Unfortunately, the assumptions do not hold up well in real domains. The main assumption, the *downward solution property*, assumes that any refinement of an abstraction will lead the lower-level search to a goal. Also, the intermediate goals generated by the abstract search can be solved independently in the order generated. We have argued in Chapter 3 that this latter characteristic is unrealistic, and we have performed analyses with a set of assumptions more appropriate for the domain characteristics we think are found in real domains.

### 7.2.3 Spatula

We have briefly mentioned Unruh's system SPATULA in Chapter 3. It is a planner built in the SOAR problem-solving architecture for dynamically formulating and using abstractions while planning. The basic idea is that when the planner is faced with a choice among operators, it will abstract the space dynamically and continue building its plan in the abstract space. This process continues dynamically until an operator becomes preferred over the other possibilities. The information learned from the abstract search is then used as search control at the executable level. Where the search control is lacking, the planner will again face a choice and dynamically abstract to choose an operator.

Although Unruh's approach is not designed for resource-bounded planning, the dynamic abstraction does provide a model for a possible alternative approach to resource-bounded planning with abstraction. Instead of working top-down from the abstract space to the executable level, the planner could work at the executable level, abstracting only as necessary to choose operators at the executable level. The drawbacks of that approach are that since the abstraction is built and destroyed dynamically, it does not remain as a framework for extending the plan or replanning.

## 7.3 Temporal reasoning

Although the focus of this thesis is not on temporal reasoning, it draws heavily on ideas from previous work in the field. The basis of our representation is temporal intervals, and our states are sets of these intervals reflecting the current state of knowledge within the planning system.

Much of the work on temporal reasoning has been on the logical foundations underlying the representation [McDermott, 1982; Allen, 1983; Shoham, 1989]. We have chosen a more applied approach, incorporating ideas from these formal models as appropriate to realize our conceptions of how time can be used in a resource-bounded system.

Allen's classic work on temporal intervals [Allen, 1983] describes how time relations can be represented and reasoned about, and [Allen and Koomen, 1983] describes how this representation could be used for building plans. As with other temporal-reasoning systems, however, there is no idea of real-time reasoning using this temporal information. The work in [Allen, 1983] describes a way of clustering intervals into hierarchies that could make the reasoning more efficient, but this was just one step in the right direction.

Dean's thesis on time maps [Dean, 1985] resembles closely the type of representation our method has grown to become. In fact, Dean briefly acknowledges the efficiency and representational problems of hypothetical worlds when using a representation of this sort to perform planning. The planning system actually described in his thesis maintains a single world state, however, and is not concerned with resource bounds. Also, abstraction and replanning are not addressed in that work.

Deviser [Vere, 1981] was one of the early planning systems to use time in a system designed for real problems. The system basically added temporal information to a NOAH-style planner. Using the added temporal information, the planner could augment the standard constraints of operator precedence found in NOAH with more sophisticated temporal information. Deviser had no handling of abstraction, nor did it concern itself with resource bounds. It was more suited to scheduling tasks before execution time, so it was not designed to handle run-time problems that required replanning or resource-bounded reasoning.

More recently, Penberthy [Penberthy, 1993] has formulated what amounts to a formalization of Deviser. Building on recent formalizations of NOAH-style planners, Penberthy adds temporal information to the formal model in much the same way that Vere added temporal information to NOAH. As such, it provides an interesting formal model, but as an implemented system, it has little to say about the problem of resource-bounded planning, since it and its underlying formal planner are built for formal cleanliness and suffer along the lines of efficiency. In addition, all the planning is at one level, and replanning is not supported.

## 7.4 Replanning

A fundamental aspect of our method is that it is designed to operate in a dynamic environment, where the world may change in unexpected ways, and actions may not always have their expected effects. We consider the plan to be a constantly-evolving structure that is modified not only by the planner but also by the outside world and potentially by other reasoning modules within the planning agent. So the planner must always be prepared to respond to changes whenever and wherever they occur within the plan.

