Solving Time-Dependent Problems: A Decision-Theoretic Approach to Planning in Dynamic Environments

Mark Steven Boddy

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-91-06
May 1991

Solving Time-Dependent Problems: A Decision-Theoretic Approach to Planning in Dynamic Environments

by
Mark Boddy

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University.

May 1991

Abstract

Controlling a robot involves making decisions that modify its behavior. Making good decisions may require time-consuming computation. Changes in the environment over time affect when this computation can be done (e.g., after obtaining the necessary information), and when a result is useful (e.g., before some event occurs). This sensitivity to when computation is performed and when decisions are made is what makes these problems "time-dependent." A controller with more than one decision to make must trade off computation time, based on the expected effect on the system's behavior. We call the resulting meta-level scheduling problem a "deliberation-scheduling" problem.

We have developed a framework for doing deliberation scheduling called "expectation-driven iterative refinement". This framework requires that the decision procedures to which time is allocated be "anytime algorithms," which can be interrupted at any point to return an answer whose expected utility increases with computation time.

We have conducted a theoretical and empirical investigation of the application of expectation-driven iterative refinement to time-dependent problems. We discuss the problem of finding or generating anytime decision procedures and gathering useful expectations on their behavior. The difficulty of doing deliberation scheduling depends in large part on the decision model to be used in computing expectations on the system's behavior. We show that for many time-dependent problems, optimal deliberation scheduling for a reasonably complex decision model is likely to be computationally expensive. We suggest some ways in which the decision model can be restricted so as to simplify deliberation scheduling. In particular, we discuss generating an explicit "deliberation schedule" on the occurrence of a restricted set of significant events.

Empirical results obtained for three different examples demonstrate that this approach can provide computationally inexpensive deliberation scheduling with an expected utility only slightly less than for a more complete (and computationally intractable) decision model. These examples also serve to elucidate some of the issues involved in generating solutions to time-dependent problems, and support our contention that expectation-driven iterative refinement is a useful tool for providing these solutions.

Acknowledgements

First and foremost, I would like to thank Tom Dean for serving as my advisor, with all that entails in terms of providing guidance, ideas, encouragement, and the occasional much-needed kick in the rear (not to mention reading an enormous number of buggy drafts). I don't think anyone could have done it better. Along the way, he taught me how to write a research paper, in addition to showing me the methods and rewards of research in computer science. Our collaborations have been rewarding. I hope to continue them for a long time to come.

The other members of my committee are Eugene Charniak and Stanley Rosenschein. They took time from very busy schedules to read and comment on this thesis with a very short turnaround time, for which I am duly grateful. Eugene Charniak has had a lot to do with making Brown such a good place to work. His steadfast insistence that artificial intelligence can be a *science* is one of the most valuable lessons I have learned at Brown. Stan Rosenschein and I have had some fascinating discussions on a wide range of research topics. His approach to research has influenced both the problems I look at and the kinds of solutions I look for.

The Brown Computer Science Department has been a pleasant refuge for the last five years. My research benefited substantially from feedback from the AI group at Brown, particularly from discussions with Ken Basye, Robert Goldman, Keiji Kanazawa, Kate Sanders, and Lynn Stein. This work has also been influenced by conversations with Jack Breese, Mark Drummond, Steve Hanks, Eric Horvitz, Leslie Kaelbling, Victor Lesser, Steve Minton, Stuart Russell, Steve Smith, and Jack Stankovic, among others.

Contents

List of Figures

Chapter 1

Time-Dependent Problems

1.1 Introduction

We are interested in constructing controllers for autonomous systems interacting with complex and dynamic environments (i.e., robots). By "dynamic environment," we mean an environment whose state evolves over time as events occur and processes progress. Those events directly under the robot's control we call *actions*. Change in the environment over time means that an action's effects may depend on when it is taken. Feedback from the environment may produce situations in which some particular action is called for. For instance, a robot moving down a crowded hallway must pay close attention in order to avoid moving obstacles and perhaps simply to avoid running into one of the walls.

Consider a stationary robot assigned the task of recognizing and rerouting objects on a moving conveyor belt. Suppose that the robot's view of the conveyor is partially obscured by a partition, and that someone on the other side of this partition places objects on the conveyor at irregular intervals. The robot's task is to identify each object, using data from a vision system, and then to dispose of it properly. Identifying objects is computationally intensive, and the longer the robot spends analyzing an image, the more likely it is to make a correct classification. The actions required to dispose of objects may also take time to complete, and must be done before the object falls off the end of the conveyor. On the basis of such factors as the cost of an incorrect identification, the change in the probability of making a correct identification with increasing computation time, and the spacing of objects on the conveyor, the robot must decide how much time to spend on identifying and deciding on the disposal of each object. If the robot is to make effective use of the time available, it should be prepared to make identifications and decisions when very little time is available, as well as to take advantage of situations in which more time is available than usual.

The computation required to identify and dispose of the objects on the belt we call *deliberation*. A great many different kinds of deliberation might be useful for a robot, including analyzing sensor data, planning to achieve some goal, projecting the effects of a set of intended actions, or planning a path through unknown territory, to name a few. All these kinds of deliberation have in common that they may require a significant amount of computation to produce a final answer.

Any robot, no matter how large and sophisticated, has finite computational resources. The need to synchronize the robot's behaviour with other events and the limited computational resources available for deliberation force the robot to trade off the benefits to be gained from deliberation against the cost of delaying action. If the robot has more than one thing to think about at any one time, it must also make tradeoffs in how much time is allocated to each task. Making these allocation decisions we call *deliberation scheduling*.

A *time-dependent problem* is one in which a system (e.g., the robot described above) is faced with a combination of task and environment such that deliberation is expensive, and how appropriate or well-advised an action is depends on when the action is taken. The system's behavior is evaluated according to a *utility function* supplied as part of the problem specification. For the example above, this utility function could be as simple as summing the number of incorrect identifications and the number of objects that fell off the end of the belt.

A *solution* to a time-dependent problem is a strategy for determining when and on what the system should deliberate, and on what, and when it should act (i.e., a strategy for deliberation scheduling). We have defined a framework for providing solutions for time-dependent problems called *expectation-driven iterative refinement*. The central assumption made in this framework is that all deliberation is performed using *anytime decision procedures*. Anytime decision procedures return an answer for any allocation of time, and the answer returned is expected to improve as additional time is allocated to that procedure.[1] Anytime decision procedures are useful in dynamic and uncertain environments for two reasons. First, they allow the system to make effective use of however much time is available, across a wide range of possibilities. Second, they can be interrupted unexpectedly (e.g., by the occurrence of some event urgently requiring a response), and still provide a reasonable answer. Deliberation scheduling in this framework involves allocating time to one or another of the available anytime decision procedures, on the basis of current information and expectations. These expectations include *performance profiles* that encode expectations on the effects of allocating time to specific anytime decision procedures.

[1] Anytime decision procedures implement *anytime algorithms* [Dean and Boddy, 1988, Boddy and Dean, 1989].

Using expectation-driven iterative refinement to construct solutions to time-dependent problems restricts a very general problem (deciding what to think about and when to act) to a somewhat simpler one: allocating the next increment of time to one of a set of anytime decision procedures. The rest of this chapter places this work in context, briefly discusses some related work, and describes the organization of the thesis. The rest of the thesis consists of a theoretical and empirical investigation of the solution of time-dependent problems.

1.2 High-Level Planning and Low-Level Control

The initial motivation for the work described in this thesis was the desire to build a robot that was both responsive to the world around it and capable of abstract problem solving to achieve a set of goals (i.e., capable of *planning*).

Constructing systems for abstract problem solving and planning has been an active research area in artificial intelligence (AI) for some time (e.g., [Green, 1969]). Many of these systems employ world models that abstract away the passage of time and the occurrence of events other than the problem-solver's own actions (e.g., the BLOCKSWORLD) [Fikes and Nilsson, 1971, Sacerdoti, 1977, Tate, 1977]. More recently, planners have been built that can take into account the passage of time as the system acts and the possibility of other events occurring, e.g., [Wilkins, 1984, Vere, 1983, Dean *et al.*, 1987].

When researchers tried using these planners on robots, they ran into problems resulting from the "top-down" nature of the resulting architecture. The kind of information available from sensors is very different from the symbolic representations the classical planners manipulate. There is also a considerable gap between abstract actions of the form "(put-on Block1 Block2)" and the kinds of commands that the hardware can accept, which are more typically specifications of velocity, position, or other simple parameters.

The most significant of these difficulties from our point of view results from the fact that planning systems operating in the simulated environments in which almost all of them were developed could do their planning "off-line" relative to the passage of time in the environment. The environment could also be made free of agents or processes other than the planner. However, robots operating in the real world cannot stop the passage of time while they decide what to do. Nor can they afford to ignore the course of events around them. A robot that stops in the middle of the street to think things over may not get run over—for a little while. Any time-consuming computation the robot does is at the cost of something else it could be doing with the same time, and is also likely to be interrupted by some unexpected event.

Some of the first attempts to deal with these difficulties led to a series of systems in which abstract goals or behaviors were compiled into low-level representations [Chapman and Agre, 1987, Rosenschein and Kaelbling, 1987, Kaelbling, 1988, Brooks, 1985]. These "reactive systems" are typically implemented as sets of hierarchically organized modules that interact in fixed ways. The problem with reactive systems is that flexibility and expressive power are sacrificed to guarantees of responsiveness. Compiling a planner into a gate-level description yields either a huge description or a "planner" with severely limited capabilities. The problems encountered in implementing abstract reasoning in reactive systems suggest that we need two very different types of computation: responsive low-level control of the robot's interaction with its environment and a high-level capability for abstract reasoning (planning).

We are left with the problem of synchronizing the high-level reasoner with the lower levels controlling the robot. Hanks and Firby accomplish this by interfacing a fairly traditional high-level planner with low-level reactive behaviors used as primitive actions [Firby, 1987, Hanks and Firby, 1990]. The Phoenix project [Cohen *et al.*, 1988, Howe *et al.*, 1990] uses an agent architecture that integrates planning and acting by implementing a two-level system: the higher-level "cognitive" component modifies the agent's behavior at irregular intervals, while a "reflexive" system provides lower-level but more timely reactions to changes in the environment. Hayes-Roth's BB1 [Hayes-Roth *et al.*, 1989] is a blackboard architecture for systems that are responsive to environmental changes, but perform abstract reasoning when time permits. The *Procedural Reasoning System* (PRS) [Georgeff *et al.*, 1987] provides what amounts to a programming language for complex robot architectures. PRS provides scheduling and arbitration among user-defined modules, called *KA*s, that can be implemented at any level of abstraction.[2]

However, none of this work directly addresses the problem of deciding what abstract reasoning should be performed when. In these systems, high-level computations typically continue until they are complete unless interrupted by some urgent request from the lower level(s). Some of these systems are flexible enough that deliberation scheduling could be added to them. For example, we could add a deliberation-scheduling KA to a PRS system (they already have "metalevel" KAs [Georgeff and Ingrand, 1989]). But this does not answer the question of what these meta-level KAs should look like or what kinds of computation they are used to control. That is where the work in this thesis comes in.

One implemented system does deliberation scheduling for abstract reasoning in an extremely time-constrained domain: Andersson's controller for a ping-pong-playing

[2] In other words, a "velocity-control" KA and a "goal-achieving" KA can be defined in the same system.

robot [Andersson, 1988] solves special cases of exactly the kind of problems we are interested in. In particular, he deals with the issues of incremental modification of plans as more information becomes available, and computation of the best answer possible in the time available. He also integrates a simple rule-based system for symbolic reasoning with the complex control laws necessary to drive a robot arm close to the limits of its design parameters. It is not clear to what extent the techniques employed could be generalized to other problems, but they are quite successful in this domain.

1.2.1 Planning for Time-Dependent Problems

Several people have recently proposed planning methods explicitly designed to be implemented as anytime decision procedures. Elkan [1990] employs nonmonotonic logic to do "incremental planning." The planning process is incremental in the sense that the plan initially generated is valid only under certain assumptions. As more time is spent on planning, these assumptions are tested and either rejected (in which case the system must construct another plan) or shown to be true. Drummond and Bresina [1990] have constructed an architecture for planning and acting in which the upper levels iteratively modify a set of *situated control rules* for the lowest level to follow. [Zweben *et al.*, 1990] present an anytime algorithm for rescheduling, which is a useful planning capability in many domains (e.g., [Dean *et al.*, 1987, Wilkins, 1989]). The enormous advantage to these proposals from our point of view is that these planning methods can be used directly in a solution of a time-dependent problem constructed using expectation-driven iterative refinement. All we need to do is gather some statistics on their performance in a particular domain.

In a different vein, [Kraus *et al.*, 1990] present a planning system in which the time spent planning is explicitly accounted for using *step-logics* in which the time required for an inference is associated with that inference. They claim that *all* time, including the time required for meta-reasoning, is accounted for in this system. More precisely, it is a system in which if there are n levels of meta-reasoning explicitly represented in the theory, the $n+1$st level is the inference machinery itself, which is keeping track of the time required for each inference. One question that must eventually be addressed is whether there is any level at which the scheduling provided by the inference machinery is appropriate.

1.3 Deliberation Scheduling and Decision Theory

Deciding how to allocate scarce resources for deliberation has been addressed under many names. "Meta-reasoning," "resource-bounded reasoning," and "bounded rationality" are some of the terms that have been used, in addition to our own term "deliberation scheduling." Whatever the term, the aim is the same: to provide a system that can use expectations on the results of its decisions to allocate deliberation time.

1.3.1 Why Use Decision Theory?

Much of the work on deliberation scheduling has employed decision theory.[3] Probability and decision theory are methods for dealing with uncertain information and outcomes on the basis of a small set of axioms concerning *rational* behavior [Raiffa, 1968]. If you accept the axioms, decision theory provides a powerful set of tools for modelling and evaluating decision making under uncertainty.

The kinds of problems we are interested in can involve several sources of uncertainty, including the future occurrence of events, sensor limitations, and incomplete knowledge of the effects of actions. In addition, we have only limited knowledge of the effects of allocating time to the decision procedures used for deliberation. Probability is a simple, consistent, and well-understood way to represent and reason about these uncertainties [Pearl, 1988, Good, 1976]. In addition, there is a large body of standard techniques for deriving probability distributions from statistical data [Green and Margerison, 1978]. Since the solutions we provide and the expectations we use are both grounded in the real world (or at the least in some simulated environment), these techniques will be useful.

Deliberation scheduling is accomplished by a sequence of decisions made by the system as time passes, events happen, and new information becomes available. These allocation decisions influence the system's behavior. Preferences on various outcomes or sequences of events are encoded in a *utility function* U. We treat the value of U as a random variable dependent on the actual outcome of a decision or a sequence of decisions and calculate its expected value. The main point we take from decision theory is that calculating the expected utility resulting from the decisions made provides a consistent basis for evaluating decision making under uncertainty [Raiffa, 1968, von Neumann and Morgenstern, 1947]. This is how we evaluate strategies for deliberation scheduling: an optimal deliberation scheduler maximizes the expected utility of its decisions, given what it knows about the environment and the effect of those allocations.[4]

[3] Not all of it, however. See for example [Shekhar and Dutta, 1989].

[4] For a more detailed discussion of the theoretical and practical advantages of decision theory as a

1.3.2 Discrete Deliberation Scheduling

A line of work running back to [Simon and Kadane, 1975] views deliberation time as being quantized into chunks of fixed, though not necessarily constant, size. Sproull [1977] applies this model to a system for planning travel itineraries, though he is more concerned with the use of decision theory to guide planning choices.

In [Etzioni, 1989], the deliberative chunks are called "methods." Each method has a fixed cost. The object is to maximize the expected reward from applying a sequence of methods, given expectations on the reward for using a particular method for a particular goal and a limit on the total method cost incurred. Etzioni shows that problems involving multiple methods applied to a single goal can be solved inexpensively, but that allowing multiple goals makes the problem intractable.

In [Russell and Wefald, 1989b], the computation to be controlled is modeled as a sequence of "inference steps" leading to the performance of an action. The utility of a particular inference step is defined in terms of its effect on the eventual choice of action and the state in which the action is executed. This model is applied to game-tree search, in which a deliberative "chunk" corresponds to expanding the children of a node [Russell and Wefald, 1989a] or just evaluating a single additional child of some node [Russell, 1990]. This work assumes that the cost of inference can all be pushed into a "time cost" that is a function only of the number of inferential steps and actions taken so far. This works for modeling something like a chess game, where there is a limit only on the total time spent deliberating, but does not work well for a more complicated set of deadlines (e.g., the robot and the conveyor belt).[5]

One of the problems with allocating deliberation time in discrete increments is that it is easy to wind up with a combinatorial problem [Einav, 1990]. This was part of our motivation in adopting the use of anytime decision procedures. If continuous behavior is a reasonable approximation, then deliberation scheduling is no longer a combinatorial problem, though it may still be computationally expensive for other reasons. Another problem is that the "grain-size" of deliberation scheduling is fixed by the size of the increments. As we show in Chapter 3, it is sometimes possible to allocate large blocks of time at once when the benefit of running one anytime decision procedure can be shown to be greater than for any other.

1.3.3 Flexible Computations

An alternative approach is to assume that deliberation is performed by anytime decision procedures [Dean and Boddy, 1988] (also called *flexible computations* in [Horvitz,

basis for resource-bounded reasoning, see [Horvitz, 1987]

[5]For a more detailed discussion of the work presented in this section, see [Dean, 1989].

1987]), and that time can be allocated in values drawn from a continuous range (this is inevitably an approximation). Our motivations for using anytime decision procedures and the consequences of that choice are discussed in detail in Chapter 2. Briefly, their advantages include an ability to make use of any amount of time available, robust behavior in the presence of unexpected interruption, and simplifying optimal deliberation scheduling.

Horvitz has addressed a number of interesting theoretical issues in the use of anytime decision procedures. These include the difficulty of providing a single characterization of the "utility" associated with a given procedure (see also Section 2.5.3), and how differing time costs can affect the optimal choice of a decision procedure in a particular situation [1988]. Horvitz has so far concentrated on the theoretical problems involving deliberating about a single task. In contrast, we have pursued a more empirical course in trying to find approximate solutions to the kind of deliberation-scheduling problems that arise in controlling robots.

1.3.4 Applications

We have already mentioned several application areas in which the decision-theoretic control of reasoning has been explored, including planning [Elkan, 1990, Drummond and Bresina, 1990], game-tree search [Russell and Wefald, 1989a, Hansson and Mayer, 1988], medical decision making [Horvitz, 1988], and robot control [Dean and Boddy, 1988, Boddy and Dean, 1989]. In addition, decision-theoretic deliberation scheduling frameworks have been proposed for computer vision [Levitt *et al.*, 1988], industrial control [Agogino *et al.*, 1988, Ramamurthi and Agogino, 1988], probabilistic inference using belief networks [Breese and Horvitz, 1990], sensor fusion [Hager, 1988], and queries in deductive databases [Smith, 1989].

Breese and Fehling [1988] provide an abstract architecture for the decision-theoretic control of an autonomous agent, including reasoning about tradeoffs among planning, acting, and gathering information. [Horvitz *et al.*, 1988] provide a survey of the application of decision theory to AI more generally, and points out some areas in which further research might be most fruitful.

1.4 Related Work in Other Areas

The kinds of optimization problems we address are similar to problems addressed in a number of other disciplines. As a result, many of the techniques we apply to time-dependent problems are drawn from or are similar to techniques in use in those disciplines.

1.4.1 Real-Time Systems and Scheduling

The usual aim of a "real-time" system is to guarantee that some task or set of tasks will be completed within a fixed deadline. This deadline may be a bound on how long the system can take to respond to a given stimulus (a response time). Computer systems used to control manufacturing processes, for example, are specially designed to guarantee response within some time bound. As the environments to which these systems are applied grow in complexity, interest has grown in applying artificial intelligence techniques, including expert systems [Mok, 1989], and planning [Laffey *et al.*, 1988]. This work, however, addresses only part of the problem we are interested in: guaranteeing a maximum response time is not sufficient. In order to use its resources efficiently, the system must make appropriate use of the time available across a wide range, including when there is more time available than usual.

More recent work in the real-time-systems community has gone beyond simple guarantees in various ways, including using models for tasks similar to anytime decision procedures [Shih *et al.*, 1989] and meta-level control that modifies scheduling behavior based on the current situation [Ramamritham *et al.*, 1987, Locke, 1986]. The metrics used to evaluate these schedulers can be simple counts of the number of deadlines violated (Ramamritham), weighted sums of how much more time could have been spent (Shih), or more complex functions of how far the completion time of the task was from the desired time (Locke). For the kinds of problems we are interested in, these metrics are insufficient—there are significant dimensions to system performance beyond when a task is completed. However, it is clear that real-time researchers are beginning to address the same kinds of problems as we are interested in, from a somewhat different perspective (see, for example, [Stankovic, 1988]).

1.4.2 Control Theory

The canonical control problem starts with a *controller* (system) and a *plant* (environment): the controller can affect the evolving state of the plant in fixed ways and has access to limited information about that state [Dorf, 1989, Bollinger and Duffie, 1988]. The object is to define and implement a *control law* governing the behavior of the controller such that some metric on the evolving states of the system (e.g., an instantaneous error signal, or some quantity integrated over time) is maximized, minimized, or kept within a given bound. This is almost precisely how we have defined time-dependent problems.

The kinds of controllers generated in control theory have some attractive features. For instance, it is sometimes possible to establish properties of *stability* or *convergence*. These properties ensure that within some finite time after the system has started, or

after a change in the plant, the state of the plant can be predicted or at least determined to lie in some desirable subset of the state space.