Other researchers have recognized the need for replanning. Most have considered replanning to be a separate operation from planning: the plan is built in isolation from the world, and then it is handed to an execution module. The execution module will call the planner when a part of the plan fails and needs replanning. This approach dates back to STRIPS [Fikes and Nilsson, 1971], where triangle tables were introduced for replanning. Triangle tables have only limited replanning power, since they are restricted to changes that move within the particular plan sequence already computed. A generalization of triangle tables, teleo-reactive trees [Nilsson, 1994], falls more in the category of reactive planning.

The first use of abstraction for replanning is due to NOAH. NOAH introduced the idea of finding the "wedge" of the plan affected by a world change, and replanning just that wedge. We follow a similar approach to that of NOAH for finding the highest level of abstraction where the world change has an effect on the plan, although as described in Chapter 3, our approach only replans as much of the plan as required by the dependencies of the plan on the particular bit of information that changes.

Pollack's work on IRMA [Pollack, 1992] is a higher-level view of the planning and replanning process. She ignores the lower-level plan-construction process, apparently assuming that plans to achieve individual goals will be short enough to be performed within limited resource bounds. This is despite criticism of traditional planners for not viewing the world as a dynamic environment while building and repairing plans (an opinion that we share). IRMA instead focuses on controlling the overall planning process. In particular, the work addresses the decision whether to plan for intentions. In other words, as new information arrives from the world, new intentions are formed. As opposed to our approach, where the planner finds partial plans to satisfy all of its goals, IRMA decides to ignore some of its goals and plan completely for the rest of them. This work also could be described as an incremental planner, but with its emphasis on incorporating dynamic information into the planning process, there are more similarities in spirit with our work in the area of replanning.

Mulder and Braspenning's work was described in Section 7.1.4. It also includes facilities for monitoring and replanning. For each of a predefined set of classes of time available, an amount of replanning is associated with it. When a conflict is noticed in its existing plan, the plan is replanned to the amount predefined. Since the entire approach relies on precomputation of the component reactive plans and the required amount of replanning, it presents a somewhat inflexible approach to the problem.

The entire field of case-based planning [Alterman, 1988; Hammond, 1988] could be considered related to replanning, since the emphasis is on fitting an existing plan to a new situation. Case-based planning and reactive planning are closely related, although it is unclear that either field would admit to the relation. Case-based planning tries to find an existing plan in its library that most closely fits the current situation, and then modifies it to be applicable to the situation. The plans in the plan library are pre-stored. The advantages of case-based planning over reactive planning are that plans can be stored as a result of experience, and that a relatively small set of pre-stored plans can be modified to fit a wide range of situations. The disadvantages are that the match in case-based planning is much more computationally expensive (and relatively ill-defined), and the plan-modification procedure is potentially expensive itself (and again ill-defined). In case-based planning little thought is given to operating in a dynamic environment because of the computational cost of the planning process.

## 7.5 Blackboard-based planning

The work in this thesis derives many of its basic ideas about incremental planning and the combination of goal-driven and data-driven reasoning from the blackboard problem-solving framework.

Some early ideas about blackboard-based planning appear in the work on OPM [Hayes-Roth and Hayes-Roth, 1979]. OPM is an attempt to replicate human planning behavior, in particular opportunistic planning and replanning. The work builds plans at multiple levels of abstraction, and builds up plans incrementally, modifying them opportunistically as new information arrives. The work is an illustration of the application of blackboards to planning to exhibit flexible behavior, and is not a general-purpose planning framework itself.

The work on BB1 control planning [Hayes-Roth *et al.*, 1986; Johnson and Hayes-Roth, 1987; Garvey *et al.*, 1987] has followed up on some of the ideas from the OPM work. The emphasis is on flexible and general use of control plans at the architectural level. As in OPM, planning is incremental, with plan execution allowed to begin while the plan is being elaborated. The approach relies on skeletal plan refinement, using a set of alternative predefined skeletal plans.