However, some of the most useful results in control theory apply only to simple systems that can be described or approximated by sets of linear differential equations. The kinds of time-dependent problems we are interested in (e.g., high-level robot control) are more complicated. The equations defining the behavior of the system may be too difficult to solve (if they can be written down at all), and there may be too many distinct entries for table lookup to be a practical alternative. In the last few years, research in the field of intelligent control has begun investigating the design and construction of controllers for more complex problems (see the Proceedings of the IEEE Symposia on Intelligent Control). The research presented in this thesis addresses an issue that work on intelligent control will inevitably face: how to synchronize high-level, abstract computation with low-level control and changes occurring in the environment.

In discrete-event control [Ramadge and Wonham, 1989], the plant is a system evolving over time through the occurrence of a sequence of events. The controller's influence over the plant is limited to restricting what events may follow certain other events. A plant is *controllable* if the controller can restrict the sequence of events that occurs to a specified subset of the possible sequences of events. The modeling of plant and controller in this research is expressive enough to represent interesting classes of time-dependent problems, but the work so far is too abstract to be directly applicable.

1.4.3 Operations Research

Operations research as a discipline is primarily concerned with formulating and solving large optimization problems. Many of these problems have their roots in planning for large-scale industrial operations (e.g., capital investment or inventory management). The techniques employed for solving these problems include various method methods for solving linear programming problems [Papadimitriou and Steiglitz, 1981, Dantzig, 1963], and dynamic programming [Bellman, 1957]. Some of these techniques are relevant to the kinds of optimization problems we address in deliberation scheduling. If the available anytime decision procedures have piecewise-linear performance profiles, then constructing an optimal deliberation schedule for the time-dependent planning problems presented in Chapter 3 can be framed as a linear programming problem.

Dynamic programming, especially some of the more recent work on extending dynamic programming to stochastic problems (e.g., [Ross, 1983, Carraway *et al.*, 1989]), is more generally applicable to deliberation scheduling. Deliberation scheduling is a sequential decision problem of exactly the sort that dynamic programming was formulated to address. Unfortunately, while optimal deliberation scheduling can often be *framed* as a dynamic programming problem, much less frequently will we be able to

solve the problem. As we show for all three of the time-dependent problems we discuss, the number of alternatives that must be considered makes the problem computationally infeasible.

1.5 Thesis Organization

In this thesis, we explore the application of expectation-driven iterative refinement to time-dependent problems. This exploration includes both theoretical analysis (Chapter 2) and the investigation of specific classes of problems (Chapters 3, 4, and 5).

In Chapter 2, we address issues common to any attempt to solve time-dependent problems using expectation-driven iterative refinement. We provide a formal language for specifying time-dependent problems, and define the form of a solution to a time-dependent problem generated using expectation-driven iterative refinement. We discuss the problem of finding or generating anytime decision procedures and gathering useful expectations on their behavior. We describe some of the kinds of anytime algorithms presented in the literature of several disciplines, and suggest how to go about combining known anytime algorithms to create new ones suitable for more complex problems. It turns out that a great number of known algorithms, capable of solving a wide variety of problems, either are already framed as anytime algorithms or can easily be adapted to be anytime algorithms.

We also explore the problem of providing a deliberation scheduler. The difficulty of optimal deliberation scheduling depends in large part on the decision model specifying the information to be used in computing expectations on the system's behavior. We show that for any time-dependent problem, optimal deliberation scheduling for a reasonably complex decision model is likely to be computationally very difficult. As an alternative strategy, we suggest some ways to restrict the decision model so as to simplify deliberation scheduling. In particular we discuss generating an explicit *deliberation schedule* on the occurrence of a restricted set of significant events. Empirical results presented in the following chapters of this thesis show that for some problems, this approach can provide computationally inexpensive deliberation scheduling for which the expected utility is only slightly less than for a more complete (and computationally intractable) decision model.

Chapter 3 is a theoretical and experimental investigation of *time-dependent planning problems*: time-dependent problems involving absolute deadlines and independent responses to individual requests. For the static case, in which the set of requests is known from the outset and does not change, we provide an optimal deliberation-scheduling policy, implemented by a deliberation scheduler that constructs an explicit deliberation scheduleand then makes allocations according to that schedule as time passes. We show

that the deliberation scheduler used in the static case can be applied to the more general case in which new requests occur as time passes, with only a small loss in expected utility compared to a deliberation scheduler that is optimal for a decision model that includes expectations on the future occurrence of events. We also show that providing an optimal deliberation scheduler for this decision model is likely to be very difficult.

In Chapter 4, we introduce and analyze a class of time-dependent problems, known collectively as the *Robot Courier* problem, in which there are no hard deadlines and tasks can be reordered. For this class of problems, utility is defined strictly in terms of how long the robot takes to complete a set of tasks. We define appropriate anytime decision procedures for deliberation, and construct performance profiles encoding expectations of the effect on the robot's performance of time allocated to these anytime decision procedures. We show that for the Robot Courier, just as for time-dependent planning problems, optimal deliberation scheduling for any decision model general enough to take into account the dynamic and uncertain nature of the environment is likely to be very difficult. We provide deliberation schedulers for three increasingly complex decision models, all of which make use of expectations rather than distributions on the behavior of the anytime decision procedures, and all of which ignore the possibility that additional requests (tasks) may occur. We compare the performance of these schedulers to one another and to the performance of the robot doing no deliberation at all, in both static and dynamic domains. The comparison shows that the average loss in performance in dynamic situations (i.e., the cost of not including expectations on future events in the decision model) is fairly small.

In the process of comparing the various deliberation schedulers, we generate data of a sort that is likely to be very useful in the design of any time-dependent controller. We examine the change in performance as the relative cost of deliberation changes. This makes clear the tradeoff between performance and computational power (which costs money, or battery power, or has some other form of opportunity cost). The other point that becomes clear in this analysis is that the system exhibits the kind of "graceful degradation" we had intended. As deliberation becomes more and more expensive, the robot's performance smoothly approaches what it would be if no planning were done at all: there is no point at which the resources available fall below some critical level and the system simply quits functioning.

In Chapter 5, we present the third and last of the classes of time-dependent problems we have investigated: the *Shooting Gallery*. This class was specifically designed to add certain features to the deliberation-scheduling problem. Among these features are the use of computationally expensive and uncertain sensors, the presence of multiple tasks such that actions taken in order to complete one task affect the difficulty of completing subsequent tasks, and the introduction of conflicting and uncertain task deadlines.

Finally, in Chapter 6 we summarize the main points of earlier chapters, discuss some open problems, and point out some promising areas for future work extending some of the results presented here.

Chapter 2

Expectation-Driven Iterative Refinement

2.1 Introduction

We are interested in the interaction of a *system* with an *environment* (or, equivalently, a *controller* and a *plant*, though we will not use the latter terminology much). The state of the environment evolves over time as events occur or processes progress. Some of these events or processes are subject to the controlling system's direction; some are not. Those events under the system's control we call *actions*. Determining what actions the system should take as the environment evolves may involve substantial computation. We call this computation *deliberation*. Deliberation is itself an action, in the sense that it is under the system's control and takes time during which the state of the environment may change.

A *time-dependent problem* involves a combination of system and environment in which deliberation is expensive, and where how appropriate or well-advised an action is may depend on when the action is taken. This creates a situation where the system must trade off the benefits of deliberation against the cost of delaying action. Both the benefits and costs of deliberation are calculated relative to a measure of how good the system's interaction with the environment is. This measure (a *utility function*) is supplied as part of the problem specification.

In the specification of a time-dependent problem, we are given an environment, a system that interacts with that environment in specified ways, and a utility function on the results of that interaction. A *solution* to a time-dependent problem is a policy for determining when the system should deliberate (and on what) and when it should act. "Should" is defined in terms of expectations on the resulting behavior of the system and the environment. Specifically, we seek to maximize the expected utility of the system's

interaction with the environment. Specifying a time-dependent problem constrains the form this interaction might take (i.e., specifies a class of such interactions). An *instance* of a time-dependent problem further constrains the possible interactions between the system and its environment by specifying the initial state and the occurrence of events not under the system's control.

In this chapter, we provide a formal definition for time-dependent problems (Section 2.2). We then present a framework for generating and analyzing solutions to time-dependent problems, called *expectation-driven iterative refinement* (Section 2.3). The central assumption in this framework is that all deliberation is performed using *anytime decision procedures*. Anytime decision procedures are procedures that return an answer for any allocation of time, where the answer returned is expected to improve as additional time is spent on that procedure.[1] Deliberation time is allocated among the available anytime decision procedures using expectations on their effect on the system's behavior. These expectations can be compiled off-line and given to the system *a priori* or generated by the system as it continues to interact with its environment. In Section 2.4, we discuss how to find or generate anytime decision procedures and gather useful expectations on their behavior. In Section 2.5, we discuss how to construct a decision model for allocating deliberation time and a *deliberation scheduler* that maximizes expected utility with respect to that model.

2.2 Formal Statement of the Problem

In this section we provide a formal language for specifying time-dependent problems. We have two motives in doing this:

- A formal language fixes the structure of the problems we are addressing. We can refer to the class of time-dependent problems defined by that language, and draw some conclusions applicable to this class as a whole.

- A formal language provides a common ground for comparing apparently dissimilar problems.

We explore in this chapter the nature of deliberation scheduling for the class of time-dependent problems defined by the language given in this section.

A time-dependent problem consists of:

- A set of event types: $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$.

[1] Anytime decision procedures implement *anytime algorithms* [Dean and Boddy, 1988, Boddy and Dean, 1989].

- A set of *action* types $\mathcal{A} \subseteq \mathcal{T}$ defining the types of events under the system's control.

- A set of *histories* $\mathcal{H} \subseteq 2^{\mathcal{T} \times \Re}$. Each element h of $\mathcal{H}$ is a set of ordered pairs $\{< \tau, t > | (\tau \in \mathcal{T}) \wedge (t \in \Re)\}$. We call these pairs *events*. Given an event $e =< \tau, t >$, the notation $\mathsf{type}(e)$ denotes the event type τ and $\mathsf{occ}(e)$ denotes the time of occurrence t relative to some global clock.[2]

- A time t_0 such that $\forall h \in \mathcal{H}, \forall e \in h, t_0 \leq \mathsf{occ}(e)$.

- A set of *external states* $\mathcal{S}^{ext}$. Each external state completely describes the environment at one moment of time. If the system controls some mechanical apparatus (e.g., a robot), the state of that apparatus is included in the external state.

- A set of *internal states* $\mathcal{S}^{int}$. Each internal state is a pair of vectors $\langle V_s, V_d \rangle$. V_s is the *sensor data*. The value of V_s for any time t is dependent only on the external state at time t: deliberation has no effect on V_s. V_d is the *deliberation scratch-pad*: space used to maintain data structures that may result from deliberation. Deliberation is the only thing that affects V_d.

- A set of *initial states* $\mathcal{I}$. Each initial state i is an ordered pair $\langle s_{int}, s_{ext} \rangle$. s_{int} specifies the system's internal state at t_0; for example, s_{int} might include the system's *a priori* expectations on the occurrence of events. s_{ext} specifies the external state at t_0. The external state at any later time t is uniquely determined by s_{ext} and the events occurring between t_0 and t.

- A function $U : \mathcal{H} \times \mathcal{I} \rightarrow \Re$ that assigns utilities to the members of $\mathcal{H}$, given an initial state i.[3]

We model the interaction of the system and the environment as the evolution of a history over time. We define a real-valued parameter $\hat{t}$ representing "now" in this evolution: the value of $\hat{t}$ starts at t_0 and increases monotonically. An event e in a history *occurs before* a time t if $\mathsf{occ}(e) < t$. Events occurring before $\hat{t}$ *have occurred*. Events that have occurred cannot be altered by the system. The system may not *know* what has happened, but there is a fixed history of events with times of occurrence between t_0 and $\hat{t}$ that constrains the history that will eventually result. Only elements of $\mathcal{H}$ containing exactly the same events occurring before $\hat{t}$ are candidates for the actual course of events that eventually happens. We call this subset of $\mathcal{H}$ the set of *active histories*.[4]

[2] Events are durationless. To represent the occurrence of processes having a finite duration, we define events corresponding to the beginning and ending of those processes.

[3] Alternatively, U could be defined on the set of external states.

[4] An active history set is very similar to what McDermott calls a *chronset* [McDermott, 1982].

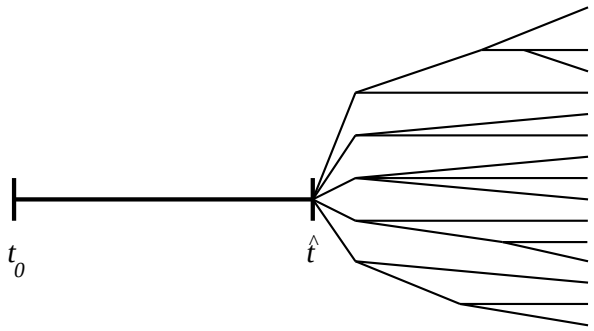

Figure 2.1: A set of active histories

For a given value of $\hat{t}$, we refer to the *current active histories* $\mathcal{H}_{\hat{t}}$. We say that the history that eventually results is *realized*, and denote this realized history by h^*. We can handle cases in which $\hat{t}$ increases without bound by taking h^* to be the invariant part of the current active histories at some time t, without insisting that that value represent "the end of time." As $\hat{t}$ increases, $\mathcal{H}_{\hat{t}}$ becomes smaller, since elements of $\mathcal{H}_{\hat{t}}$ containing events that did not occur, or not containing events that did, are removed. Figure 2.1 is a schematic representation of $\mathcal{H}_{\hat{t}}$ for a value of $\hat{t} > t_0$. The single line before $\hat{t}$ represents the fixed sequence of events that have already occurred, and the tree structure after $\hat{t}$ represents the possible sequences of events that may occur in the future.

We model the performance of an action by further restricting the current set of active histories. Saying that the system performs an action of a specified type is equivalent to adding an event of that type, with a time of occurrence equal to $\hat{t}$, to the prefix of events that must be a subset of every active history. This is the only control the system has over the incremental winnowing of $\mathcal{H}_{\hat{t}}$ as $\hat{t}$ increases (i.e., over the history that is eventually realized). This definition does not restrict the number of actions that the system can take simultaneously. For a particular time-dependent problem, however, there probably will be restrictions. For example, a robot with separate actuators for two arms and a movable camera could conceivably take three actions at once, though the choice of actions that can be taken at the same time is restricted. The robot could not, for instance, initiate two actions to move the camera to different locations at the same time.

The membership of the set of possible histories $\mathcal{H}$ may also be restricted by some form of causal relationship or "domain physics." For example, a causal rule requiring that an event of type τ_1 always be followed by an event of type τ_2 after a delay of n time units can be thought of as restricting $\mathcal{H}$ to include only those histories in which this relationship holds. These relationships may be probabilistic. We can generalize

the causal rule above to say that the occurrence of an event of type τ_1 influences the probability of occurrence of an event of type τ_2 at a particular time. In this case the evolution of the current active histories $\mathcal{H}_{\hat{t}}$ as $\hat{t}$ increases can be modeled as a random process. We make no restrictions on what sorts of relationships may govern the evolution of $\mathcal{H}_{\hat{t}}$.

We are primarily interested in sequences of events (histories), but it turns out to be convenient to introduce states as a shorthand for sets of histories. We make no commitment on how these states are represented. States can include calculated quantities (the level of water in the bathtub, given when the faucet was turned on and how far) as in [Hendrix, 1973, Dean and Siegle, 1990], or propositions whose presence or absence is determined by something like STRIPS rules [Fikes and Nilsson, 1971]. Events with uncertain outcomes can be modeled by leaving the type of the event uncertain until the event occurs, after which the type is determined (not necessarily known by the system) and in turn determines the resulting state(s). The initial state affects both what elements of $\mathcal{H}$ may eventually be realized and the utility resulting from the realization of a particular history h.[5]

Specifying a time-dependent problem defines a class of possible interactions between system and environment (i.e., a set of histories). The nature of the environment, the actions the system might take, and the causal relationships among actions and events are all fixed. This defines a space from which instances can be drawn and examined. We specify an *instance* of a time-dependent problem by fixing the initial state and possibly making some additional restrictions on or modifying a distribution over the current active histories $\mathcal{H}_{\hat{t}}$ for $\hat{t} = t_0$. For example, we might specify the occurrence of a set of requests at various times, or arrange for an event to occur at a particular time, where the type of the event depends on the system's actions occurring before that time. What we want to avoid is specifying parts of the realized history h^* that depend on actions the system takes, and thus on the deliberation it performs.

2.3 Expectation-Driven Iterative Refinement

In a time-dependent problem specified as in the last section, the system's only influence on the succession of partial histories (and thus on the value of the utility function $U(h)$) is deciding what actions occur and when. A *solution* to a time-dependent problem is a mechanism for making these decisions. The aim of this thesis is to make the generation and analysis of solutions to time-dependent problems easier. Our approach to the analysis and solution of time-dependent problems has been to define a framework

[5] Think of the initial state as a shorthand for the current active histories at t_0.

restricting the form of the solutions we propose called *expectation-driven iterative refinement*. A solution to a time-dependent problem using expectation-driven iterative refinement consists of:

- A set of *anytime decision procedures* $\mathcal{P}$ to which deliberation time may be allocated. These procedures provide an answer for any allocation of computation time, and the answer provided is expected to improve with additional allocations of time.

- An *agenda* of intended actions. The agenda is an ordered list of events corresponding to actions the system has committed to taking at the indicated time. Actions are added to the agenda only by deliberation. As time progresses ($\hat{t}$ increases), actions on the agenda are taken at the appropriate time.

- A deliberation-scheduling policy. This policy determines allocations to the elements of $\mathcal{P}$. The deliberation-scheduling policy may also determine when to execute the next action on the agenda (i.e., early, on time, or late). The deliberation-scheduling policy is implemented by a *deliberation scheduler*.

The anytime decision procedures used for deliberation can be schedulers, path planners, or sensor analysis routines, or can perform some other computation affecting the system's behavior. The decision procedures used for deliberation have access to the system's internal state. The sensor data can be read but not modified, and the deliberation scratchpad can be modified arbitrarily. Unless otherwise specified, we will assume that there is no limit on scratchpad space and that the various decision procedures use disjoint pieces of it.[6]

We introduce the agenda as a means of synchronizing the run-time execution of actions with the occasional and uncertain scheduling of deliberation time to determine what action to take when. It serves a further purpose in providing a boundary between the capabilities of the deliberation procedures and those of the deliberation scheduler. Deliberation procedures can add actions to the agenda, while the deliberation scheduler can only advance or delay the time of occurrence of the next action on the agenda.

Some such boundary is necessary in order to constrain the nature of the optimization problem the deliberation-scheduling policy addresses. If the deliberation scheduler can make any change to the agenda that some deliberation procedure can, then it would have to do the same expensive calculations, since we are interested in providing optimal deliberation-scheduling policies. On the other hand, it is clear that the deliberation

[6]This will not always be the case. For the Robot Courier, for example (Chapter 4), the results of one procedure are used by another.

scheduler should have some control over the execution of actions. For instance, it may be advantageous to delay acting in order to do additional deliberation.[7]

In order to include deliberation in the passage of time, we define a set of event types $\mathcal{D}$ to be added to $\mathcal{T}$, each corresponding to the starting point of a period over which deliberation is performed on (the processor is allocated to) a specified element of $\mathcal{P}$. These events are "actions" in the sense that they are under the system's control. We do not include them in $\mathcal{A}$ because:

- They are part of the solution framework, not the problem specification.
- They have no effect on $\mathcal{H}_{\hat{t}}$, in that performing an action in $\mathcal{D}$ does not constrain what events with types in $\mathcal{T} - \mathcal{D}$ may occur.[8] It may alter the system's expectations on $\mathcal{H}_{\hat{t}}$, in that scheduling deliberation time affects what actions in $\mathcal{A}$ the system is likely to take in the future.
- Events with types in $\mathcal{D}$ are not considered in calculating the utility of a history.

Events with types in $\mathcal{D}$ may be performed in exactly the same way as other actions. An event with a type in $\mathcal{D}$ starts an interval of deliberation allocated to some procedure p. Deliberation on p continues until the occurrence of another event with a type in $\mathcal{D}$.

Actions that the system takes, including scheduled deliberation, "take time" in the sense that they are implemented as events occurring as $\hat{t}$ advances. The computation performed in doing deliberation scheduling is not scheduled, and thus is not included in the passage of time as measured by the changing value of $\hat{t}$. Neglecting the computational cost of deliberation scheduling can be justified only if that cost is small. For the problems discussed in Chapter 3 (time-dependent planning) and Chapter 4 (the Robot Courier) we can provide near-optimal deliberation scheduling at a very small computational cost.

There are several approaches we might take in situations where the cost of deliberation scheduling is not small enough to be ignored:

- Provide suboptimal allocations, along with some notion of how far from optimal the resulting allocations are. This is the approach taken in Chapters 3 and 4.
- Schedule deliberation by running a procedure that constructs an explicit schedule. Run this procedure relatively infrequently and amortize the computational cost.

[7] Exactly how we should restrict the deliberation scheduler's manipulation of the agenda is an open question. The choice we make here is a mimimal one: we add to the deliberation scheduler's capabilities only as needed to do effective deliberation scheduling. Another thing to keep in mind is that we are assuming that the deliberation scheduler is computationally very cheap, and so that it cannot be doing any significant computation.

[8] Events with types in $\mathcal{D}$ cannot participate in causal relationships.

- Schedule deliberation scheduling explicitly. This requires adding another scheduler that abstracts even further from the original problem (so as to run faster) than the deliberation scheduler.

Applying expectation-driven iterative refinement reduces a large problem (providing a solution to a time-dependent problem) to a pair of smaller ones:

- Defining the anytime decision procedures in $\mathcal{P}$.
- Defining a deliberation-scheduling policy.

In Section 2.4, we discuss the problem of finding or generating anytime decision procedures and gathering useful expectations on their behavior. In Section 2.5, we explore the difficulties involved in constructing a decision model and a deliberation-scheduling policy that maximizes expected utility with respect to that model.

2.4 Anytime Decision Procedures

Anytime decision procedures are implementations of *anytime algorithms*: iterative algorithms that are guaranteed to produce an answer at any point, where that answer is expected to improve with each iteration. Several issues must be settled in the use of anytime algorithms:

- What kinds of anytime algorithms are there?
- What does it mean for an answer to "improve" as more time is allocated?
- How do we construct new anytime algorithms for complex problems?

There is also a question that is more properly addressed in terms of the implemented version of an anytime algorithm (i.e., an anytime decision procedure):

- What kinds of expectations on the behavior of anytime decision procedures will be useful, and what kinds of expectations can we gather?

There are two reasons for using anytime decision procedures. The first is that there may be a wide range of resource availability. We want to be able both to produce a useful answer even for very small allocations of time and to take advantage of any extra time available to produce a better answer. This consideration applies even when we can predict the future with a high degree of accuracy. Lesser [1988] has investigated the use of *design-to-time* procedures. Design-to-time procedures are decision procedures that are constructed at run time to fit the time available, that must run to completion to produce an answer. Design-to-time procedures provide a great deal of flexibility in

how much time is to be allocated to a particular problem, as long as the allocation can be made at the time the decision procedure is constructed. If prediction is not reliable, then using design-to-time procedures for deliberation may not work out so well. The advantage to using anytime decision procedures in this case is always having a partial answer. This means that the system can reschedule to accommodate unexpected events, can supply an answer if some event requiring a response occurs unexpectedly early, and can improve an answer further if the event is late.

2.4.1 Examples of Anytime Algorithms

Almost any algorithm can be implemented in an anytime procedure by embedding it in a second procedure that runs the original algorithm in an inferior process. When the parent process is interrupted and asked for an answer, it checks to see if the inferior process has terminated; if so it returns the answer generated by the inferior process, and otherwise returns some default answer. In most cases, however, we can provide a more useful anytime procedure (i.e., one that produces a succession of increasingly useful results). For instance, many search algorithms employ a metric for comparing solutions. Such an algorithm could easily be designed to return its current best answer at any point in the computation. Such "well-behaved" anytime algorithms are ubiquitous in computer science. We discuss some examples below.

Numerical Approximation

The study of methods for iterative approximation is a large and active area in numerical analysis [Tompkins and Wilson, 1969, Hageman and Young, 1981]. Two common techniques are Taylor series approximations (e.g., computing π or e) and iterative finite-element methods. Finite-element methods can be used for approximating continous functions [Varga, 1962], including the solutions to differential equations [Manohar, 1988]. It is usually possible to characterize the convergence properties of these methods, including bounds on the error in the answer resulting from a given number of iterations [Varga, 1962].

Finite-element methods are not confined to traditionally "mathematical" problems. Hager [Hager, 1988] applies finite-element methods to calculate the most probable interpretation for uncertain sensor data, as well as to determine what new readings can best reduce the uncertainty in that interpretation.

Probablistic Algorithms

An algorithm can be "probabilistic" in a number of senses. Those most directly useful as anytime algorithms are the *Monte Carlo* algorithms [Harel, 1987], in which some

parameter is estimated with increasing confidence as the result of drawing repeated samples at random from the search space. The iterative nature of the resulting algorithms is not their only advantage. It is sometimes possible to derive an answer having very high confidence and/or precision from a number of samples that is much smaller (perhaps exponentially smaller) than the space we would have to search to provide an exact answer.

There are two main types of Monte Carlo methods. In the first, the algorithm provides an answer that is increasingly likely to be correct as further samples are drawn. An example of this type is primality-testing using witnesses [Harel, 1987]. This algorithm exploits the fact that with probability approximately $\frac{1}{2}$, any of the numbers smaller than the number being tested is a *witness* to its being composite: finding a witness establishes that the number is not prime. That a number chosen at random is not a witness increases the probability that the number being tested is prime. The time necessary to run the primality-testing algorithm depends on the probability bound required; the more points tested, the smaller the probability that we falsely identify a number as a prime. An anytime procedure for primality testing using this approach would continue choosing numbers at random and testing them as witnesses until it was interrupted (or until it determined that the number was composite), and then return the probability that the number was in fact a prime. The probability of incorrectly classifying a candidate number as prime is approximately halved for each iteration.

There are also "numeric" Monte Carlo methods in which the answer returned is a number or a function. In this case, the answer improves with succeeding iterations in the sense that the probability that the difference between the answer returned and the real answer is outside some fixed bound decreases. For example, one way of estimating the area under a curve that is too complex to integrate is to define a rectangle that includes the entire curve between the integration limits. By testing whether points drawn at random from this rectangle are under or over the curve (inside or outside), we estimate the fraction of the area inside the rectangle under the curve. We can use Chebyshev's inequality to bound the probability that the error in this estimate is greater than δ:

$$P\left((\frac{S_n}{n} - A) \geq \delta\right) \leq \frac{1}{4n\delta^2}$$

where A is the fraction of the rectangle under the curve (the real area), n is the number of points sampled, and S_n is the number of points falling under the curve.

Heuristic Search

Algorithms for heuristic search, in particular those employing variable lookahead and fast evaluation functions, can be cast as anytime algorithms [Pearl, 1985, Korf, 1990].

Each iteration expands another node or nodes in the search tree, possibly causing a change in the ranking of possible next moves or in the best known solution. Using decision-analytic methods to determine how (and whether) to expand search trees is currently an active research area [Russell and Wefald, 1989b, Hansson and Mayer, 1988].

Probabilistic Inference

In the most common form of probabilistic inference, we start with a set of random variables, information on how the values of these variables depend on one another, and a set of observations fixing values for some of the variables. We then calculate the posterior distribution for the value of some random variable or variables. Despite recent advances in the construction and evaluation of belief nets as a method for probabilistic inference [Pearl, 1988], this problem is NP-hard [Cooper, 1987]. A wide variety of methods has been developed for approximate evaluation of belief nets (i.e., providing bounds on the posterior distribution, rather than the exact distribution). Several of these methods are anytime algorithms in the sense that the bounds get smaller for additional iterations of the basic method [Horvitz *et al.*, 1989, Heckerman, 1985, Henrion, 1986].

Dynamic Programming

Dynamic programming involves breaking down a problem down in such a way that the best answer can be calculated and stored for larger and larger pieces of the original problem, until finally the best answer for the entire problem is found. Consider the problem of multiplying the following sequence of matrices:

$$M_1 = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{pmatrix}, \quad M_2 = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \end{pmatrix}$$

$$M_3 = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} \end{pmatrix}$$

Matrices are associative, so we can multiply M_1 and M_2, then multiply the result by M_3. Or we can multiply M_2 and M_3 first. How many multiplications are needed in each case? Multiplying M_1 by M_2 requires $4*2*4 = 32$ multiplications. Multiplying the resulting 4×4 matrix times M_3 requires an additional $4*4*4 = 64$ multiplications, for a total

cost of 96. Multiplying M_2 and M_3 first results in a total cost of $(2*4*4)+(4*2*4) = 64$. For larger sequences, the savings can be considerable. The naive solution to the problem (separately checking each way of associating the matrices) entails examining an exponential number of alternatives. Using dynamic programming to solve this problem for larger sequences, we would start by caching all the pairwise multiplication costs, then proceed to cache the best (cheapest) way to multiply any three matrices together, then any four, and so on, each time using the results cached at preceding levels.

It seems reasonable that caching more results would mean a lower expected cost for a solution picked at random. This intuition is borne out experimentally. We repeatedly generated sequences of 10 matrices with dimensions randomly chosen from the interval $[1, 100]$. We defined a search procedure for picking associations for multiplying a sequence together. This procedure works recursively by breaking the current sequence into two pieces at a random point, finding the cost of multiplying the resulting subsequences, and adding the cost of combining their product matrices. If a cached answer is found for a particular subsequence, that answer is used, otherwise the procedure bottoms out at pairwise multiplications.

For each sequence, a dynamic programming solution was generated, one step at a time, where each step consisted of calculating and storing the optimal way to multiply some subsequence of size k. After each step, the average cost of the solution that would be generated by the search procedure was calculated. Averaged over many trials, the improvement in expected cost increases as shown in Figure 2.2. The periodic dips are steps at which k changes. Apparently, having the first cached answer for a subsequence of size k does not help as much as adding answers for additional k-length subsequences. The big jump at the end is the result of going from randomly choosing among nine possibilities, one of which is optimal, to knowing the optimal answer.

Symbolic or Discrete Algorithms

Another approach to combinatoric problems is to use greedy approximation algorithms that have a reduced solution space (they are "approximate" also in that the optimal answer may not be in the reduced solution space). An example of this type of algorithm is the 2-opt algorithm used to generate approximate solutions for instances of the traveling salesman problem (TSP). 2-opt begins with a cheaply generated tour that includes each city specified in the TSP instance. It then chooses two arcs in the tour, removes them, and reconnects the disconnected cities to form a new tour of smaller cost. In the standard approach, this cycle is repeated until there is no pair that can be exchanged to improve the tour. It has been shown empirically that running 2-opt to completion produces tours that average within about 8% of the cost of the optimal tour. There are more complicated edge-exchange algorithms that do much better [Lin

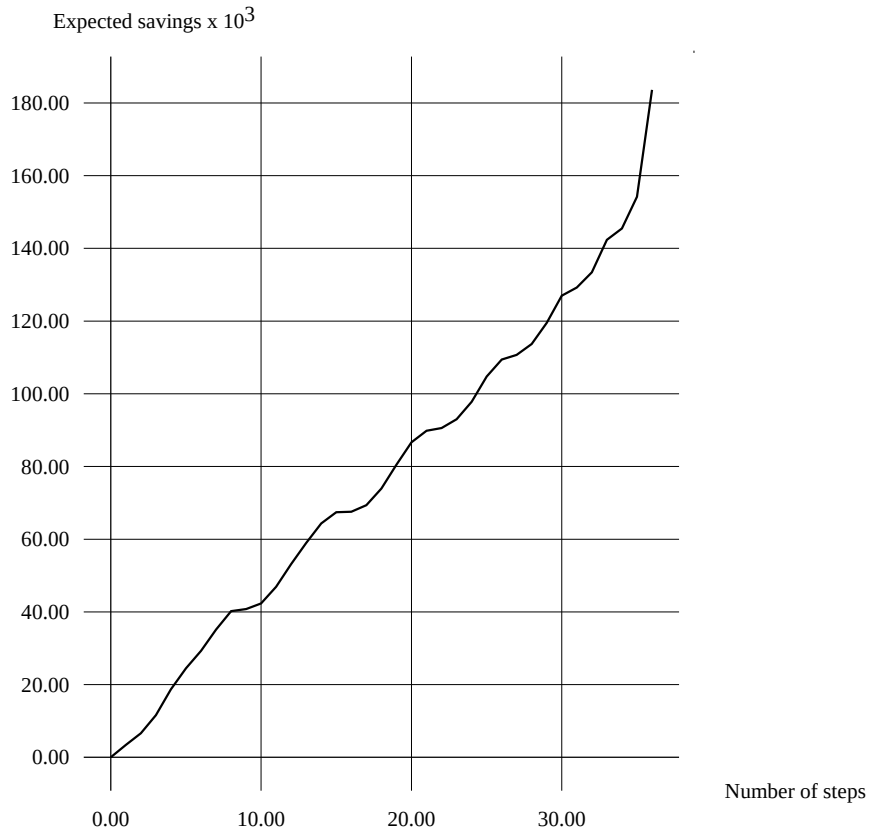

Figure 2.2: Expected savings over answers cached

and Kernighan, 1973]. An anytime procedure using 2-opt exchanges pairs of arcs until it is interrupted and asked for an answer, or until no exchange will yield a tour of smaller cost, at which point it returns the current tour.

Symbolic processing in general can be viewed as the manipulation of finite sets (of bindings, constraints, entities, etc.). The study of discrete iterations [Robert, 1986] investigates the behavior of iterated functions over finite sets. Algorithms that operate over sets "converge" in terms of the membership of the set in question. Elements are successively added to or removed from a set representing an approximate answer so as to reduce the difference between that set and a set representing the correct answer The analysis of discrete iterations is closely tied to work on connectionist models [Hinton and Sejnowski, 1983], cellular automata [Farmer *et al.*, 1984], and VLSI system design [Mead and Conway, 1980].

2.4.2 Performance Profiles

In order to allocate deliberation time effectively, we use expectations on the behavior of the anytime decision procedures available. Ultimately, what we want to know is the expected effect of a particular allocation on the utility of the system's behavior. More commonly, what we have are expectations on how some parameter of the answer returned by an anytime decision procedure changes with increasing allocation. Expectations on some parameter of the answer returned are simpler to obtain, and not so dependent on the structure of a particular problem, as expectations on how the system's behavior will change. We can compute expectations on the answers returned by an anytime decision procedure given a particular input distribution, and then use that anytime decision procedure and those expectations for any time-dependent problem for which the assumptions made about the input distribution are reasonable. We call such a set of expectations a *performance profile*.

In this section, we address three issues related to the generation and application of performance profiles:

- What does it mean for an answer to "improve?"
- What sort of information might be useful in a performance profile?
- What sort of information is reasonably easy to obtain?

Convergence

Part of the definition of an anytime decision procedure is that the answer it returns is expected to improve with increasing allocations of deliberation time. Answers might "improve" in the following ways, among others:

- Convergence to an optimal value (e.g., finding a minimum-length path).
- Narrowing the bounds on an estimate. Here the bounds might be nearly exact; for some algorithms the bounds on the correct answer established by each iteration are easy to calculate, and each iteration takes a more or less predictable amount of time.
- Increasing the probability that the answer is correct (discrete case), or within some ϵ of correct (continuous case).
- Increasing the intersection of or decreasing the difference between the current answer and the correct answer (the "answers" in this case are sets).

Several parameters of any given anytime decision procedure may be of interest. In addition, the answers returned by an anytime decision procedure depend on the data it takes as input. Thus a single anytime decision procedure might have several performance profiles, each providing information on the distribution of a different parameter or for a different distribution over the possible values of its inputs.

Which of these profiles is most appropriate or useful depends on the task at hand. Horvitz [Horvitz, 1988] discusses various algorithms for sorting an array viewed as anytime algorithms (he calls them *flexible computations*). Each algorithm makes repeated changes to the array, each one bringing the array closer to being sorted. What is meant by "closer" differs for each algorithm, however. Suppose that we start with an unsorted array, and look at the changes made to that array as the sort progresses. Insertion sort produces a lengthening sorted section at the beginning or end of the array. Bubble sort produces an increasing number of elements in the correct relative position. Shell sort produces a gradually decreasing maximum distance for entries from their true position. Clearly, which of these algorithms is most appropriate for a particular application depends in part on what kind of "approximate sort" is be most useful; the analogy with our case is evident.

Expectations

In previous work [Dean and Boddy, 1988, Boddy and Dean, 1989], we have compiled performance profiles that tracked the expected value of (some parameter of) the answer returned by a given anytime decision procedure as a function of deliberation time. Some examples of performance profiles of this kind are given in Figure 2.3. These continuous profiles are approximations. The corresponding procedures are iterative, and thus provide answers that improve in a series of discrete steps. Continuity is an assumption we make, and is required for applying many of the techniques we employ (e.g., the deliberation schedulers in Chapter 3 would not be optimal otherwise).

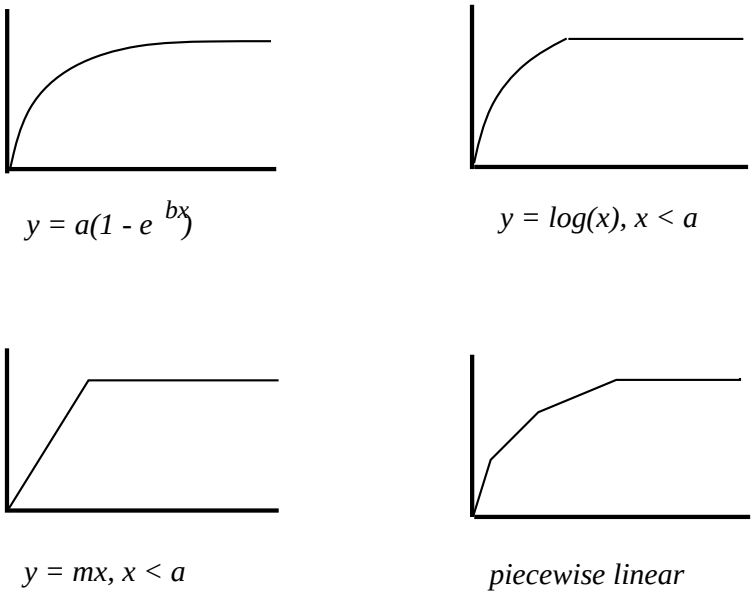

Figure 2.3: Some performance profiles

All of the profiles in Figure 2.3 exhibit *diminishing returns*: the improvement in the answer for an additional allocation does not increase (and will eventually decrease) with the time already spent. This is a property of many interative algorithms, including most of the examples discussed in Section 2.4.1. It turns out to be a useful property, in that it can simplify the optimization problem involved in deliberation scheduling (see Chapter 3, for example). There are certainly iterative methods that do not exhibit diminishing returns. Consider the problem of finding a 10-digit combination for a safe, using a procedure that at each step produces the next digit [0-9] in the combination. The probabilities of guessing the correct combination, given the digits already know, at each iteration form the sequence $\langle 10^{-10}, 10^{-9}, \ldots, .1, 1\rangle$. If we take the expected utility to be the utility u of opening the safe times the probability of opening the safe, the gain in expected utility for the last step ($0.9u$) is much greater than for the first step ($0.000000009u$).

Knowing the expected value is in general insufficient—there are situations where you need to compute a marginal distribution, not just a sum or product of expected values. For instance, two path planners that produce paths with the same expected length for a given allocation of deliberation time can have very different expected utilities if arrival time has a hard deadline: the planner with greater variance on expected length may have a lower expected utility, because it has a greater probability of missing the deadline. Figure 2.4 shows an example of this. Each graph shows a distribution of travel times and a deadline, and in each case, the mean μ and deadline d are the same.

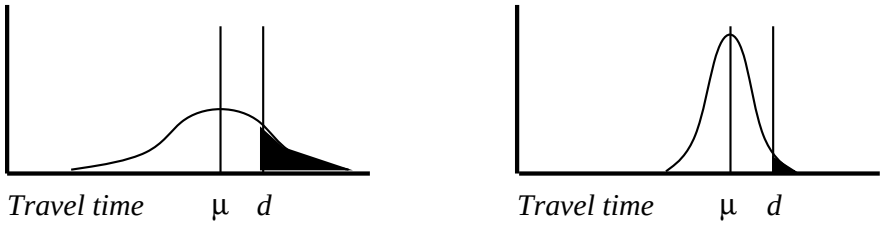

Figure 2.4: Expected travel time and a deadline

The area of the shaded region is the probability of missing the deadline.

A completely general performance profile is a surface in 3-space, not a 2D graph. Taking a slice across this surface for a given allocation of deliberation time provides a distribution on some parameter of the answer returned. Performance profiles like those in Figure 2.3 graph the means of the distributions represented in the more general form. The problem is that this more general form of profile requires more storage space. It also takes longer to gather enough data to compute reasonable distributions for each different allocation, or even for just a few of them. In addition, calculations using the means are usually much simpler and faster than computing a distribution would be.

In providing solutions for the time-dependent problems presented in this thesis, we recognize that using just the expected values may result in suboptimal deliberation scheduling, and we do it anyway. The savings in both space (storing the profiles) and time (gathering the necessary data and doing the calculations) are simply too great. We can define the cost of using the simpler performance profiles in terms of the expected utility of the system's performance. Let $E(U_x)$ be the expected utility of using an optimal deliberation scheduler for a decision model that includes the simpler kind of performance profiles, and $E(U_{IU})$ be the expected utility of using an optimal deliberation scheduler for a second decision model that differs only in using the 3D performance profiles. Then the cost of using the simpler profiles is just $E(U_x) - E(U_{IU})$.[9] For time-dependent planning problems (Chapter 3), this cost is zero, because overall utility is the sum of the utility of independent responses.

As long as we ignore the cost of deliberation scheduling, the cost of using the simpler profiles is at least zero. In situations where the space or time required for deliberation scheduling is significant and must itself be scheduled, the "cost" of calculating using only the expected values may well be negative, because the calculations are so much simpler.

[9] Henrion [1984] calls this the *expected value of including uncertainty.* (EVIU)

Execution

We view deliberation time as being allocated in discrete increments, not necessarily of a fixed size. The deliberation scheduler must decide on the basis of current information how to allocate the current increment. As time passes, new information becomes available that may influence the deliberation scheduler's decision. Some of this information concerns the occurrence (or non-occurrence) of events or the effects of the system's actions. Some of it may have to do with the behavior of the decision procedures to which the deliberation scheduler allocates time. The information available on the current state of the deliberation procedures can vary widely. These are some of the kinds of information that would be useful:

- When the algorithm has terminated (i.e., when improvement will result from further computation).
- How much more time can be allocated before the algorithm terminates.
- How good a partial answer actually is (e.g., for Monte Carlo algorithms, the expected error for a given number of iterations is known).

Which of these pieces of information are available depends on the system's deliberation procedures. Let's look at some specific examples of anytime decision procedures, and what can be determined about the current state of their computations:

- Robot Courier path planning by heuristic search (Chapter 4):
  - We know when it terminates (a complete path has been planned).
  - We know how much is left to do (the distance remaining to the goal).
  - The interim answer is rated in terms of expected time to traverse the distance (we have a good estimate for the expected time given a complete path).
- Robot Courier tour improvement by edge exchange (Chapter 4).
  - We know when it terminates (no more exchanges are possible).
  - We do not know how much is left to do (there is no way to determine the number of further exchanges possible).
  - The interim result is rated by the length of the tour, compared to a fairly noisy estimate for the length of the best tour possible.
- Identification by features (identify correctly with increasing probability):
  - We might know when it terminates (do we know if there are relevant but unexamined features?).