## 7.6 Integrated execution and planning

A number of researchers have integrated planners into a robot control architecture to achieve multi-level planning for real-time problems [Lyons and Hendriks, 1992; Hanks and Firby, 1990; Gat, 1992; Myers and Wilkins, 1994]. All of these add a planner to an existing reactive robot control architecture: Lyons and Hendriks build on the RS reactive model [Lyons, 1990], Hanks and Firby as well as Gat build on the RAP reactive model [Firby, 1989], and Myers and Wilkins combine the PRS system [Georgeff and Lansky, 1987] with the SIPE planner [Wilkins, 1988]. In each case, the reactive controller is designed to handle the real-time issues that arise. The planner is run concurrently with the reactive controller to aid in the robot control problem, but is not a real-time program itself. Rather, the results of the planner are used when they become available—assuming that the world has not changed and invalidated the plan—to guide the reactive controller.

The reactive controller may use the planner in a variety of ways: the planner may restrict the space of possible actions that the robot controller may follow [Lyons and Hendriks, 1992]; the planner may generate goal orderings for the robot controller [Gat, 1992]; or the planner may generate actions to fill in situations for which there are no reactions good enough [Hanks and Firby, 1990; Myers and Wilkins, 1994]; or the planner may replan when conditions or subgoals fail [Myers and Wilkins, 1994]. But in all of these cases, the planner itself is not sensitive to real-time issues, but rather works in isolation from the world and with no representation of the temporal information available about the world. This leaves the planner susceptible to the same sorts of problems that motivated the work in this thesis.

# Chapter 8

# Discussion

In this thesis, we have presented a method for building plans in a dynamic environment. In this chapter we discuss directions that the work suggests for further exploration. We close with a brief review of the work.

## 8.1 Issues

With a complex problem such as real-time planning in the real world, we cannot expect one piece of work to resolve all remaining questions. That is not the general nature of science, nor certainly is it of AI. In this section we discuss some of the issues that have arisen from the work and some directions where it could lead.

### 8.1.1 Incorporating pre-computed plans

In our initial description of the problem, we emphasized that this approach was designed to fill in the void where pre-stored plans could not cover. This is not to denigrate pre-stored plans. If the planning agent could retrieve a satisfactory plan for a situation without resorting to planning, that would be preferable. Our argument is that pre-stored plans by themselves are not adequate for a real-world planning agent.

One obvious extension of the work would be to incorporate pre-stored plans when they are available, resorting to planning from scratch only when we indeed reach the frontiers of our knowledge. A planning system that could draw on both pre-stored plans and synthesized plans would potentially be able to span a spectrum of domains, from those where the agent is endowed with a wealth of experience to those where the agent is lacking in all but the most basic of information.

We will sketch a basic outline of how pre-computed plans could be integrated into our planner without affecting the basic underlying planning skills. Many of the details will no doubt be more complicated to realize than they are to describe in general.

A pre-stored plan will be a sequence of states and actions, where the states and actions are the same as in the normal planning search space. The only difference is that the work of searching through the search space is saved. We will generalize the idea a small amount and add the ability for the sequence to be augmented with alternative branches. In other words, a pre-stored plan may be seen as a portion of a search space stored away for re-use.

Since we are working within a temporal representation, the pre-stored plan will need to be represented temporally as well. Since absolute times will be next to useless, the pre-stored plans will need to have times relative to some starting reference point. This view of pre-stored plans make them sound like a very complex planning operator, which is essentially how we consider them. In fact, one could store an entire pre-computed plan in a single macro-operator. The problem with that is that the longer the plans get (and hence the more search they will save), the less likely they are to have all their conditions satisfied in a new situation.

We adopt the approach of treating the plan as a set of operators that we are trying to add into the plan. If we decide to add the conglomerated operator, we will add the initial operator into the plan if and when it is applicable. We then pick off the remaining operators and add them into the plan as long as they are applicable. Note that with our generalization of branching plans, we may have some branches that will transfer and some that will not. What we end up with is a highly directed search deep into the search space. If the plan transfers beneficially, we will find that the worth of the plan has increased dramatically.