- We might know how *many* more features there are to look at.
- The interim answer has a lower probability of being correct. We may be able to calculate this probability from the nature of the features already examined.

- Identification by black box:
  - We might know when it terminates (if the box tells us).
  - We do not know how much is left to do.
  - The interim answer has a lower probability of being correct, but we only have expectations on previous behavior to tell us what that probability is.

The significance of this information is that it influences the construction of a deliberation-scheduling policy. Any execution information becoming available over time provides new information that can be included in the decision model used for allocating deliberation time. For example, knowing that a procedure has terminated unexpectedly early (i.e., when the performance profile still indicates some expected benefit to allocating additional time) allows the deliberation scheduler to reallocate time that would otherwise be wasted.

2.4.3 Constructing Anytime Decision Procedures

The anytime algorithms in Section 2.4.1 perform fairly simple computations. For a given time-dependent problem, deliberation may involve some computation that requires a combination of known anytime decision procedures. For example, the system may need to combine the results of several database queries or to run several different kinds of analysis on a data set in order to come up with an interpretation. A library of anytime decision procedures and performance profiles that can be combined to construct new procedures should simplify the process of generating solutions for time-dependent problems.

We can combine anytime decision procedures in various ways. The results of the *component* anytime decision procedures must be combined to produce the output of the *composite* anytime decision procedure. The final result may be the direct output of a component procedure or some simple combination (e.g., addition) of the results of several component procedures. One component anytime decision procedure may take as input the output of several others. It is also possible for several anytime decision procedures to use the result of a single procedure (e.g., a sensor interpretation routine).

We start by considering two simple cases:

- The results of k anytime decision procedures are combined in some inexpensive way (parallel combination).
- One anytime decision procedure takes as input the output of k other anytime decision procedures (series combination).

We first show how each of these cases is dealt with, and then discuss generalizing to more complex combinations of anytime decision procedures.

Parallel Combination

We start with k algorithms and the corresponding performance profiles, and some function that will be used to combine their outputs. For any given allocation of deliberation time, we have distributions (or at least expectations) on the outputs and a function for combining them that can be used to construct a distribution on the function's result. Constructing an anytime algorithm is accomplished by interleaving small increments of time allocated to each of the k component procedures.

Consider a situation where we want the best estimate possible in the time available of the membership of some relation, where information on that relation can be found in any of several different databases. Vrbsky *et al.* [Vrbsky *et al.*, 1990] have proposed a set of anytime algorithms for relational database queries in which the answers returned improve in the sense that the set returned is closer and closer to the actual set of tuples satisfying the query. We interleave allocations to queries on each database. When the answer is requested, we sum the current results. The expected error in the final answer (the number of elements incorrectly included or excluded) can be calculated by summing the expected error for each individual query, given the time allocated (again, we assume that the errors are independent). Thus, let $E_i(\delta)$ be the expected number of membership errors (inclusions or exclusions) in the set of tuples returned by procedure i for an allocation of time δ. Then the total expected error is

$$E = \sum_{i=1}^{k} E_i(\delta_i)$$

where δ_i is the time allocated to procedure i. If all of the queries have the same performance profile, and if it exhibits diminishing returns, then E is minimized by making all the δ_i equal.

Series Combination

Suppose we are to design a controller for some kind of projectile weapon. Suppose further that we have available anytime decision procedures for tracking and aiming.

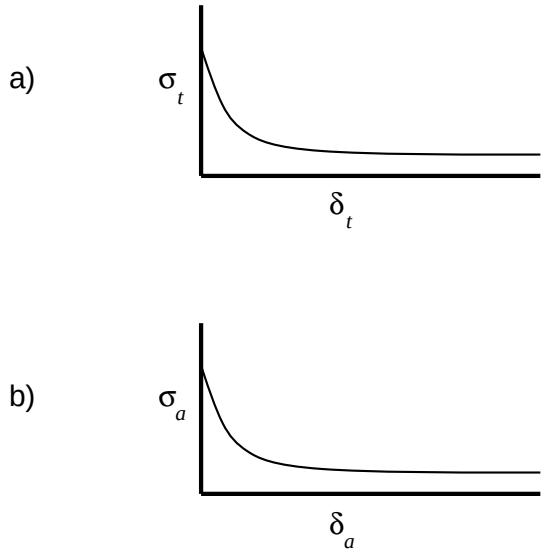

Figure 2.5: Variance as a function of deliberation for tracking and aiming errors

The tracking procedure produces an increasingly accurate estimate of the position of a stationary target as a function of time. The tracking error has a normal distribution $N(0, \sigma_t^2)$.[10] Figure 2.5-a plots σ_t^2 is plotted as a function of deliberation time. The target's estimated position is then fed to the aiming procedure, which decides how to manipulate the gun to hit a specified location. As the time allocated increases, the aiming procedure refines its answer, taking into account wind, Coriolis effects, ambient temperature and pressure, and perhaps other factors. The aiming error also has a normal distribution $N(0, \sigma_a^2)$, with σ_a^2 as a function of deliberation time as shown in Figure 2.5-b.

For series combination, we need to know how the output of the final procedure depends on its input. In this case, the effect of input errors on the output of the aiming procedure is simple to calculate. The position error—where the projectile hits compared to the target's location—is the sum of tracking and aiming errors. The distribution of position error for a particular allocation of deliberation time can be obtained by convolving the distributions for tracking error and aiming error (assuming that the errors are independent). Since the two error distributions are gaussian, the distribution for the position error is the normal distribution $N(0, \sigma_t^2 + \sigma_a^2)$. Let $\sigma_p^2 = \sigma_t^2 + \sigma_a^2$ be the variance for the position error. Given an amount of time δ available for deliberation, the value of σ_p^2 depends on the value of the allocations for tracking (δ_t) and aiming ($\delta_a = \delta - \delta_t$). Figure 2.6 shows the change in σ_p^2 as δ_t is varied from 0 to δ.

[10] We can eliminate systematic errors.

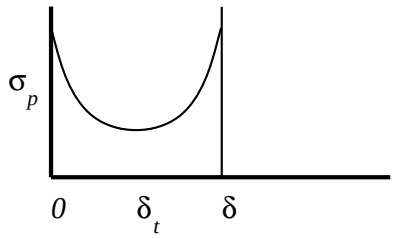

Figure 2.6: Variance in position error for different allocations to δ_t and δ_a

Figuring out the optimal allocations for δ_a and δ_t for a given value of δ is now straightforward: find the allocations that minimize the variance of the position error. Finding the optimal allocations and executing the procedures consecutively yields a composite procedure that provides optimal answers across a wide range of available times. The procedure is not, however, a particularly good anytime decision procedure: rather, it is similar to what Lesser calls a *design-to-time* algorithm [Lesser and Corkill, 1983]. If the composite procedure is interrupted unexpectedly, it may not have run the aiming procedure at all. If δ is the time the procedure was actually allowed to run, the time actually spent on the tracking procedure may be the optimal allocation for some larger value of δ, and the time spent on aiming may be less (perhaps much less) than is optimal for the time actually available. The variance on position error is thus higher than it would have been for optimal allocations (i.e., we are close to the right end of Figure 2.6, not near the middle).

To produce a better anytime decision procedure, we interleave execution of the two procedures, caching intermediate results. Let $\langle i, j \rangle$ represent the allocation of i time units to the first procedure and j time units allocated to the second procedure. The following sequence makes all the possible allocations from $\langle 0, 0 \rangle$ to $\langle m, n \rangle$ (for $m \leq n$):

$$
\begin{array}{lll}
\langle 0,0 \rangle, & \ldots, & \langle m,0 \rangle \\
\langle 1,1 \rangle, & \ldots, & \langle m,1 \rangle \\
\langle 2,2 \rangle, & \ldots, & \langle m,2 \rangle \\
. & & \\
. & & \\
\langle m,m \rangle & & \\
\langle m,m+1 \rangle & & \\
. & & \\
. & & \\
\langle m,n \rangle & &
\end{array}
$$

One problem with this sequence is that some of the steps may result in an answer that is expected to be *worse* than the preceding answer. The sequence can be sorted to

alleviate this difficulty, as long as no element $\langle i, j \rangle$ ends up before any element $\langle k, l \rangle$ such that $k \leq i$ and $l < j$. This sequence also takes $O(m^2 - m + n)$ steps, instead of the $m + n$ required if the two procedures are executed consecutively.

Leaving out some of the intermediate steps yields a procedure that is less expensive to run. The trick is to do this in such a way that the steps are all of the same size. For the current example (the controller for a gun), the best answers are for allocations that are nearly equal. The sequence

$$\langle 0,0 \rangle, \langle 1,0 \rangle, \langle 1,1 \rangle, \langle 2,1 \rangle, \langle 2,2 \rangle, \langle 3,2 \rangle, \langle 3,3 \rangle, \ldots, \langle m, m-1 \rangle, \langle m, m \rangle,$$

provides the optimal allocations for consecutive values of δ starting at zero, and takes only $2m = \delta$ steps, the same as required to execute the procedures consecutively. The problem is that, even using the answers cached in previous steps, more than one unit of time is needed for some of the steps. For example, getting from $\langle 2,2 \rangle$ to $\langle 3,2 \rangle$ requires spending 1 unit on tracking (from 2 to 3), and then 2 units on aiming, because we haven't cached any partial answers for aiming starting with the answer returned for $\delta_t = 3$. The next-to-last step in this sequence takes m time units.

On the other hand, all the steps in the following sequence take 1 time unit and the total number of steps is $5m/2$:

$$\begin{array}{lll}
\langle 0,0 \rangle & & \\
\langle 1,0 \rangle & & \\
. & & \\
. & & \\
\langle m/2, 0 \rangle, & \ldots, & \langle m/2, m/2 \rangle \\
\langle (m/2)+1, 0 \rangle & & \\
. & & \\
. & & \\
\langle m, 0 \rangle, & \ldots, & \langle m, m \rangle
\end{array}$$

We can rearrange this sequence to interleave the steps $\langle m/2, 0 \rangle, \ldots \langle m/2, m/2 \rangle$ with the following steps to produce a sequence such that the answer improves every k steps, $k \leq 3$, and the step costs are still all 1 time unit. At the cost of increasing the length of the sequence, we can increase the number of intermediate allocations for the first procedure (only $m/2$ in the sequence above). This may mean that the composite procedure provides better answers for small allocations of deliberation time (small values of δ).

The added expense of interleaving the component procedures in this way results in a interesting tradeoff: if we can make reasonably good estimates of how much deliberation time is available, then the design-to-time procedure (running the two procedures

consecutively) may have a higher expected utility. One can also pursue approaches intermediate between pure anytime and design-to-time procedures by varying the size of the increments allocated solely to one of the component procedures (tracking or aiming, in this example). This tradeoff also suggests that there may some incentive to construct composite procedures for deliberation at run time, in that the best sequence of steps may depend on the time available.

A somewhat different example of series combination from the one analyzed here has been proposed by Breese. In [Breese, 1990], he incrementally constructs influence diagrams [Shachter, 1986] corresponding to partial decision models, and then solves them approximately.[11] The time allocated to both model construction and finding an approximate solution is determined using expectations on the effect of a given allocation on the expected utility of the final decision.

Generalizing

Moving beyond simple combinations complicates the situation. We can think of the information dependencies among anytime decision procedures as a directed graph in which the nodes are anytime decision procedures or combinations of data in some other way (e.g., addition), and the arcs represent the flow of information (i.e., the results of one procedure being used by another). We call such a graph a *procedural-dependency graph*.[12] One obvious question to ask is whether procedural-dependency graphs must be acyclic. In principle, there is no reason why they have to be. Consider a pair of routines that alternately modify the same database, where each modifies the data in ways that affect the output of the other routine.[13] In practice, we will probably assume that procedural-dependency graphs are acyclic.

A procedural-dependency graph provides a partial ordering for the corresponding anytime decision procedures according to the flow of information from one to another. One problem with this formulation is that undirected cycles represent dependencies we may have to take into account. Consider the example in Figure 2.7. If a_1 with probability p produces an output that causes both a_2 and a_3 to produce very bad answers, this fact is accurately represented in the distributions calculated for the outputs of a_2 and a_3. Unfortunately, if in constructing a output distribution for a_4 we only consider a_2 and a_3, the probability of these very bad answers happening at the same time will be calculated as p^2 rather than the correct value of p. We can use simple parallel and serial

[11] This is a design-to-time procedure.

[12] Procedural-dependency graphs are similar to *data-flow graphs* [Treleaven *et al.*, 1982]

[13] If the modifications don't affect the other routine, the procedures are independent and there is no cycle.

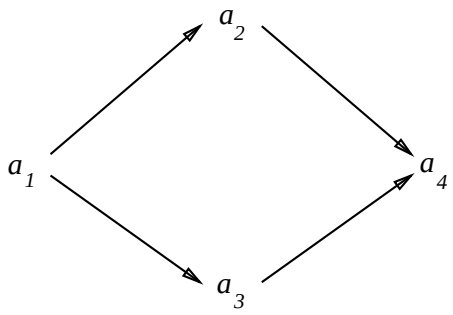

Figure 2.7: Undirected cycle in a procedural-dependency graph

combinations to construct performance profiles for composite procedures corresponding to arbitrary procedural-dependency graphs. But the dependencies represented by undirected cycles will not taken into account.

For a given set of deliberation allocations, a procedural-dependency graph can be viewed as a belief net [Pearl, 1988] in which the arcs represent probabilistic dependencies and the corresponding distributions are given by the performance profiles. Results obtained in calculating posterior distributions from belief nets [Cooper, 1987] suggest that "solving" procedural-dependency graphs in a similar way that takes undirected cycles into account is likely to be difficult. This correspondence also suggests that some of the considerable existing work on belief nets might be of use in analyzing complex procedural-dependency graphs.

Procedural-dependency graphs give us a general architecture for representing and reasoning about composite anytime decision procedures. Some issues may prove troublesome, e.g., undirected cycles in the information dependency graph. Whether or not these difficulties are significant depends on how often they arise in practice, how hard they are to fix, and what kinds of errors are introduced by ignoring them. Suppose we only consider paths of length one (thus missing any undirected cycles). The loss in expected utility of the system's resulting performance is the only significant factor, and probably can be determined only on a case-by-case basis. It is also entirely possible that graphs containing undirected cycles will not be a problem because there is little or no call for composite deliberation procedures that are complex enough to need them. The end result of this combination should be a procedure for performing a single deliberative "task," one of a suite of procedures to which a deliberation scheduler might allocate time.

2.5 Deliberation Scheduling

The specification of a time-dependent problemincludes a utility function defined on the system's interaction with its environment (on histories, in the terms used in Section 2.2). This specification also restricts the form that interaction can take—in particular, the actions the system can take and their consequences in various situations.

Generating a solution to a time-dependent problem using expectation-driven iterative refinement requires that we find anytime decision procedures for deliberation and compile performance profiles encoding expectations on the behavior of those procedures, as described in Section 2.4. In this section, we are concerned with the next step: allocating time to those procedures. We use the utility function U, the performance profiles, and perhaps other expectations on the behavior of the system and the environment to construct a *deliberation-scheduling policy*: a function mapping from the current state of the system (which may include information on the current or projected state of the environment) to a decision. Each decision is a choice of whether or not to execute the next action on the agenda, and what to deliberate on in the next increment of time. A deliberation-scheduling policy is implemented by a procedure known as a *deliberation scheduler*.

In this section, we explore the problem of constructing a deliberation-scheduling policy, and how that problem changes, depending on the information we take into account. The expected utility of the system's behavior is calculated according to a *decision model* that includes information on the behavior of the system or the environment. The performance profiles are always part of this decision model. Expectations on the occurrence of events not under the system's control may or may not be contained in the decision model. The decision model need not be restricted to the information available to the deliberation-scheduling policy. In particular, the deliberation-scheduling policy is limited to information available in the system's internal state (i.e., known to the system), but there is no such limitation on the information included in the decision model by which the system's performance is judged.

2.5.1 Sequential Decision Problems

Constructing a deliberation-scheduling policy can be viewed as a *sequential decision problem* [Ross, 1983]. Deliberation time is allocated in discrete increments, not necessarily of a fixed size. At the start of each increment, the deliberation scheduler provides a decision. Time passes, events happen, and the internal state of the system changes due to the deliberation performed. Then the deliberation scheduler makes another decision, and another, and so on.

There are standard methods in decision analysis for analyzing sequential decision

problems [Ross, 1983, Bellman, 1957, Raiffa, 1968]. Let $\mathcal{S}$ be a set of states, $\mathcal{D}$ the set of possible decisions, $R : \mathcal{S} \times \mathcal{D} \rightarrow \Re$ a *reward* function, and $\phi : \mathcal{S} \times \mathcal{D} \rightarrow \mathcal{S}$ a function mapping from the current state and a decision to the next state. The reward resulting in a single step from making decision d in state s_i is $R(s_i, d)$. The next state is $\phi(s_i, d)$. We call the sum of the rewards from a sequence of decisions the *value* of the sequence. The maximum possible value for one step starting in state s_i is

$$V_1(s_i) = \max_{d \in \mathcal{D}} R(s_i, d)$$

The value resulting from d depends on the decisions made in all the following states. Choosing the decision d that maximizes the value of the sequence of n states starting in s_i involves finding

$$V_n(s_i) = \max_{d \in \mathcal{D}} [R(s_i, d) + V_{n-1}(\phi(s_i, d))]$$

This can be solved, at least in principle. Dynamic programming [Bellman, 1957] is frequently employed for the solution of this type of problem. A very long or infinite sequence of decisions can be handled using *discounting*, in which the reward resulting from being in a given state is weighted inversely to how far in the future the state is. The resulting optimization problem is

$$V_n(s_i) = \max_{d \in \mathcal{D}} [R(s_i, d) + \alpha V_{n-1}(\phi(s_i, d))]$$

where $\alpha < 1$. We calculate an approximate answer, where the number of terms considered depends on α and the precision required.

The problem becomes more complex when the outcome of a decision is uncertain. In this case, our aim is to maximize the *expected* value of a sequence of decisions. Let $P_{i,j}(d)$ be the probability of ending in state s_j, starting from s_i and making decision d. The recursive definition of the optimization problem for this case is

$$V_n(s_i) = \max_{d \in \mathcal{D}} [R(s_i, d) + \sum_{s_j \in \mathcal{S}} P_{i,j}(d) V_{n-1}(s_j)]$$

Stochastic dynamic programming [Ross, 1983] is a solution methodology for problems of this sort. Constructing a deliberation-scheduling policy using stochastic dynamic programming involves caching optimal decisions for increasingly longer sequences of states, starting in any of the states in $\mathcal{S}$. One potential problem with applying this methodology to time-dependent problems may be the number of states for which decisions have to be calculated and cached. The following properties of the system and the environment may affect the optimal decision, and thus may need to be included in the description of the current state:

- The current time (because deadlines may serve to distinguish between otherwise identical states).
- The types and times of occurrence of previous events.
- Previous allocations to the deliberation procedures.
- The system's internal state, including:
  - The current agenda.
  - The system's expectations on what will happen in the future.

This list is more suggestive than conclusive about the possible computational problems in using dynamic programming on time-dependent problems. Section 3.3.3 provides a simple example in which there is a single processor for deliberation, actions take no time, utility is a simple sum of the responses to individual events, and there are still too many distinct states for dynamic programming to be feasible.

2.5.2 Solving the Optimization Problem

As argued in the previous section, providing an optimal deliberation-scheduling policy for a decision model that makes full use of the information available is likely to be somewhere between difficult and impossible. There are two ways in which we might try to simplify the problem enough that deliberation scheduling becomes feasible:

- Simplify the decision model.
- Construct a deliberation-scheduling policy that makes suboptimal decisions for the decision model we employ.

Simplifying the decision model is the approach taken in Chapters 3 and 4. In those chapters, the decision model includes no expectations on the occurrence of events in the future except for the system's intended actions. Deliberation scheduling is done by constructing an explicit *deliberation schedule* that makes commitments about what deliberation will be done when. The deliberation scheduler adds deliberation events according to the schedule, and generates a new schedule on the occurrence of one of some small set of event types. An optimal deliberation schedule maximizes expected utility, given what the system knows at the time the schedule is constructed. Another possible simplification is to compute expected utility using the means of distributions, rather than the distributions themselves.

We can also make a tradeoff between compile-time or design-time computation and run-time computation. At one extreme are the purely compile-time solutions provided

by dynamic programming. At the other extreme are deliberation schedulers that construct and solve a new optimization problem for every decision. We have already shown that the purely compile-time approach is unlikely to be feasible. Doing all the work at run time is probably also a bad idea, in that we require the run-time component to be fast. The approach that has worked so far, and that we expect will continue to do so, is to find an appropriate balance of the two, depending on the information available ahead of time and the difficulty of the run-time optimization problem. One interesting approach in this middle ground is to construct (at compile time) several deliberation schedulers, each implementing a different deliberation-scheduling policy. At run time, the system chooses among these schedulers, on the basis of the current or anticipated situation.[14]

2.5.3 Calculating Expected Utility from Performance Profiles

Allocating time to anytime decision procedures can affect the system's behavior in complicated ways, depending on the computation a particular anytime decision procedure performs and the current situation. Here are some examples of deliberation procedures that might prove useful but affect the system's behavior in very different ways:

- Deciding to execute an action. The action may be left incompletely specified (e.g., a robot decides to move from A to B but does not choose a path to follow).
- (Re)ordering intended actions.
- Analyzing and interpreting sensor data.
- Projection to refine the system's expectations on future events.

Performance profiles encode expectations that can be used to calculate the expected utility of the resulting behavior. For some problems (e.g., the time-dependent planning problems in Chapter 3), this calculation is straightforward. In Chapter 3, overall utility is calculated by summing the utility of individual actions, each taken in response to a different event. The performance profiles for deliberation encode expected utility directly, rather than encoding expectations on some parameter from which expected utility must be calculated.

Calculating expected utility from performance profiles is not always so easy. Let us look again at the projectile-weapon controller from Section 2.4.3, and this time, let the target be mobile rather than stationary. Information on the target's trajectory is

[14] This is similar to *gain scheduling* in control theory [Bollinger and Duffie, 1988], in which the controller can switch among a set of linear control laws, depending on which one best approximates an optimal controller, given the current state of the system.

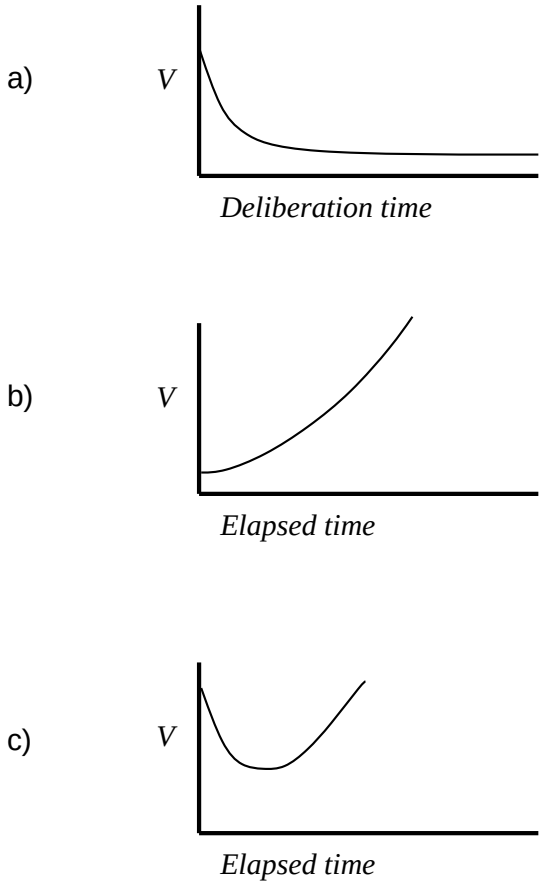

Figure 2.8: Combining error distributions

gathered by a tracking system that feeds data to the controller to use in aiming the gun. Deliberation in this case involves deciding when and where to shoot (we only get one shot), a computation that includes correcting for various factors influencing the gun's aim.

Suppose that we have compiled a performance profile for deliberation, giving a distribution on the error between the coordinates aimed at and where the projectile lands. Figure 2.8a graphs the variance on a gaussian distribution on aiming error as a function of deliberation time. With perfect information on the target's position, this profile can be used to calculate the probability that the projectile will land within some distance ϵ of the target.

Of course, the controller does not have perfect information on the target's position: there is inevitably some error in the estimate provided by the tracking system that increases as time passes since the estimate was provided. In other words, the longer the controller waits to fire, the poorer its estimate of the target's position, and the

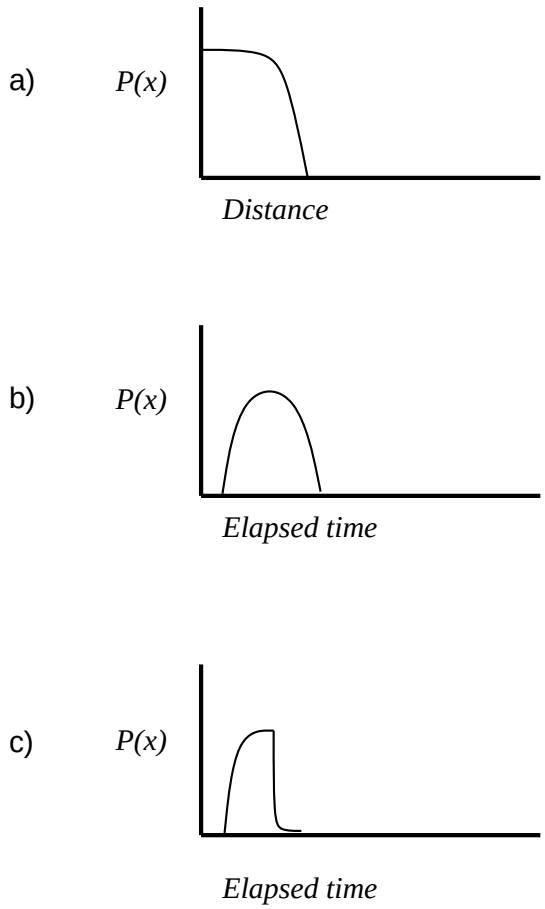

Figure 2.9: Calculating expected utility

greater the probability of missing by some significant amount. Figure 2.8b graphs the variance on a gaussian distribution on tracking error as a function of elapsed time since the tracking system was consulted.

Since aiming error and tracking error are conditionally independent for a fixed value of time, we can sum their variances to get the variance for the distance between the target and where the projectile lands.[15] This variance, graphed in Figure 2.8c, can be used to determine the probability that the projectile will land within some distance ϵ of the target.

What we are ultimately interested in, however, is not how precisely the controller aims the gun. It's whether the target is successfully disabled. Suppose that we know the probability that the target will be disabled as a function of its distance from where the projectile lands. That function might look something like Figure 2.9a. Combining

[15]This is somewhat simplified. The actual errors are vectors.

this information with the graph in Figure 2.8c, we can calculate the probability that the target will be disabled as a function of the time spent deliberating, as in Figure 2.9b.

One final effect must be taken into account: the target may find cover to hide behind. Suppose that the probability of successfully disabling the target drops considerably when it is in a protected position. If the target is approaching some kind of cover, the probability of shooting it down as a function of deliberation time looks something like Figure 2.9c. The expected utility is then the probability of success times the utility of success, plus the probability of failure times the utility of failure. If the utility of failure is 0, then the graph of expected utility as a function of deliberation time is simply the graph in Figure 2.9c, multiplied by a constant.

The profile we started with in Figure 2.8a looks very little like the curve we end up with in Figure 2.9c. Except for problems in which the relationship between the results of deliberation and the utility of the resulting behavior is very direct, this kind of analysis is going to be needed. Furthermore, there are some additional complications we have not considered. For instance, in this problem the terms "utility" and "value" can be used interchangably because the utility function is *risk-neutral*, i.e., maximizing the expected value maximizes the expected utility. This is not always the case. Given a value function $V(x)$ over possible outcomes, utility is defined as a function $U(V(x))$. If U is linear, then maximizing expected value maximizes expected utility. More generally, U is not linear. The classic example involves a lottery, in which you stand to win some amount v with probability p. Suppose that you are offered instead the certainty of gaining the value $v * p$. The expected values are the same. Risk-neutral behavior will find no preference between the lottery and the certain gain. *Risk-averse* behavior would be to prefer the certain gain [Raiffa, 1968]. For a problem where utility is not risk-neutral, we would have to make another calculation, combining value as a function of deliberation time and utility as a function of value to obtain expected utility as a function of deliberation time.

2.5.4 Utility Functions and Deadlines

So far, we have said nothing about the utility function U beyond the fact that it maps from the history describing the system's interaction with the environment to the real numbers. It is clear that the form of the utility function has a profound effect on the difficulty of deliberation scheduling. For example, utility functions that involve summing the results of individual responses (as in Chapter 3) result in easier optimization problems than functions that involve a more complex dependence on what events occur when. Since we are interested in addressing problems that arise from interaction with some real environment, some families of functions seem more likely to arise than others. Accordingly, we would like to restrict our attention to those classes

of functions likely to prove useful.

One possible restriction involves separating the "utility" of an action or a set of actions from the "cost" of doing the actions at particular times. This restriction is not required in order to apply expectation-driven iterative refinement to a given time-dependent problem, but in practice it helps make the optimization problems tractable. One source of this time cost might be deadlines imposed by the environment. Another time-based influence on utility is the information available to the system: a decision made early but based on limited information may not be as useful as one made later when more information is available, even if the cost of delaying action is substantial.

Most of the previous work on time-dependent problems has made a similar split. Locke [1986] is interested in constructing schedules of tasks when there is a value attached to completing a task as close as possible to an intended time. There is no notion of task utility apart from the completion time. Russell and Wefald's work on game-tree search [Russell and Wefald, 1989b] includes a time cost that is inversely related to the distance to a deadline and is independent of the action taken. In Horvitz's work on high-stakes medical decision-making [Horvitz, 1988], overall utility is the sum of an *object-related* or *intrinsic* utility that is independent of time and an *inference cost*. For a more comprehensive survey of related work on time-dependent problems, see Section 1.4.

Defining a general model for time cost is difficult: if it is too general, we will be unable to say anything interesting about using it to solve problems. The key to coming up with useful restrictions is to look at the kinds of time cost that arise in practice. Consider some different kinds of deadlines, for example:

- Additive costs:
  - Bounded step cost (e.g., one-time fine for being late).
  - Bounded linear cost (e.g., fines for an overdue book).
  - Unbounded linear cost (e.g., per-day charge for storage).
  - Unbounded exponential cost (e.g., interest on unpaid taxes).
- Multiplicative costs:
  - Step to some fraction of intrinsic utility (the right answer is worth only half as much if it's late).
  - Step to 0 (the right answer is worth nothing if it's late).
  - Linear decrease in the fraction of intrinsic utility.
- Absolute costs:

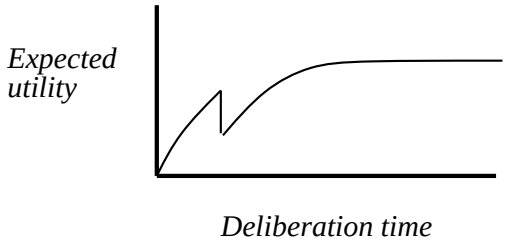

Figure 2.10: Additive time costs

- Bounded step (the library is closed, so we wasted the time to go there, independent of the value of having made it on time).
- Unbounded step (the movie "The Seventh Sign"—Armageddon ensues if a deadline is not met.)

Our original assumption in [Dean and Boddy, 1988] was that combining time-cost and utility for a particular response would yield a function that had a single maximum for expected utility (e.g., Figure 2.9c). This assumption is in general too restrictive, but is reasonable for some problems (e.g., the time-dependent planning problems in Chapter 3) and results in an optimization problem that is relatively easy to solve.

Some of these kinds of time cost can make deliberation scheduling harder. Consider the graph of utility over deliberation time in Figure 2.10. Here, a simple one-time deadline cost has been added to an exponential curve graphing object-related utility as a function of deliberation time. Trying to maximize a sum of such functions where the deliberation allocations are interdependent is a combinatorial problem. Maximizing a similar sum of functions, each with a single maximum, is much easier. Since the constraints on deliberation allocation define a convex polytope, we can find the maximum using gradient descent.[16]

2.6 Summary and Conclusions

This chapter has specified and explored the structure of time-dependent problems. We address issues common to any attempt to solve time-dependent problems using expectation-driven iterative refinement. In the following chapters, we apply the methods developed here to specific examples.

In this chapter, we give a formal specification of what we mean by a time-dependent problem. We also define the form of a solution to a time-dependent problem generated

[16]The constraints in question enforce such restrictions as only deliberating on one thing at a time, and not allocating deliberation time after the answer is to be used or before necessary information is available. See Section 3.6 for more details.

using expectation-driven iterative refinement. These solutions require that we specify a set of anytime decision procedures for doing deliberation and a deliberation-scheduling policy. It turns out that a great number of known algorithms, capable of solving a wide variety of problems, either are already framed as anytime algorithms or can easily be adapted to be anytime algorithms. We also show how to go about combining known anytime algorithms to create new ones suitable for more complex deliberation.

We show that constructing a deliberation-scheduling policy involves solving a sequential decision problem, and that for most time-dependent problems, providing an optimal deliberation-scheduling policy for a reasonably complex decision model is likely to be computationally very difficult. As an alternative strategy, we suggest some ways to restrict the decision model so as to simplify providing a deliberation-scheduling policy. In particular, we discuss eliminating expectations on the occurrence of events and generating an explicit *deliberation schedule* on the occurrence of a restricted set of significant events. Empirical evidence presented in the following chapters of this thesis shows that for some problems, this approach can provide computationally inexpensive deliberation scheduling for which the expected utility is only slightly less than for a more complete (and completely intractable) decision model.

There are several directions in which the work described here could be extended. One direction that seems promising is to extend the start we have made on a general framework for constructing complex anytime decision procedures from simpler ones. Other possibilities include further investigation of compile-time versus run-time tradeoffs, and finding some way to systematize the calculation of expected utility from performance profiles, given the required functions and distributions.

Chapter 3

Time-Dependent Planning

3.1 Introduction

The robot in Figure 3.1 has been assigned the task of classifying and intercepting or rerouting objects on a moving conveyor belt. The robot's view of the conveyor is partially obscured by a partition, and objects emerge from behind this partition at irregular intervals. Between the time each object clears the partition and the time it reaches the end of the conveyor, the robot must classify the object and react appropriately. At any given point, there may be several objects on the conveyor, each requiring a separate disposition. If making this classification requires time-consuming computation, the robot must trade off time spent determining the disposition of one object from time spent on other objects. The number of objects on the belt and the speed at which the belt moves may vary over time, thus providing a wide range of situations in terms of contention for the available computational resources.

Alternatively, suppose a database server has many clients (Figure 3.2), who will wait a known (though not necessarily constant) amount of time for a response, after which time the response has no value. Suppose this server includes a procedure or procedures implementing an anytime algorithm for processing database queries.[1] Given these anytime query-answering procedures, a server faced with a stream of requests must decide on how much time to spend computing each answer, on the basis of the cost of supplying an incomplete answer to any given request or of supplying no answer at all.

These two problems have some common features:

- They are event-driven: events occur that require some response (e.g., a disposition

[1] Vrbsky et al. [Vrbsky *et al.*, 1990] discuss a mechanism for providing partial answers to relational database queries: "partial" in the sense that the set of tuples returned may either contain tuples that do not satisfy the query, or may not contain some of the tuples that do satisfy the query. As a procedure implementing this scheme is allowed to run, the membership of the set of tuples returned converges to the set of tuples satisfying the query.

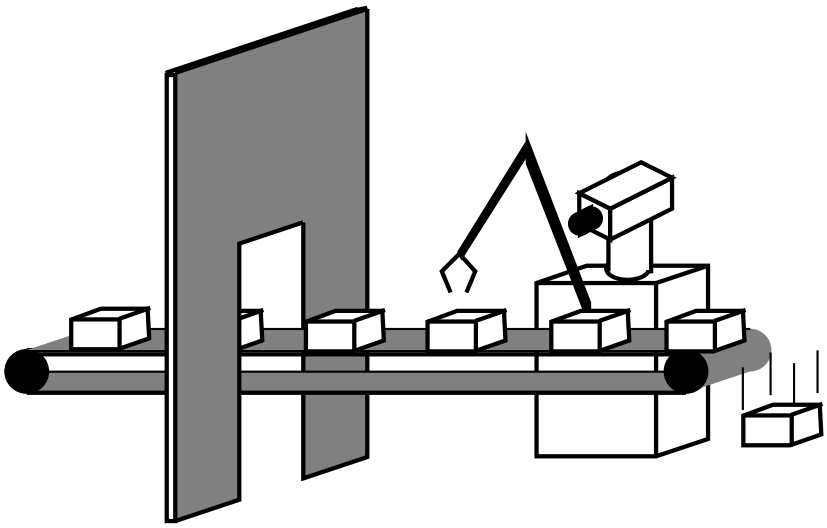

Figure 3.1: A robot and a conveyor belt

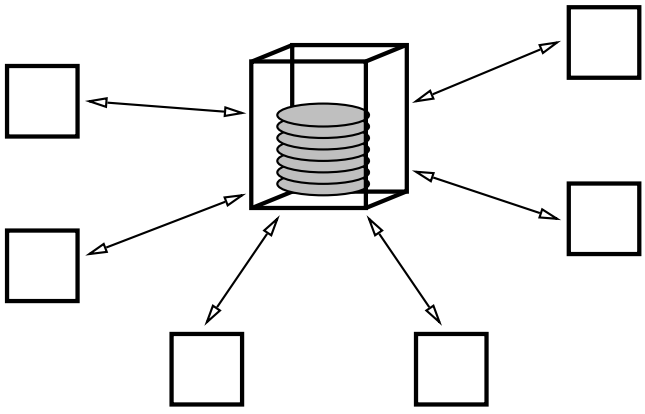

Figure 3.2: A database server

for an object, or an answer to a query). There is a deadline by which the response must be completed; the system may know the deadline exactly, or may have a distribution for it.

- Deadlines are absolute; a late response is of no value.

We assume that the utility of a response to a particular event is independent of the responses to other events, except that time spent on one response cannot be spent on another.

We call problems with these features *time-dependent planning problems.* In Section 3.2, we define time-dependent planning problems formally. We then explore the problem of doing deliberation scheduling for time-dependent planning problems (Section 3.3.3), and present a deliberation scheduler that constructs an explicit deliberation schedule (Section 3.4). These deliberation schedules yield optimal deliberation scheduling for a decision model that does not include expectations on the future occurrence of events requiring responses. In Section 3.6, we compare the cost of using these deliberation schedules to the use of a "perfect-information" deliberation scheduler.

3.2 Formal Statement of the Problem

A time-dependent planning problem consists of:

- A set of event types, $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$.
- A set of *observation* types, $\mathcal{O} \subseteq \mathcal{T}$, corresponding to observations that may be made by the system.
- A set of *condition* types, $\mathcal{C} \subseteq \mathcal{T}$, corresponding to deadlines for a response (the completion of an action) by the system.
- A set of *action* types, $\mathcal{A} \subseteq \mathcal{T}$, defining the types of responses available to the system, including the null action ϕ.
- A set of *histories* $\mathcal{H} \subseteq 2^{\mathcal{T} \times \Re}$. Each element h of $\mathcal{H}$ is a set of ordered pairs $\{(\tau, t) | (\tau \in \mathcal{T}) \wedge (t \in \Re)\}$. We call these pairs *events.* Given an event $e = (\tau, t)$, the notation $\mathsf{type}(e)$ denotes the event type τ and $\mathsf{occ}(e)$ denotes the time of occurrence t relative to some global clock. In addition, $\mathsf{resp}(e)$ denotes the element of h that is an action taken in response to e. If there is no such element, $\mathsf{resp}(e) = \phi$.
- A time t_0 such that $\forall h \in \mathcal{H}, \forall e \in h, t_0 \leq \mathsf{occ}(e)$.

- A function $\mathsf{delay} : \mathcal{A} \rightarrow \Re$. Actions themselves are durationless events, but there may be some delay before an action has a desired effect. We call this delay the *response time*. The time at which an action a has its intended effect is denoted by $\mathsf{comp}(a)$. $\mathsf{delay}(a) = \mathsf{comp}(a) - \mathsf{occ}(a)$. $\mathsf{delay}(\phi) = 0$.

- A set of *external states* $\mathcal{S}^{ext}$. Each external state completely describes the environment at one moment in time. If the system controls some mechanical apparatus (e.g., a robot), the state of that apparatus is included in the external state.

- A set of *internal states* $\mathcal{S}^{int}$. Each internal state is a pair of vectors $\langle V_s, V_d \rangle$. V_s is the *sensor data*; its value depends only on the initial state and the events that have occurred. Deliberation has no effect on V_s. V_d is the *deliberation scratchpad*, space used to maintain data structures that may result from deliberation. Deliberation is the only thing that affects V_d.

- A set of *initial states* $\mathcal{I}$. Each initial state i is an ordered pair $\langle s_{int}, s_{ext} \rangle$. s_{int} specifies the system's internal state at t_0, and might, for example, include the system's *a priori* expectations on the occurrence of events. s_{ext} specifies the external state at t_0.

- A function $u : \mathcal{C} \times \mathcal{A} \rightarrow \Re$ giving the utility of an action of type α taken in response to, and completed before, a condition of type c.

- A function $U : \mathcal{H} \rightarrow \Re$.

$$U(h) = \sum_{\{e \in s | \mathsf{type}(e) \in \mathcal{C}\}} \begin{cases} u(\mathsf{type}(e), \mathsf{type}(\mathsf{resp}(e))) & \text{if } \mathsf{comp}(\mathsf{resp}(e)) < \mathsf{occ}(e) \\ 0 & \text{otherwise} \end{cases}$$

A *solution* to a time-dependent planning problem includes the following elements:

- A set of *anytime decision procedures* $\mathcal{P}$ to which deliberation time may be allocated. In most of this chapter, we consider problems in which running a single anytime decision procedure affects only the response to a single condition.

- An *agenda* of intended actions. The agenda is an ordered list of events corresponding to actions the system has committed itself to taking at the indicated time. Actions are added to the agenda only by deliberation. As time progresses ($\hat{t}$ increases), actions on the agenda are taken at the appropriate time.

- A *dsp*. This policy is implemented by a deliberation scheduler that makes allocations to the elements of $\mathcal{P}$.

In order to include deliberation in the passage of time, we define a set of event types $\mathcal{D}$ to be added to $\mathcal{T}$, each corresponding to the starting point of a period over which deliberation is performed on a specified element of $\mathcal{P}$. These events are "actions" in the sense that they are under the system's control. We do not include them in $\mathcal{A}$ because they cannot serve as responses to conditions.

The system performs an action with a type in $\mathcal{A}$ or $\mathcal{D}$ by specifying that an event of that type is a part of every element of the current active histories $\mathcal{H}_{\hat{t}}$, with a time of occurrence equal to $\hat{t}$. This is the only control the system has over the incremental winnowing of $\mathcal{H}_{\hat{t}}$ (i.e., over the realized history h^*). The time required to choose an action in $\mathcal{D}$ is not accounted for in the passage of time represented by the increasing value of $\hat{t}$. The computation leading to the choice of actions in $\mathcal{A}$ is done using the anytime procedures in $\mathcal{P}$, and is thus accounted for, because the time required is represented by the occurrence of events with types in $\mathcal{D}$. In other words, the deliberation leading to a choice of actions in the world is accounted for, while the allocation of deliberation time (meta reasoning) is assumed to happen "in the noise." There may be time-dependent problems for which good deliberation allocation is too expensive for this assumption to be useful. We suggest some ways in which the cost of deliberation allocation can be ameliorated or accounted for in Section 2.5.