A particular place where pre-stored plans would be useful is in implementing abstract operators. We would like to be able, given enough information, to mimic the pseudo-abstraction of NOAH and its derivatives, where an abstract operator is accompanied by the prescription for how to expand it into more specific operators. In our approach, we could have pre-stored plans for achieving the intermediate goals generated by the abstract plan. The plans, if they proved to be applicable in the current planning situation (after all, the same intermediate goal may appear in more than one circumstance), would in effect provide an implementation of the abstract goals, while not robbing the basic planning method of its ability to explore alternatives in the case where the pre-stored plan is not the best possibility available.

The related issue of learning plans could also revolve around the intermediate goals. As in SOAR, where chunks are learned dependent on the particular results returned from a subgoal, we propose learning plan fragments as they achieve particular intermediate goals. Since multiple intermediate goals may be affecting the planning process, the plan learner would need to unravel the operator dependencies into chains that achieve each of the goals, and store the relevant chain for use in other situations.

### 8.1.2 Generating abstractions automatically

Since the planning method relies heavily on abstraction, it is critical that the planner have at its disposal a good abstraction of the domain. The abstraction will determine the framework around which the plan is built.

Automatically generating abstractions is a field unto itself, generating theses for those who have chosen to attack it [Christensen, 1991; Knoblock, 1991; Unruh, 1993]. In general, the problem requires a careful analysis of the domain to identify abstractions that will be useful, although the work of [Unruh, 1993] provides an example of a dynamically-computed abstraction based on the context of the problem-solving episode.

The context-dependent approach for dynamically abstracting from a particular problem-solving episode is particularly intriguing, but that approach works in a bottom-up manner, which presents challenges about how to use the approach for resource-bounded planning and replanning.

### 8.1.3 Generating strategies automatically

The results of the analytical and empirical studies show that an optimal plan-expansion strategy can be found for a given problem. The analytical model serves as an approximation to the actual domain, and it lends itself to numerical methods to find the optimal strategy. So far this technique exists only outside the actual planner.

Certainly the strategy could be chosen ahead of time by analyzing the domain and using the mathematical model to predict the best approach. But a more interesting use of the technique would have the planner automatically analyze the domain and adjust the strategy to track the optimal strategy for the particular part of the plan where the planner is at that moment.

### 8.1.4 Richer temporal representations

There is a tension between complete and efficient representations in much of AI, not just in this work. We described our method in terms of a more sophisticated temporal representation than the one actually used for the implementation.

The ideal would be to have a language that could provide full expressive power but could also cut corners and lose completeness under resource bounds. In other words, we would like some sort of "anytime" representation that is sensitive to the amount of time available for its computations. The details of such an approach are unclear.

## 8.2 Conclusion

In this thesis, we have presented a planner that can build plans under resource bounds in dynamic environments. In particular, we have shown:

- The planner represents the dynamic and continuous nature of information and events in the real world. The planner is built upon a foundation of a temporal representation, which maintains information about the world over time intervals.

- The planner continually incorporates new information into its model of the world and adapts its plan accordingly. The planner views the plan as a dynamic data structure that is undergoing constant modification, both from the planner itself and from external forces, such as the world or other reasoning components. The planner maintains records of the dependencies that parts of the plan have on world states. As the states change, the plan is updated using the dependencies. In addition, new branches may be added to the plan as the states change.

- The planner operates within arbitrary and changing time bounds, building the best plan it can within the amount of time available. The plans are built incrementally at multiple levels of abstraction. As deadlines approach, the planner will commit to its best plan. Given more time, the planner will produce a better, more complete plan before committing to it. As the plans at the various levels of abstraction are elaborated, they provide an increasingly complete and accurate approximation to a full plan.

We have presented an analytical model of our planning method. Using the model, we can find the benefits of abstraction, and we can find an optimal strategy for expanding the plans at the various levels of abstraction. Our empirical results validate the method and the analysis.