3.3 Analysis of the General Case

In this section we examine some of the implications of the above definition for time-dependent planning problems. In particular, we are concerned with calculating the expected utility of the system's behavior for a particular deliberation scheduling strategy.

3.3.1 Projection

Observations are used by the system to make predictions about the future occurrence of events, in particular about the occurrence of conditions requiring a response. For the class of problems analyzed in the rest of this chapter, we make the following assumptions about the nature of observations:

- Each observation can be used to predict the occurrence of a single event (condition) of known type and time of occurrence. In other words, any observation that has occurred (has a time of occurrence less than $\hat{t}$) can be used to constrain the current active histories to be only those histories that include the corresponding condition.
- The computation needed to do this prediction is negligible.

More generally, observations may not completely constrain the type or time of occurrence of the corresponding condition, or there may not be a simple mapping from observation to condition. Observations can be viewed as information that can be used to modify expectations on the type and time of the events that may occur. Some of the implications of this more general formulation are discussed in Section 3.7.1.

3.3.2 Calculating Expected Utility

The history h^* that is eventually realized is in general be unknown *a priori*: the system does not know precisely what types of events will occur when. Observations provide information that constrains the range of possibilities. The system cannot predict what actions it may take as the result of decisions it has not yet made. The response time $\mathsf{delay}(a)$ for an action a may be uncertain, even when the type of a is known.

For each condition type $c \in \mathcal{C}$, the system can choose an anytime decision procedure for deciding on a response to an event of that type. Each decision procedure, when allocated some amount of time, returns some $a \in \mathcal{A}$ corresponding to its best guess for the proper response. We define φ from $\mathcal{C} \times \mathcal{P} \times \Re^+$ to $\mathcal{R}$ so that for each $c \in \mathcal{C}$, $p \in \mathcal{P}$, and positive real number δ, $\varphi(c, p, \delta) = a$, where a is the system's best guess about the appropriate response to a condition of type c having spent δ in deliberation using decision procedure p.[2]

We can use information about the current set of active histories and a distribution over its elements (i.e., probabilities that particular histories will be realized), to simplify calculating expected utility. Suppose that all the conditions that will occur are known.[3] In this case, the only uncertainty in the history is the actions the system will decide upon as responses to those conditions. For a particular set of active histories H, each member of H contains the same set of conditions $\{c_1, c_2, \ldots, c_k\}$. In this case:

$$E(U) = \sum_{1}^{k} E(u(\mathsf{type}(c_i), \varphi(\mathsf{type}(c_i), p_i, \delta_i))) * P(\mathsf{comp}(\mathsf{resp}(c_i)) < \mathsf{occ}(c_i))$$

given that the system decides to allocate δ_i units of time to procedure p_i for deliberating about the response to c_i. In other words, the overall expected utility is the sum of the expected utilities of the original responses, multiplied by the probability that the responses will be completed before their respective deadlines. The utilities of the individual responses are independent, given a fixed allocation of deliberation time. When a response completes ($\mathsf{comp}(\mathsf{resp}(c_i))$) is determined by $\mathsf{occ}(\mathsf{resp}(c_i))$, which the

[2] The system in general has expectations on φ (encoded in performance profiles), but is unlikely to know φ precisely.

[3] This corresponds to a situation in which the "observations" provide perfect information and all occur at the very beginning of the history.

system can determine, and by delay(type(resp(c_i))) (the response time), which it cannot determine and may know only approximately. If the response time is precisely known, we have

$$E(U) = \sum_{1}^{k} E(u(\mathsf{type}(c_i), \varphi(\mathsf{type}(c_i), p_i, \delta_i)))$$

The expected value is then:

$$\mu(c, p, \delta) = E(u(\mathsf{type}(c), \varphi(c, p, \delta))).$$

This is the parameter for which we gather statistics on the anytime decision procedures used to select responses.

For time-dependent planning problems, performance profiles can be used directly to calculate expected utility. Each decision procedure run results in a response to a particular event. Each response has a utility dependent only on the relevant event and when the response is executed. Thus we can compile performance profiles for decision procedures where the parameter whose expected value is being tracked is utility (μ in the equation above), rather than some other parameter from which utility must be calculated.[4] In the rest of this chapter, we refer to the "expected utility" resulting from a given allocation of deliberation time to a particular anytime decision procedure, though in discussions of more general time-dependent problems this terminology would not be appropriate.

3.3.3 Optimal Decisions

We wish to maximize the expected utility resulting from the allocation decisions made by the system. The basic decision to be made is made repeatedly: for each increment of time, which of the available decision procedures will be run? This is a *sequential decision problem* of a form commonly addressed in operations research [Ross, 1983]. In that field, an *optimal decision policy* is defined as a policy that at each point makes the decision maximizing the expected utility of the current decision, given expectations on the decisions yet to be made.[5]

Consider the two examples at the start of this chapter. For the conveyor belt, there is some earliest point at which an object's existence and position on the belt can be determined. For the database server, queries cannot be predicted (other than in a statistical sense) before they are made. Each moment of time that passes provides new information in terms of the events that did (or didn't) happen. For time-dependent planning problems, this new information could take any of the following forms:

[4] Chapters 4 and 5 disuss problems in which expected utility is a more complex function of time allocated to the decision procedures.

[5] Infinite sequences of decisions are handled using *discounting*, in which the expected reward resulting from a particular decision is weighted inversely to how far in the future the decision is.

- An event (observation or condition) occurred.
- An event (observation or condition) failed to occur.
- New information on an anytime decision procedure became known:
  - An anytime decision procedure finished, in the sense that there is known to be no benefit to allocating additional time. The system may or may not be able to detect this.
  - An anytime decision procedure did not finish when expected.
  - The intermediate result provided by an anytime algorithm was better/worse/different than expected. Again, the system may or may not know enough about the anytime algorithm's behavior to detect this.

A deliberation scheduler that does not take this new information into account in making allocation decisions may not do as well as one that does (the expected utility of the resulting behavior may be lower). In this and succeeding sections of this chapter, we assume that the system has no information about the operation of the available anytime decision procedures beyond the expectations contained in the performance profiles (e.g., the system has no way of telling whether a given decision procedure has finished, in the sense given above). Thus the only new information available as time passes comes from the occurrence of observations.

Suppose we have the situation depicted in Figure 3.3a. Two conditions, c_1 and c_2, have been predicted. The corresponding performance profiles are described by the following functions:

$$c_1 : \qquad \mu_1(\delta) = \begin{cases} \delta, & \delta < 10 \\ 10, & \delta \geq 10 \end{cases}$$

$$c_2 : \qquad \mu_2(\delta) = \begin{cases} 10\delta, & \delta < 8 \\ 80, & \delta \geq 8 \end{cases}$$

The shaded areas show optimal deliberation allocations for these conditions, assuming that no more observations or conditions occur.

Now suppose that the situation is as in Figure 3.3b, where the observation o_3 and condition c_3 may occur with probability p, but only at the indicated times. o_3 is a perfect predictor for c_3 in the sense that once o_3 occurs (or does not occur), the system knows with certainty whether c_3 will occur. c_3's performance profile is given by:

$$\mu_3(\delta) = \begin{cases} 12\delta, & \delta < 10 \\ 120, & \delta \geq 10 \end{cases}$$

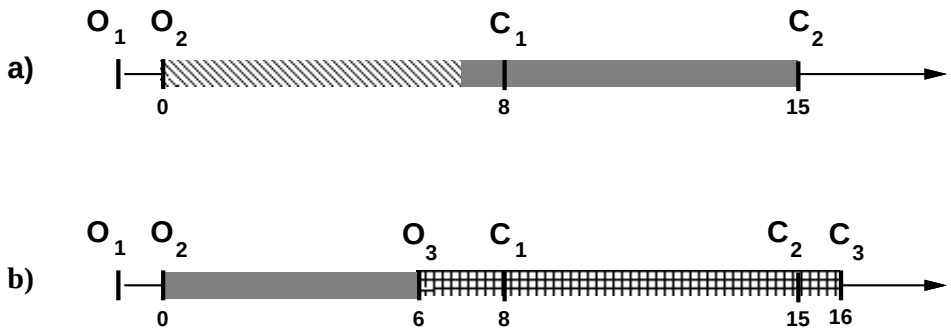

Figure 3.3: Optimal deliberation allocation

An optimal schedule starting at $\hat{t}$, had the system known that c_3 would occur, is shown by the shading. Comparing this schedule with the one in Figure 3.3a suggests that a system that does allocation using only what it knows, rather than expectations of what will happen, will sometimes deliberate on the wrong condition.

Once o_3 has occurred, the system has all the knowledge needed to make optimal use of the remaining deliberation time. The interesting questions concern how to allocate the time between $\hat{t}$ and $\mathsf{occ}(o_3)$. This decision should be made on the basis of expectations on what will happen at $\mathsf{occ}(o_3)$ (i.e., whether o_3 happens or not). The question we seek to answer is: what should the allocation of time to c_1 and c_2 before $\mathsf{occ}(o_3)$ be, given that o_3 occurs with probability p? Let $\delta_{1,1}$ be the time spent on c_1 before $\mathsf{occ}(o_3)$, $\delta_{1,2}$ be the time spent on c_1 after $\mathsf{occ}(o_3)$, and analogously for $\delta_{2,1}$, $\delta_{2,2}$, and c_2. Then the quantity we seek to maximize is:

$$\delta_{1,1} + 10\delta_{2,1} + p(120) + (1-p)(\delta_{1,2} + 10\delta_{2,2})$$

The first two terms are the expected utility resulting from allocations to c_1 and c_2 before $\mathsf{occ}(o_3)$. The last two are the payoffs resulting from the occurrence of o_3 (in which case all the remaining time is allocated to c_3) or from allocating the remaining time to c_1 and c_2. There are some constraints on the values for $\delta_{1,1}$, $\delta_{2,1}$, $\delta_{1,2}$, and $\delta_{2,2}$:

$$\begin{aligned}
&\delta_{1,1} + \delta_{2,1} \leq 6 \\
&\delta_{1,1} + \delta_{1,2} \leq 10 \\
&\delta_{1,2} \leq 2 \\
&\delta_{2,1} + \delta_{2,2} \leq 8 \\
&\delta_{1,1} + \delta_{2,1} + \delta_{1,2} + \delta_{2,2} \leq 15 \\
&\delta_{1,2} + \delta_{2,2} \leq 9
\end{aligned}$$

If $p = .5$, the optimal allocations are

$$\delta_{1,1} = 0$$
$$\delta_{2,1} = 6$$

If o_3 does not happen, the allocations after that point are:

$$\delta_{1,2} = 2$$
$$\delta_{2,2} = 2$$

The expected utility for this allocation is

$$0 + 6(10) + .5(120) + .5(2 + 2(10)) = 131$$

If $p = 0.05$, the optimal allocations are:

$$\delta_{1,1} = 6$$
$$\delta_{2,1} = 0$$

If o_3 does not happen, the allocations after that point are:

$$\delta_{1,2} = 1$$
$$\delta_{2,2} = 8$$

The expected utility for this allocation is

$$6 + 0(10) + .05(120) + .95(1 + 8(10)) = 88.95$$

The optimal policy for the first decision (fixing $\delta_{1,1}$ and $\delta_{2,1}$ at time 0) is a single test and two alternatives: if $p > .1$, schedule as though o_3 will occur. If $p \leq .1$, schedule as though it will not.

Optimal deliberation scheduling is a sequential decision problem, in which each allocation decision depends on expectations on what will happen in the future. For this example, the sequence consists of two decisions—one made at $\hat{t}$, and one made at $\mathsf{occ}(o_3)$. This makes the problem artificially easy. o_3's failure to occur conveys as much useful information as its occurrence. In a less restricted instance, observations could occur (or not occur) not just at a single point, but for any instant in time. Each such instant thus provides new information (a new *state*) and requires a new allocation decision.

Following methods used in operations research, we might try using *dynamic programming* [Bellman, 1957, Ross, 1983] to solve our optimization problem.[6] Dynamic programming involves caching the optimal answer to a series of subproblems of increasing size, until finally we can determine the optimal answer to the full problem. For

[6]The analysis performed for the example above is a simple form of dynamic programming.

problems involving a sequence of decisions made over time, these subproblems will be subsequences of decisions, starting from a particular state. We could start with the last tick in some history, for example, caching the optimal allocation for that tick for any of the possible states, and then move back to consider the next-to-last tick. Our problem is complicated by the fact that we have only expectations on the schedule of events and the results of decisions. This complicates the computation by requiring us to calculate expected utility based on what *may* happen for each sequence of decisions (typically a set of possibilities), rather than from a single deterministic outcome.

Calculating a policy for a sequential decision model requires reasoning about expectations from a given state of the world. For time-dependent planning problems, the current state is determined by:

- The types and times of occurrence for predicted events that have not occurred yet.
- Any deliberation that has already been done on these events.
- State-specific expectations on the future occurrence of events.

Even if times of occurrence and deliberation allocations are integer-valued, there may be a very large set of possible states. While some of these states may be equivalent from the point of view of allocating deliberation time, determining that may not be easy. Certainly for the example we have gone through, changing the type or time of occurrence of c_1 or c_2 or the amount of deliberation already done on either condition could change the optimal allocation.

3.4 Constructing Deliberation Schedules

In this section and the next, we explore the construction of explicit *deliberation schedules* specifying the allocation of deliberation time for the conditions for which observations have occurred. These schedules are optimal for a decision model that does not include expectations on the future occurrence of conditions requiring responses. The deliberation scheduler constructs a deliberation schedule whenever new observations occur, and between these occurrences simply follows the last schedule generated. In Section 3.6, we demonstrate that simplifying the decision model used for deliberation scheduling in this way leads to only a small decrease in the expected utility of the allocations made by the deliberation scheduler.

In this section, we present a deliberation-scheduling algorithm for a simple class of time-dependent planning problems restricted by the following assumptions:

- Response time for actions is zero (actions take no time).

- Prediction is perfect (given a set of observations, we know with certainty what events will happen and when). One way to model this would be to have the sensor data V_s consist of a schedule of predicted events. A more realistic model might have the sensor data report the occurrence of observations, from which deliberation procedures construct the schedule.

- Only one anytime decision procedure can be used to generate a response for any given condition (i.e., there is a function mapping from condition types to decision procedures).

We refer to this class of time-dependent planning problems as the *Basis Problem*. All of the problem classes investigated in this chapter are extensions to the Basis Problem.

As we saw in the previous section, performance profiles for the anytime decision procedures used in time-dependent planning problems graph expected utility against deliberation time. The instantaneous change in utility with time allocated is called the *gain:*

$$\gamma(c, p, \delta) = \frac{\partial \mu(c, p, \delta)}{\partial \delta}$$

The utility of the answer returned by an anytime decision procedure is expected to change with increasing allocation time in a predictable way: the answer will keep getting better. Typically there is some "best" answer that the answer returned will approach. We add a further restriction:

- The gain γ does not increase for increasing δ (diminishing returns).

This restriction is motivated by the recognition of the fact that the cliché about 90% of the work taking 10% of the time is frequently true. Most (though not all) of the anytime algorithms listed in Section 2.4 have this property. Since our concern is with domains where deliberation time is both limited and uncertain, an algorithm with this property is certainly preferable.[7] Figure 3.4 shows some examples of the kinds of profiles that meet the restrictions we have imposed.

3.4.1 A Deliberation-Scheduling Procedure (DS-1)

In this section, we present a procedure that constructs optimal deliberation schedules for the Basis Problem for a decision model that includes the assumption that no more events will occur. For a given value of $\hat{t}$, there is a set of conditions, $\mathcal{L}_t$, that have been predicted (the corresponding observations have occurred) but have not yet occurred. Constructing a deliberation schedule involves determining how to allocate deliberation

[7] The problem of allocating deliberation to anytime decision procedures with increasing gain is addressed in Section 2.5.

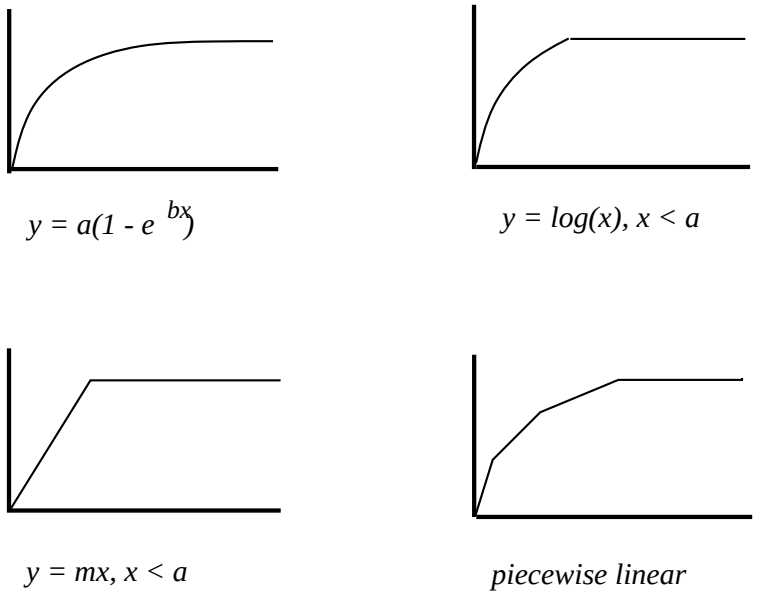

Figure 3.4: Performance profiles for anytime decision procedures

time among the conditions in $\mathcal{L}_t$. The deliberation schedule generated maximizes the expected utility of the responses to the conditions in $\mathcal{L}_t$, assuming that no further conditions occur.

Let the sequence $c_1, \ldots c_n$ be the conditions in $\mathcal{L}_t$ such that $\mathsf{occ}(c_i) \leq \mathsf{occ}(c_{i+1})$. Let δ_i be the deliberation time allocated to c_i. Let $\mu_i(\delta) = \mu(c_i, p_i, \delta)$ be the expected utility of the response to c_i given δ spent on deliberation, and $\gamma_i(\delta) = \gamma(c_i, p_i, \delta)$ be the gain.[8] Let $\mathcal{S}_d$ be the initially empty set of deliberation events that the system is committed to performing (the deliberation schedule). As time advances ($\hat{t}$ increases), the events in $\mathcal{S}_d$ will be added to the evolving history. "Deliberation events" are events marking the beginning of an interval over which deliberation is performed using a specified anytime decision procedure. Let $d_i \in \mathcal{D}$ be the type of an event marking the start of deliberation on the response to c_i.

Given the time $\hat{t}$ and a set of conditions $\mathcal{L}_t$, the procedure **DS-1** in Figure 3.5 generates a deliberation schedule starting at $\hat{t}$ and ending at $\mathsf{occ}(c_n)$, the time of occurrence of the last condition in $\mathcal{L}_t$. The first loop of the procedure (1) iterates back over the time between $\mathsf{occ}(c_n)$ and $\hat{t}$, adding increments of a fixed size Δ to δ_i for the condition c_i with the highest gain $\gamma_i(\delta_i)$, given the allocations made so far. The actual schedule is constructed in the second loop (2). All the deliberation allocated for each condition is scheduled in a contiguous block of time. Let $s_i \in \mathcal{S}_d$ be the event starting deliberation

[8] The anytime decision procedure p_i is uniquely determined by the type of c_i.

```
Procedure DS-1(t̂, L_t):

(1)  for i = n downto 1
         δ_i ← 0
         start ← (if i = 1, then t̂, else occ(c_{i-1}))
         stop  ← occ(c_i)
         incr  ← ⌊(stop − start)/Δ⌋
         for cnt = 1 to incr
             k ← arg max_{i ≤ j ≤ n} γ_j(δ_j)
             δ_k ← δ_k + Δ
         k ← arg max_{i ≤ j ≤ n} γ_j(δ_j)
         δ_k ← δ_k + stop − start − (incr * Δ)
     t ← t̂
     S_d ← {}
(2)  for i = 1 to n
         S_d = S_d + {(d_i, t)}
         t = t + δ_i
```

Figure 3.5: Generating a deliberation schedule for the Basis Problem

on c_i. The schedule constructed by the loop (2) in **DS-1** is such that there is only one such event for each condition in $\mathcal{L}_t$.

For any $\epsilon > 0$, there is some $\Delta > 0$ such that this deliberation-scheduling algorithm is optimal within ϵ. By optimal, we mean that no other deliberation-scheduling algorithm can produce a schedule with a higher expected utility, where expected utility is calculated assuming that no further events occur.

Theorem 1 *For any $\epsilon > 0$, there is some $\Delta > 0$ such that* **DS-1** *generates a schedule within ϵ of optimal for the conditions in $\mathcal{L}_t$.*[9]

One point to note about the algorithm implemented by **DS-1** is that it must examine the entire current schedule of predicted conditions to generate an optimal deliberation schedule. This is related to the fact that a deliberation scheduler that runs **DS-1** on the occurrence of new observations and follows the resulting schedules makes allocations that are suboptimal from the point of view of an observer knowing the entire history h^* (from hindsight, for example).

[9]This theorem is proved in Section 3.8.

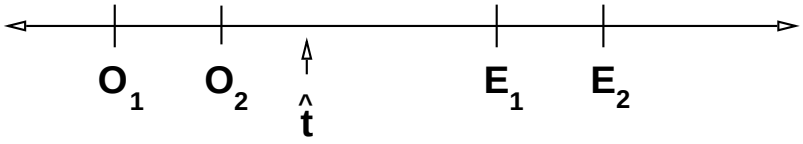

Figure 3.6: A time-dependent planning problem

Figure 3.7: Performance profiles

3.4.2 An Example of Deliberation Scheduling

Consider the timeline in Figure 3.6. At time $\hat{t}$, the system knows about two conditions, E_1 and E_2, from the occurrence of the observations O_1 and O_2. Suppose that the anytime decision procedures for E_1 and E_2 are as in Figure 3.7. We will trace the operation of the procedure **DS-1** as it constructs a deliberation schedule for this case.