# Bibliography


[Allen and Koomen, 1983] J. F. Allen and J. A. Koomen. Planning using a temporal world model. In *Proceedings of IJCAI-83*, pages 741–747. IJCAI, 1983.

[Allen, 1983] J. F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, 1983.

[Alterman, 1988] R. Alterman. Adaptive planning. *Cognitive Science*, 12:393–421, 1988.

[Ash and Hayes-Roth, 1990] D. Ash and B. Hayes-Roth. Temporal representations in blackboard architectures. Technical Report KSL-90-16, Knowledge Systems Laboratory, Stanford University, 1990.

[Boddy and Dean, 1989] M. Boddy and T. Dean. Solving time-dependent planning problems. Technical Report CS-89-03, Brown University Department of Computer Science, February 1989.

[Bratman *et al.*, 1988] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. Technical Report Technical Note 425R, SRI International, 1988.

[Chapman, 1987] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[Christensen, 1989] J. Christensen. Abstraction and search control in planning. PhD thesis proposal, May 1989.

[Christensen, 1991] J. Christensen. *Automatic abstraction in planning.* PhD thesis, Stanford University, 1991.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of AAAI-88.* AAAI, 1988.

[Dean, 1985] T. Dean. *Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving.* PhD thesis, Yale University, October 1985.

[Drummond *et al.*, 1993] M. Drummond, K. Swanson, J. Bresina, and R. Levinson. Reaction-first search. In *Proceedings of IJCAI-93*, 1993.

[Drummond, 1989] M. E. Drummond. Situated control rules. In *Proceedings of the first international conference on Principles of Knowledge Representation and Reasoning*, pages 103–113, 1989.

[Durfee and Lesser, 1986] E. H. Durfee and V. R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of AAAI-86*, pages 58–64. AAAI, 1986.

[Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Firby, 1989] J. R. Firby. Adaptive execution in complex dynamic worlds. Technical Report 672, Department of Computer Science, Yale University, 1989.

[Garvey *et al.*, 1987] A. Garvey, C. Cornelius, and B. Hayes-Roth. Computational costs versus benefits of control reasoning. In *Proceedings of AAAI-87*, pages 110–115, 1987.

[Gat, 1992] E. Gat. Integrating planning and reasoning in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of AAAI-92*, pages 809–815. AAAI, 1992.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of AAAI-87*, pages 677–682. AAAI, 1987.

[Ginsberg, 1989] M. Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4):40–44, 1989.

[Ginsberg, 1994] M. Ginsberg. Approximate planning. *Artificial Intelligence*, 1994.

[Hammond, 1988] K. J. Hammond. Case-based planning. In J. Kolodner, editor, *Proceedings of Workshop on Case-Based Reasoning*, pages 17–20, May 1988.

[Hanks and Firby, 1990] S. Hanks and R. J. Firby. Issues and architectures for planning and execution. In K. Sycara, editor, *Innovative Approaches to Planning, Scheduling, and Control.* Morgan Kaufmann, 1990.

[Hayes-Roth and Hayes-Roth, 1979] B. Hayes-Roth and F. Hayes-Roth. A cognitive model of planning. *Cognitive Science*, 1979.

[Hayes-Roth *et al.*, 1986] B. Hayes-Roth, B. Buchanan, O. Lichtarge, M. Hewett, R. Altman, J. Brinkley, C. Cornelius, B. Duncan, and O. Jardetzky. PROTEAN: Deriving protein structure from constraints. In *Proceedings of AAAI-86*, pages 904–909, 1986.

[Hayes-Roth *et al.*, 1987] B. Hayes-Roth, A. Garvey, Jr. Johnson, M. V., and M. Hewett. A modular and layered environment for reasoning about action. Technical Report KSL-86-38, Knowledge Systems Laboratory, Stanford University, 1987.

[Hayes-Roth, 1985] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[Horvitz, 1987] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. Technical Report KSL-87-29, Knowledge Systems Laboratory, Stanford University, 1987.