At $\hat{t}$, $\mathcal{L}_t = \{E_1, E_2\}$. The algorithm starts by considering the time between E_1 and E_2. This time can only be used for E_2, and each increment is allocated accordingly. So at the end of the first iteration of the loop (1), $\delta_1 = 0$ and $\delta_2 = \mathsf{occ}(E_2) - \mathsf{occ}(E_1)$. The second iteration is slightly more complicated in that two possible allocations for each increment are considered. Since the gain for E_2 is greater than for E_1, increments are added to δ_2 until $\delta_2 \geq \delta_2^*$, the maximimum useful allocation. Note the error arising from the fact that the interval width Δ does not divide δ_2^* evenly; at this point, any further increments are allocated to E_1. Because there is not enough time available, $\delta_1 < \delta_1^*$.

The loop (2) schedules deliberation time, resulting in the projected history in Figure 3.8, where the shaded areas indicate what deliberation is scheduled when, with deliberation for E_1 being performed first.

3.5 Analytic Solutions for Special Cases

The procedure described in the last section generates an optimal deliberation schedule for a wide range of types of anytime decision procedures. As long as the expected

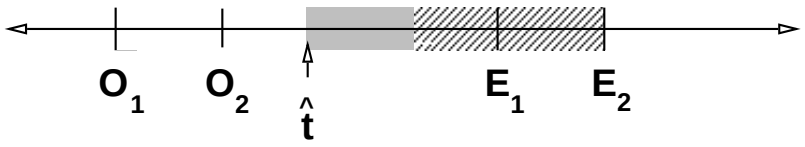

Figure 3.8: The deliberation schedule

utility of the answer returned increases smoothly with deliberation time and the gain does not increase at all, **DS-1** provides an optimal deliberation schedule. However, the algorithm involves a tradeoff between efficiency and accuracy that is dependent on the interval width Δ. Making Δ small enough to guarantee an acceptable deliberation schedule may make the procedure take a long time to run. Since we have framed time-dependent planning problems in such a way as to ignore the cost of deliberation scheduling, this is potentially a problem.

If the behavior of the anytime decision procedures given in a particular time-dependent planning problem can be described with more restricted kinds of performance profiles, we can provide alternative deliberation-scheduling algorithms that generate deliberation schedules more efficiently.

3.5.1 Linear Programming

For time-dependent planning problems where the performance profiles are piecewise linear with only two linear pieces, as in Figure 3.9, generating an optimal deliberation schedule can be formulated as a linear programming problem. Let γ_i be the expected gain in utility per unit time spent deliberating on c_i, δ_i be the time allocated to c_i, and δ_i^* be the upper bound on δ_i (the point past which the expected gain in utility is zero). Given a set $\mathcal{L}_t$ of conditions $\{c_1, \ldots, c_n\}$ such that $\mathsf{occ}(c_{i-1}) \leq \mathsf{occ}(c_i)$, providing an optimal deliberation schedule is equivalent to maximizing the function $f(\delta_1, \ldots, \delta_n) = \sum_1^n \gamma_i \delta_i$, subject to the constraints:

- $\delta_i \geq 0, \ \ 1 \leq i \leq k$
- $\delta_i \leq \delta_i^*, \ \ 1 \leq i \leq k$
- $\sum_1^i \delta_j \leq \mathsf{occ}(c_i) - t, \ \ 1 \leq i \leq k$

This approach works only for problems in which the performance profiles have this simple piecewise linear form. As we show in the next section, however, it is sometimes possible to provide an analytic solution for a smaller piece of the deliberation scheduling problem: allocating the time between two conditions, given the allocations that have been made so far.

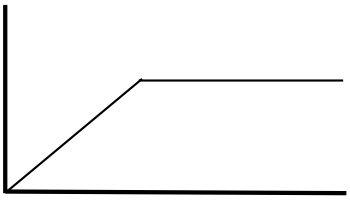

Figure 3.9: Simple piecewise linear performance profile

3.5.2 An Alternative Way of Generating Deliberation Schedules

The algorithm implemented by **DS-1** requires only a small set of assumptions about the behavior of the anytime decision procedures being scheduled. In particular, there is no requirement that the expected behavior of the anytime decision procedures (the performance profiles) be described (or describable) in a specific manner. For **DS-1**, it is sufficient to have a box that returns the gain $\gamma(\delta)$ for a given anytime decision procedure for a given allocation of deliberation time. If the performance profiles can be described in terms of known functions, then it may be possible to use the alternative procedure described in this section. This procedure has two advantages: it iterates only over the number of conditions in $\mathcal{L}_t$, not over each increment of time allocated, and it solves the problem exactly, without errors from piecewise allocation to intervals of finite width. Like **DS-1**, this procedure generates optimal deliberation schedules.

Again, for a given value of $\hat{t}$, the system has a set of conditions, $\mathcal{L}_t$, that it knows about. Let the sequence $c_1, \ldots c_n$ be the conditions in $\mathcal{L}_t$ such that $\mathsf{occ}(c_i) \leq \mathsf{occ}(c_{i+1})$. Let δ_i be the deliberation time allocated to c_i. Let $\mu_i(\delta)$ be the expected utility of the response to c_i, given δ spent on deliberation, and let $\gamma_i(\delta)$ be the gain. Let $\mathcal{S}_d$ be the initially empty set of deliberation events that the system is committed to performing (the deliberation schedule); as time advances ($\hat{t}$ increases), the events in $\mathcal{S}_d$ are added to the evolving history. Let the type $d_i \in \mathcal{D}$ be the type of an event marking the start of deliberation on the response to c_i.

The first loop of the procedure (1) iterates over the events in $\mathcal{L}_t$. For each event c_i, the optimal allocation of the time between $\mathsf{occ}(c_i)$ and $\mathsf{occ}(c_{i-1})$ is determined in the while loop (2) using the procedure **allocate** described in more detail below. The actual schedule is constructed in the second for loop (3). All of the deliberation allocated for each condition is scheduled in a contiguous block of time.

The operation of **DS-2** is much like that of **DS-1**. Time is allocated sweeping back over a set of predicted conditions, and then is scheduled in contiguous blocks. The difference is that **DS-2** allocates the time between conditions in larger chunks, iterating at most $O(n)$ times in the loop (2), where n is the number of conditions in $\mathcal{L}_t$. The procedure **allocate** in **DS-2** determines the allocation of the interval between $\mathsf{occ}(c_i)$

```
Procedure DS-2(t̂, L_t):

(1)  for i = n downto 1
         δ_i ← 0
         start ← (if i = 1, then t̂, else occ(c_{i-1}))
         stop  ← occ(c_i)
(2)      while (start ≠ stop)
             γ* ← max i ≤ j ≤ n γ_j(δ_j)
             max ← {c_j | i ≤ j ≤ n ∧ γ_j(δ_j) = γ*}
             γ' ← max c_j ∈ {c_i, ... c_n} − max γ_j(δ_j)
             {a_1, ..., a_|max|} ← allocate(max, γ', stop − start)
             for k = 1 to |max|
                 δ_{j_k} ← a_κ
             stop ← stop − Σ_1^|max| a_j
     t ← t̂
     S_d ← {}
(3)  for i = 1 to n
         S_d = S_d + {(d_i, t)}
         t = t + δ_i
```

Figure 3.10: The deliberation scheduling algorithm **DS-2**

and $\mathsf{occ}(c_{i-1})$. The exact form of this procedure depends on how the performance profiles are described, but its basic operation is the same. The procedure takes as arguments:

- A set max consisting of the conditions in $\{c_i, \ldots c_n\}$ with a maximal gain.
- A value γ' that is the maximal gain among the remaining elements of $\{c_i, \ldots c_n\}$.
- An interval of time to be allocated, initially equal to $\mathsf{occ}(c_i) - \mathsf{occ}(c_{i-1})$.

The idea is to allocate maximal additional increments $a_1, \ldots, a_{|\mathsf{max}|}$ to the elements of max such that:

- $\forall c_{j_\eta}, c_{j_\kappa} \in \mathsf{max},\ \gamma_{j_\eta}(\delta_{j_\eta} + a_\eta) = \gamma_{j_\kappa}(\delta_{j_\kappa} + a_\kappa)$ — all of the resulting gains are equal.
- $\forall c_{j_\eta} \in \mathsf{max},\ \gamma_{j_\eta}(\delta_{j_\eta} + a_\eta) \geq \gamma'$ — all of the resulting gains are at least equal to the next largest gain.
- $\sum_1^{|\mathsf{max}|} a_\eta \leq$ stop - start — allocate no more than the time available.

The first restriction follows from the fact that allocating time to a condition with a lower gain yields a smaller expected utility (see the proof of optimality for **DS-1**). The second condition follows from the same property, and ensures that the membership of max is updated as gains decrease with increasing allocation. The third condition keeps the procedure from allocating more time than is actually available.

Theorem 2 DS-2 *generates an optimal deliberation schedule for* $\mathcal{L}_t$.[10]

If we can transform the restrictions on $a_1, \ldots a_{|\mathsf{max}|}$ into a linear programming problem, then writing the procedure **allocate** will be straightforward. We present three examples of types of performance profiles for which this can be done.
1) Functions of the form

$$f(x) = \alpha * (1 - e^{-\lambda x}),\ \lambda > 0$$

The necessary restrictions are:

- $\forall c_{j_\eta}, c_{j_\kappa} \in \mathsf{max},\ \gamma_{j_\eta}(\delta_{j_\eta} + a_\eta) = \gamma_{j_\kappa}(\delta_{j_\kappa} + a_\kappa)$ — all of the resulting gains are equal.
- $\forall c_{j_\eta} \in \mathsf{max},\ \gamma_{j_\eta}(\delta_{j_\eta} + a_\eta) \geq \gamma'$ — all of the resulting gains are at least equal to the next largest gain.

[10] This theorem is proved in Section 3.8.

- $\sum_1^{|\mathsf{max}|} a_\eta \leq$ stop - start — allocate no more than the time available.

The gain is given by: $\gamma_i(\delta_i) = \alpha_i \lambda_i e^{-\lambda_i \delta_i}$. Taking the natural log preserves inequalities, and we get:

$$\ln \gamma_i(\delta_i) = \ln(\alpha_i \lambda_i) - \lambda_i \delta_i$$

Substituting back into the constraints given above we get the set of inequalities:

- $\forall c_{j_\eta}, c_{j_\kappa} \in \mathsf{max},\ \ln(\alpha_{j_\eta} \lambda_{j_\eta}) - \lambda_{j_\eta}(\delta_{j_\eta} + a_\eta) = \ln(\alpha_{j_\kappa} \lambda_{j_\kappa}) - \lambda_{j_\kappa}(\delta_{j_\kappa} + a_\kappa)$.
- $\forall c_{j_\eta} \in \mathsf{max},\ \ln(\alpha_{j_\eta} \lambda_{j_\eta}) - \lambda_{j_\eta}(\delta_{j_\eta} + a_\eta) \geq \ln \gamma'$.
- $\sum_1^{|\mathsf{max}|} a_\eta \leq$ stop - start.

The objective function we seek to maximize is $\sum_1^{|\mathsf{max}|} a_\eta$, a linear program. For performance profiles with this form, the procedure **allocate** can also be implemented using a simple iterative technique (iterating over conditions, not time points) that calculates the optimal allocation for the interval between pairs of conditions repeatedly, sucessively eliminating conditions that receive no allocation of time.
2) Bounded logarithmic functions of the form

$$f(x) = \begin{cases} \lambda \log x, & x < x^* \\ \lambda \log x^*, & x \geq x^* \end{cases}$$

If $\mu_i(\delta_i)$ is a function of this form, then the gain $\gamma_i = \lambda_i / \delta_i$ for $\delta_i < \delta_i^*$ and 0 otherwise. The resulting linear program is to maximize $\sum_1^{|\mathsf{max}|} a_\eta$, subject to:

- $\forall c_{j_\eta}, c_{j_\kappa} \in \mathsf{max},\ \lambda_{j_\eta}(\delta_{j_\eta} + a_\eta) = \lambda_{j_\kappa}(\delta_{j_\kappa} + a_\kappa)$.
- $\forall c_{j_\eta} \in \mathsf{max},\ \lambda_{j_\eta}(\delta_{j_\eta} + a_\eta) \leq \gamma'$.
- $\sum_1^{|\mathsf{max}|} a_\eta \leq$ stop $-$ start.
- $\forall c_{j_\eta},\ \delta_{j_\eta} < \delta_{j_\eta}^*$ (never allocate more than the maximum useful time for any condition).

3) General piecewise linear functions (Figure 3.11). If $\mu_i(\delta_i) = C_{i,\delta_i} \delta_i$ for some c, then the gain $\gamma_i = C_{i,\delta_i}$. The value of C_{i,δ_i} depends on both the condition c_i and the current allocation δ_i, but is fixed for some range of values for δ_i. Thus we again have a linear program, for values of δ_i for which C_{i,δ_i} does not change. This requires only that we add an additional set of inequalities restricting each δ_i in order to force a new call to **allocate** every time C_{i,δ_i} changes for some i.

There are almost certainly other classes of anytime decision procedures for which the procedure **allocate** can be implemented efficiently. These three examples cover a

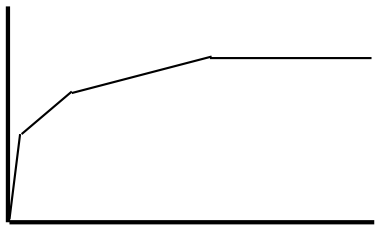

Figure 3.11: General piecewise linear performance profiles

broad range of possible performance profiles (or approximations thereto), and were not difficult to define or analyze. This provides some support for our optimistic estimate of the work required to frame a given time-dependent planning problem so as to be able to construct deliberation schedules efficiently.

3.6 Dynamic Problems

The deliberation schedules generated by the procedures **DS-1** and **DS-2** are optimal for a decision model that does not include expectations on the future occurrence of events. We can define deliberation schedulers using these procedures that generate new schedules when observations occur, and otherwise following the allocations given by the last deliberation schedule constructed. How does the expected utility of the resulting allocations compare to the allocations that would be generated by an optimal deliberation scheduler that had more information? We call the decision model that does not include any expectations on events the "static case." The situation actually faced by the system is one in which new observations and events may occur at any time.

In this section, we demonstrate empirically that applying the static schedulers to a dynamic environment results in deliberation allocations that on average are very close to optimal for a decision model that includes expectations on the occurrence of events. We do this by comparing the performance of **DS-2** to a deliberation scheduler that has perfect information (i.e., knows every condition that will happen, whether or not the corresponding observation has occurred) over randomly generated histories of observations and conditions. The scheduler is not allowed to make allocations for a condition before the corresponding observation appears. This perfect-information scheduler is actually "super-optimal" in the sense that there are cases where its expected performance is better than that of an optimal deliberation-scheduling policy that makes use of expectations on the future occurrence of events (the example in Section 3.3.3, e.g.). The same example demonstrates that there are cases for which the static schedulers do not do as well on average as an optimal deliberation-scheduling policy making use of

expectations on the occurrence of events.

We present the results of three sets of simulations in which several parameters of the problem were varied, including:

- Performance profile slope. The simulations we report on all use piecewise linear performance profiles. Increasing the range of slopes that the profiles for anytime decision procedures may take on should increase the advantage of doing effective deliberation scheduling, because there is more to gain from correctly trading off time allocated to different anytime decision procedures.

- Maximum useful allocations. Changing the maximum allocations changes the amount of contention between deliberation procedures. The greater the maximum allocations, the larger the average number of procedures that might be scheduled for any interval of time, and the more tradeoffs must be made. Increasing the range of possible maxima should have an effect similar to increasing the range of slopes.

- Prediction delay (the time between the occurrence of an observation and the occurrence of the corresponding condition). This varies in both expected value and the width of the range of possible values. Changing the expected value changes the "prediction horizon:" the longer the prediction delay, the more conditions have been predicted when the system constructs a deliberation schedule. Varying the width of the range of prediction delays changes the extent to which conditions can be added in the middle of an existing deliberation schedule. If the range is zero, newly predicted conditions always occur after conditions for which the corresponding observations have already occurred.

For each set of simulations, we compare the performance of three algorithms:

- The perfect-information algorithm described above, also called the "gold standard."

- Local **DS-2**. This is the static scheduler that generates a deliberation schedule every time a new observation occurs.

- An average-case algorithm. This procedure does not keep track of a schedule. Instead, when an observation occurs, it allocates deliberation time for the corresponding condition c according to

$$\delta = \min(\mathsf{occ}(c) - t, \frac{\gamma_c}{\bar{\gamma}}\bar{t})$$

  t is the later of the current time $\hat{t}$ and the earliest time that has not yet been allocated. γ_c is the gain for the anytime decision procedure for deliberating about

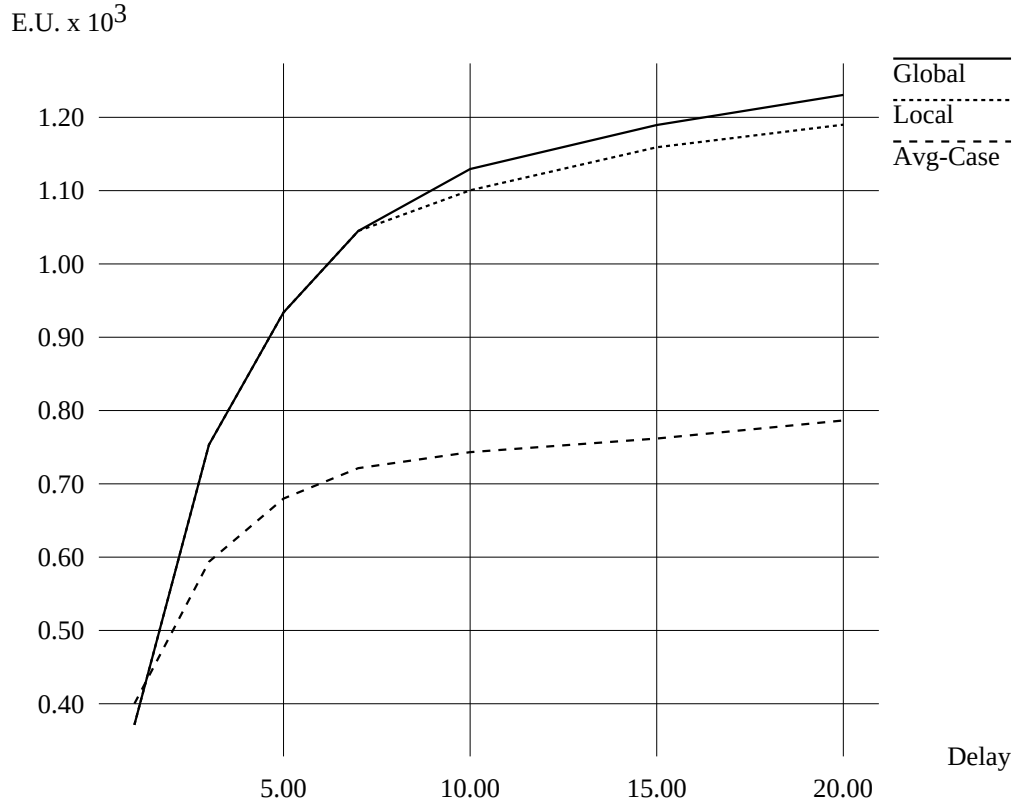

Figure 3.12: Varying prediction delay with a range on the gain

$c, \bar{\gamma}$ is the expected gain over all possible condition types, and $\bar{t}$ is the mean time between the occurrence of conditions. Time allocated to a particular event is never reallocated.

Each set of simulations involved repeatedly generating histories of observations and events, using a fixed probability that an observation would occur for any time unit (0.01), out to a fixed endpoint (30000 time units). The three algorithms were then simulated on the resulting schedules and the expected utilities of the resulting allocations were computed and averaged across trials.

The results of the first set are graphed in Figure 3.12. The independent variable is the ratio $\overline{d}$ of the prediction delay to the expected time between observations (100 time units).[11] For these trials, the slope of the performance profile for each condition was chosen at random from the values $0.0025, 0.005, 0.01, 0.02,$ and 0.05. The maximum

[11] Varying this ratio affects the average number of conditions the local procedure knows about when it constructs a deliberation schedule.

useful deliberation time was fixed at 800 time units. As $\overline{d}$ decreases, all three algorithms do worse. This is not surprising: the only possible time for allocation is between observation and condition. For small values of $\overline{d}$, there is usually at most a single predicted event, and so no tradeoffs can be made to increase expected utility. There is also simply less time: in any part of a history where conditions occur at less than the expected rate, $\overline{d} = 1$ means that there is time that cannot be used, time that would be available (and would be allocated) for a larger value of $\overline{d}$.

The most interesting feature of these experiments is the relative performance of the local algorithm and the gold standard. Not only does the local algorithm do very well, it does better (relative to the gold standard) for smaller values of $\overline{d}$, which is somewhat counterintuitive. One might expect that increasing the prediction delay, and thus the piece of the schedule that the system knows about when it generates a deliberation schedule, would improve the relative performance of the local algorithm. What happens in practice is that this effect is dominated by the constriction in the possible tradeoffs as $\overline{d}$ diminishes. As the prediction delay gets smaller, the number of conditions for which deliberation time can be traded off gets smaller as well, decreasing the opportunities for the gold standard to exploit its additional information. The fact that the gold standard has no advantage until $\overline{d}$ is about 8 and a minimal advantage all the way out to $\overline{d} = 20$ indicates that the efficient local scheduling algorithm produces schedules that are very close to optimal for a wide range of time-dependent planning problems.