[Johnson and Hayes-Roth, 1987] V. Johnson and B. Hayes-Roth. Integrating diverse reasoning methods in the BB1 blackboard control architecture. In *Proceedings of AAAI-87*, pages 30–35, 1987.

[Knoblock, 1991] C. A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. PhD thesis, Carnegie Mellon University, May 1991.

[Korf, 1987] R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[Korf, 1990] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2–3):189–211, 1990.

[Laffey *et al.*, 1988] T. J. Laffey, P. A. Cox, J. L. Schmidt, S. M. Kao, and J. Y. Read. Real-time knowledge-based systems. *AI Magazine*, 9(1):27–45, 1988.

[Laird *et al.*, 1987] J. E. Laird, P. S. Rosenbloom, and A. Newell. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[Lyons and Hendriks, 1992] D. M. Lyons and A. J. Hendriks. A practical approach to integrating reaction and deliberation. In J. Hendler, editor, *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, pages 153–162, June 1992.

[Lyons, 1990] D. M. Lyons. A process-based approach to task-plan representation. In *IEEE International Conference on Robotics and Automation*, May 1990.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634–639. AAAI, 1991.

[McDermott, 1982] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Mulder and Braspenning, 1992] F. W. Mulder and P. J. Braspenning. Reactive multigranular planning: An analysis. Draft, 1992.

[Myers and Wilkins, 1994] K. L. Myers and D. E. Wilkins. Taskable reactive agents. Technical report, SRI AI Lab, 1994.

[Newell and Simon, 1972] A. Newell and H. A. Simon. *Human Problem Solving*. Prentice-Hall, Inc., 1972.

[Nilsson, 1992] N. Nilsson. Toward agent programs with circuit semantics. Technical Report STAN-CS-92-1412, Stanford University, January 1992.

[Nilsson, 1994] N. J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[Pardee *et al.*, 1989] W. J. Pardee, M. A. Shaff, and B. Hayes-Roth. Intelligent control of complex materials processes. In *Proceedings of the Third Annual Workshop on Blackboard Systems*, pages 173–189. AAAI, 1989.

[Penberthy, 1993] J. S. Penberthy. *Planning with Continuous Change*. PhD thesis, University of Washington, 1993.

[Pollack, 1992] M. E. Pollack. The uses of plans. *Artificial Intelligence*, 57:43–68, 1992.

[Rymon *et al.*, 1992] R. Rymon, B. L. Webber, and J. R. Clarke. Progressive horizon planning: Planning exploratory-corrective behavior. Draft, 1992.

[Sacerdoti, 1974] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Sacerdoti, 1977] E. D. Sacerdoti. *A Structure for Plans and Behavior*. Elsevier, 1977.

[Schoppers, 1987] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of IJCAI-87*, pages 1039–1046. IJCAI, 1987.

[Shoham, 1989] Y. Shoham. Time for action: On the relation between time, knowledge, and action. In *Proceedings of IJCAI-89*, pages 954–959,1173. IJCAI, 1989.

[Simmons, 1986] R. Simmons. “commonsense” arithmetic reasoning. In *Proceedings of AAAI-86*, pages 118–124. AAAI, 1986.

[Sowa, 1984] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[Tate, 1977] A. Tate. Generating project networks. In *Proceedings of IJCAI-77*, pages 888–893. IJCAI, 1977.

[Unruh, 1993] A. Unruh. *Using automatic abstraction for problem-solving and learning.* PhD thesis, Stanford University, April 1993.

[Vere, 1981] S. A. Vere. Planning in time: Windows and durations for activities and goals. Technical report, California Institute of Technology Jet Propulsion Laboratory, 1981.

[Washington and Hayes-Roth, 1989] R. Washington and B. Hayes-Roth. Input data management for real-time AI systems. In *Proceedings of IJCAI-89*, pages 250–255. IJCAI, 1989.

[Wilkins, 1988] D. E. Wilkins. *Practical Planning.* Morgan Kaufman, 1988.