The average-case algorithm improves as $\overline{d}$ increases and then flattens out. For very small values of $\overline{d}$, there is less time available (as discussed above). As $\overline{d}$ continues to increase, the average-case algorithm is not making tradeoffs among conditions, and thus gets no benefit from either the additional information available or the greater opportunity to swap allocations among conditions.

The results of the second set of simulations are graphed in Figure 3.13. Once again, we vary $\overline{d}$ along the x axis. The difference between the first and second set of simulations is that in the second set, the maximum allocation can also vary. The maximum useful deliberation time is chosen from a uniform distribution over the values 100, 200, 400, 800, 1600. The performance profiles for conditions are drawn from the cross-product of the possible slopes and maximum allocations. Thus a total of 25 distinct types of conditions can occur. As can be seen from the graph, allowing the maximum allocation to vary increases the gold standard's advantage slightly, especially for intermediate values of $\overline{d}$ (between 5 and 8). The reason for this appears to be that the range in maxima gives a greater advantage to trading off deliberation time, and over longer intervals. The global algorithm exploits this opportunity more effectively.

Finally, in Figure 3.14, we allow the prediction delay to vary: what on the preceding

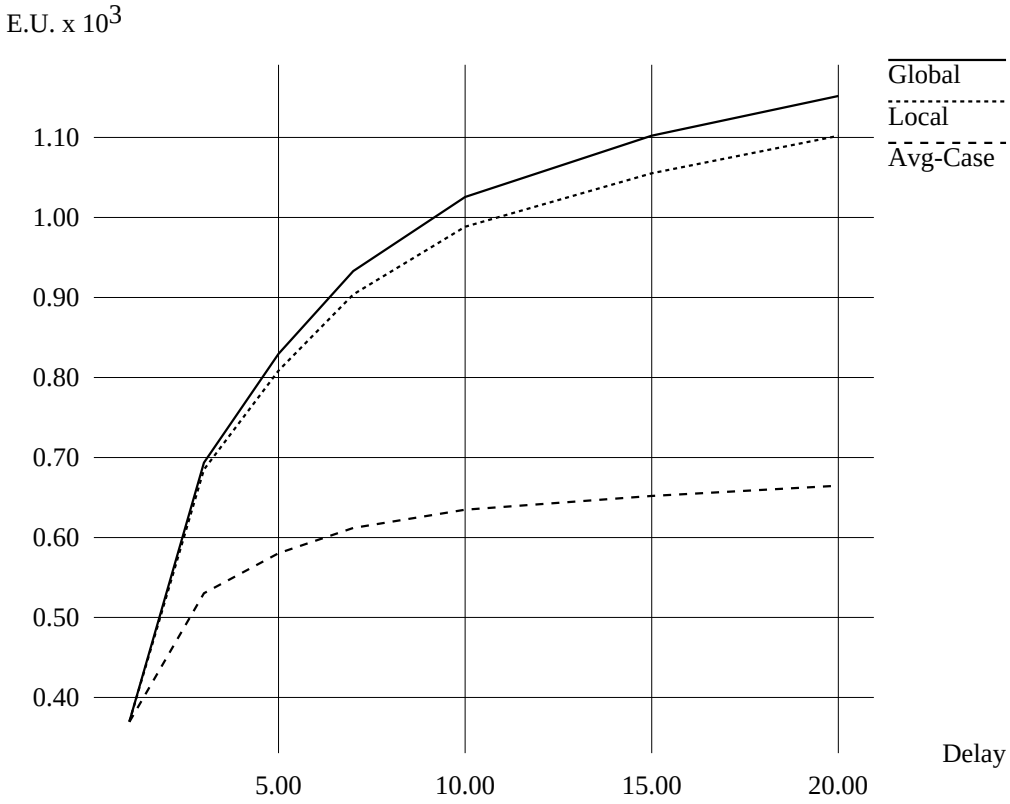

Figure 3.13: Adding a range on the maximum

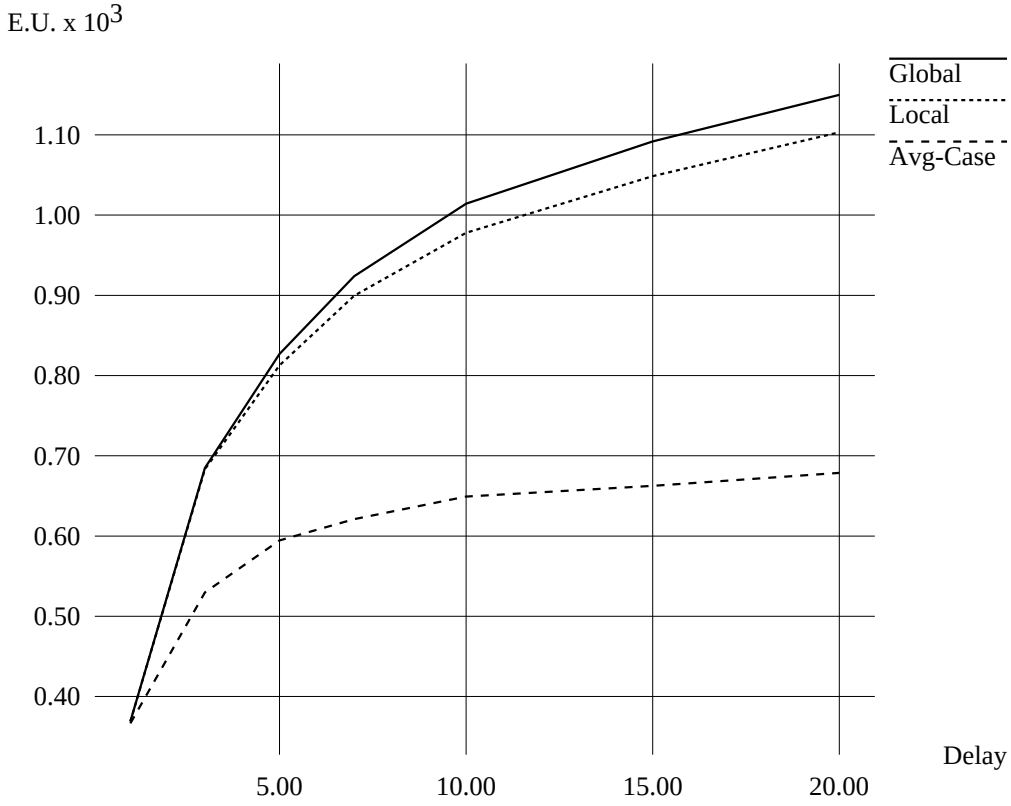

Figure 3.14: Adding a range on delay

graphs was a fixed delay is now the midpoint of a range of delays. For a given delay d, the range is from $0.5d + 50$ to $1.5d - 50$. This changes the dynamic behavior of the problem in that new events can be added in the middle of the schedule. The purpose of this change is to determine how sensitive the local algorithm is to having its schedule upset, other than at the very end. As it turns out, there is no appreciable change in the performance of the local algorithm relative to the gold standard. This seems to indicate that the degree to which the local algorithm's allocations are suboptimal is much more sensitive to the mean number of conditions it knows about than to the frequency with which a new event is added in the middle of an existing schedule. This suggests that deliberation schedules are being substantially revised on the occurrence of new observations, and that it is rare for new information to render an allocation that has already been executed suboptimal, even when that information concerns events in the middle of the existing schedule.

The results of these experiments indicate that, for a wide range of time-dependent

planning problems, a local version of **DS-1** has expected performance close to the theoretical maximum, given the available information. This is true even (in fact especially) when the horizon over which the algorithm can "see" is a small multiple of the expected time between events. The average-case algorithm, which does not look at the schedule of predicted events at all, does relatively well only for very simple classes of problems and very small prediction delays. The basic intuition explaining the near-optimal performance of the local algorithm is that new information rarely changes the optimal allocation for the part of the deliberation schedule currently being executed.

3.7 Summary and Conclusions

This chapter presents a theoretical and experimental investigation of a class of time-dependent problems involving absolute deadlines and independent responses to individual requests. For the static case, in which the decision model does not include expectations on the future occurrence of events, we provide an optimal deliberation-scheduling policy, implemented by a deliberation scheduler that constructs an explicit deliberation schedule and then makes allocations according to that schedule. We show that the deliberation scheduler used in the static case can be applied to the dynamic problem in which new observations occur as time passes with only a small loss in expected utility compared to an optimal "perfect-information" deliberation scheduler that knows the future occurrence of events. We also show that providing an optimal deliberation-scheduling policy for a decision model that includes expectations on the future occurrence of events may be difficult, in that the number of states for which a policy must be determined may be very large.

The methodology developed in this chapter is the one we apply to time-dependent problems in general. A problem is specified in terms of domain characteristics (what kinds of events happen, what kinds of actions the system can take, and a utility function on the resulting sequence of events). We then determine the anytime decision procedures to be used to do the required computation. We gather expectations on the behavior of those anytime decision procedures and how that behavior affects the expected utility of the resulting system behavior. Deliberation scheduling is a stochastic optimization problem. We use expectations on the future course of events and the answers returned by the anytime decision procedures, and seek to maximize the expected utility of the resulting system behavior. A solution to a time-dependent problem is a deliberation scheduler that provides a (possibly approximate) solution to this optimization problem at run time.

3.7.1 Extensions

The class of time-dependent planning problems analyzed in this chapter could be extended in a number of ways. In particular, one could relax any of the following assumptions used in this chapter:

1. Response time is zero (actions take no time).
2. Prediction is perfect and cheap (given an observation, we know with certainty what type of event will happen and when).
3. Observations provide evidence for the occurrence of exactly one condition.
4. Only one anytime decision procedure can be used to generate a response for any given condition (i.e., there is a function mapping from condition types to decision procedures).
5. Running any anytime decision procedure affects the response to exactly one condition. This rules out procedures for things like probabilistic projection or sensor analysis with more than one object in the sensor field.
6. There are no changes to the environment that affect the utility of the system's responses, aside from the occurrence of the conditions themselves.

In general, which of these assumptions may be relaxed, and how, is entirely determined by the characteristics of a given domain. In the rest of this section, we discuss the possible consequences of relaxing each of these assumptions.

First, consider a case in which response time is nonzero. Let $\varphi_c(\delta) \in \mathcal{A}$ be the robot's current best guess for how to react to a condition c, given δ spent on deliberation. Suppose that we have an accurate estimate of the response time $\mathsf{delay}(\varphi_c(\delta))$, and that $\mathsf{delay}(\varphi_c(\delta))$ affects the utility of the system's performance only if the response completes after the condition occurs.[12] In this case, the algorithm **DS-1** from Section 3.4 will generate an optimal schedule, given the current set of deadlines.[13] Note that we have added a dynamic element to the problem, similar to the effect of new observations occurring: if $\mathsf{delay}(\varphi_c(\delta))$ changes significantly during deliberation, then the deadline for responding to c changes, requiring the deliberation scheduler to determine a new set of allocations. As in the dynamic case, this may result in suboptimal allocations. Problems in which the time required for a given response ($\mathsf{delay}(\varphi_c(\delta))$) is

[12] Suppose that you are to be paid according to how neatly you stack a pile of bricks. You get more money for doing a better job but no money at all if you do not finish by a certain time.

[13] For a particular condition c, the deadline for responding to c can be obtained by subtracting $\mathsf{delay}(\varphi_c(\delta))$ from $\mathsf{occ}(c)$.

not precisely known can also be solved easily. The key insight is that we are maximizing the expected value of a sum of independent random variables, and not considering any notion of minimizing risk or error. Thus an allocation that maximizes the product of the expected utility of the response to an event and the probability that the response completes before the event is (locally) optimal.

Relaxing assumption 2 produces a situation in which the system has only uncertain information on the types and times of occurrence of conditions. One consequence of this more general formulation is that projection may be very expensive. General probabilistic reasoning is known to be intractable [Cooper, 1987]. A small but growing body of work on probabilistic temporal reasoning [Berzuini *et al.*, 1989, Dean and Kanazawa, 1988, Hanks, 1990, Haddawy, 1990] holds out some hope, however, of doing restricted forms of probabilistic projection efficiently. "Efficiently" should not be taken to mean "with negligible cost," however. One possibility would be to define an anytime decision procedure for doing probabilistic prediction and then to allocate time to it. Unless this procedure could be parameterized and time allocated to refining the prediction of a single specified event, it would violate assumption 5.

The difficulties resulting from relaxing assumption 3 depend on other features of the problem. If prediction is still deterministic but one observation can predict more than one event (a single request for multiple responses, each with its own deadline), then there is no difficulty. Where problems may arise is in doing probabilistic projection. An observation that provides evidence for more than one condition, or a condition for which more than one observation supplies evidence, means additional dependencies that must be taken into account.

Relaxing assumption 4 results in a combinatoric problem. Suppose there are several different anytime algorithms that might be applied to any condition. The problem then becomes: what choice of algorithm for each event results in the maximum expected utility? Whether the additional complexity introduced is significant, and if so whether it can be worked around by providing an approximate solution, will depend on the characteristics of a given domain.

Assumption 5 makes calculating the expected utility of the system's behavior easy: we simply sum the expected utilities of the responses resulting from a given set of deliberation allocations, which are conveniently available in the performance profiles. How much more difficult this calculation becomes will depend almost entirely on specific problem characteristics.

Relaxing assumption 6 admits the possibility of changes in the environment affecting the system's behavior other than the occurrence of new observations or conditions. Suppose that the system actively gathers sensory information to be used in deliberation, and that the accuracy of the system's information decreases with the length of time

since it was gathered. Consider the following scenario: An event c is predicted to occur at time t. At some point, commit(c), between t and the predicted occurrence of c, a set of sensor readings, data(c), is collected. In the time between commit(c) and occ(c), the robot must determine a response to c based upon extrapolations from data(c). The accuracy of the extrapolations performed during deliberation depends upon the distance between commit(c) and occ(c). If the extrapolations were perfect, then the expected utility of the computed reaction would depend solely upon deliberation time. Given that the extrapolations are not perfect, the problem of deliberation scheduling—in particular, the problem of determining commit(c)—is somewhat more complicated than the problem analyzed in this chapter. By representing utility as a function f of deliberation time and of the length of time between commit(c) and the occurrence of c, we can make use of the partial derivatives of f to implement a variant of the scheduling algorithm **DS-1**. This holds as long as f has a single local maximum. f is a function of both δ and occ(c) – commit(c). Allocating contiguous deliberation time is done as before. The difference is that sometimes the allocation for a single condition will not be contiguous.

3.7.2 Future Work

Even a cursory investigation of possible extensions makes it clear that we can construct a lot of different problem classes with the same fundamental structure (absolute deadlines and independent responses). Small changes in problem specification may produce large changes in how deliberation allocation is accomplished. Nor will all of these problem classes necessarily have interesting applications or illuminate interesting theoretical points. Thusl the appropriate approach to the space of problems sketched above is a descriptive one. Understanding one point (problem class) in this space and how changes may be made to arrive at other classes of problems yields a basis for providing solutions for specific applications.

However, theoretical issues not dealt in this thesis might benefit from a similar analysis (i.e., investigating the properties of a simple class of problems designed to illuminate that issue, rather than aimed at a particular application). Among these issues is the inclusion of uncertain information. Temporal reasoning with uncertain information is a hard problem. The most common responses to this fact have been either to limit the expressivity of the language [Berzuini *et al.*, 1989, Dean and Kanazawa, 1988, Haddawy, 1990], or to use the structure of the application being addressed to constrain the inference that is performed [Wellman, 1988, Hanks, 1990]. It would be interesting to extend the model analyzed in this chapter to include uncertain information, and then to see what combination of these approaches might prove to be effective. An additional issue that would almost certainly have to be dealt with is the explicit allocation of

deliberation time for projection.

3.8 Proofs

3.8.1 Proof that DS-1 generates optimal deliberation schedules

Proof: The proof proceeds in three steps:

1. Show that **DS-1** maximizes $\sum \mu_i(\delta_i)$ over all possible allocations of time (ignoring the effects of a finite interval width Δ).[14]

2. Show that scheduling the allocated deliberation time in contiguous chunks (i.e., with no switching back and forth among deliberation procedures) causes no decrease in expected utility.

3. Show that the suboptimality of $\sum \mu_i(\delta_i)$ is a function of Δ and can be made arbitrarily small by shrinking Δ.

1) We claim that the allocations $\{\delta_i, 1 \leq i \leq k\}$ made by **DS-1** maximize $\sum \mu_i(\delta_i)$. Suppose not. Then there is some set of allocations $\{\delta'_i, 1 \leq i \leq k\}$, such that $\sum \mu_i(\delta'_i) > \sum \mu_i(\delta_i)$. We provide a transformation from $\{\delta_i, 1 \leq i \leq k\}$ to $\{\delta'_i, 1 \leq i \leq k\}$ such that for each step of this transformation, the expected utility decreases or stays the same. This contradicts the assumption that $\sum \mu_i(\delta'_i) > \mu_i(\delta_i)$. We assume that $\sum \delta'_i = \sum \delta_i$, because allocating additional deliberation time never decreases expected utility.

Another way to write the difference in expected utilities between the two sets of allocations is

$$\sum(\mu_i(\delta'_i) - \mu_i(\delta_i)) > 0$$

We consider only the conditions for which $\delta_i \neq \delta'_i$. Let $C^+ = \{c_i \in \mathcal{L}_t \mid \delta'_i > \delta_i\}$, and $C^- = \{c_i \in \mathcal{L}_t \mid \delta'_i < \delta_i\}$. Let c_{min} be the element of C^- minimizing $\gamma_i(\delta_i)$. There must be a subset of C^+ of conditions $\{c_{j_1}, \ldots, c_{j_k}\}$ such that

$$\delta_{\mathsf{min}} - \delta'_{\mathsf{min}} \leq \sum_{1}^{k}(\delta'_{j_\eta} - \delta_{j_\eta})$$

and

$$\gamma_{\mathsf{min}}(\delta_{\mathsf{min}}) \geq \gamma_{j_\eta}(\delta_{j_\eta}),\ 1 \leq \eta \leq n$$

A set with these properties can always be found. $\sum \delta_i = \sum \delta'_i$, so the net change in deliberation time allocated must be zero. The allocations δ_i were generated by **DS-1**.

[14] We have already shown (Section 3.3.2) that maximizing this sum maximizes the expected value of the utility function U.

If $\gamma_{j_\eta}(\delta_{j_\eta})$ were greater than $\gamma_{\mathsf{min}}(\delta_{\mathsf{min}})$ for some η, then if the last increment allocated to δ_{min} *could* have been allocated to δ_{j_η}, it would have been, because **DS-1** allocates to the condition with the highest gain. Thus either $\gamma_{j_\eta}(\delta_{j_\eta}) < \gamma_i(\delta_i)$, or the entire allocation to δ_i is from time after the occurrence of the condition c_{j_η}, and none of it could possibly been assigned to δ_{j_η}.

Suppose we start with the allocations $\{\delta_i, 1 \leq i \leq k\}$. We subtract $\delta_{\mathsf{min}} - \delta'_{\mathsf{min}}$ from δ_{min} and distribute the same quantity among $\{\delta_{j_1}, \ldots \delta_{j_k}\}$ to make $\{\delta^*_{j_1}, \ldots \delta^*_{j_k}\}$, such that $\delta^*_{j_\eta} \leq \delta'_{j_\eta}$. Consider the effect on the expected utility of this redistribution. Because γ does not increase with increasing δ, we know that

$$\gamma_{\mathsf{min}}(\delta'_{\mathsf{min}}) \geq \gamma_{\mathsf{min}}(\delta_{\mathsf{min}})$$

and

$$\gamma_{j_\eta}(\delta^*_{j_\eta}) \leq \gamma_{j_\eta}(\delta_{j_\eta}),\ 1 \leq \eta \leq k.$$

We also know that

$$\mu_{\mathsf{min}}(\delta_{\mathsf{min}}) - \mu_{\mathsf{min}}(\delta'_{\mathsf{min}})) \geq \gamma_{\mathsf{min}}(\delta_{\mathsf{min}}) * (\delta_{\mathsf{min}} - \delta'_{\mathsf{min}})$$

and

$$\sum(\mu_{j_\eta}(\delta^*_{j_\eta}) - \mu_{j_\eta}(\delta_{j_\eta}) \leq \sum(\gamma_{j_\eta}(\delta_{j_\eta}) * (\delta^*_{j_\eta} - \delta_{j_\eta})).$$

So

$$\begin{aligned}
\mu_{\mathsf{min}}(\delta_{\mathsf{min}}) - \mu_{\mathsf{min}}(\delta'_{\mathsf{min}}) &\geq \gamma_{\mathsf{min}}(\delta_{\mathsf{min}}) * (\delta_{\mathsf{min}} - \delta'_{\mathsf{min}}) \\
&= \sum(\gamma_{\mathsf{min}}(\delta_{j_\eta}) * (\delta^*_{j_\eta} - \delta_{j_\eta})) \\
&\geq \sum(\gamma_{j_\eta}(\delta_{j_\eta}) * (\delta^*_{j_\eta} - \delta_{j_\eta})) \\
&\geq \sum(\mu_{j_\eta}(\delta^*_{j_\eta}) - \mu_{j_\eta}(\delta_{j_\eta}))
\end{aligned}$$

So the net change in expected utility for this redistribution of deliberation time is at most zero.

The result of this redistribution is a new set of allocations. If we reconstruct the sets C^+ and C^- from the new set and the δ' set, the new C^- will have one fewer element than the old one (c_{min} now has the allocation δ'_{min}). The new C^+ may contain new elements. Any elements in the new C^+ that were in the old C^+ will have the same or greater allocation of deliberation time as in the original allocation.

We can find the new c_{min} and repeat the redistribution of deliberation time. Since the elements of the new C^+ have the same or larger allocations of deliberation time as the corresponding elements of the old C^+, the new values of $\gamma_{j_\eta}(\delta_{j_\eta})$ are at most equal to the old values. So the analysis for the new redistribution proceeds exactly as for the old one and reaches the same conclusion.

After a number of iterations (redistributions) equal to the initial size of C^-, the resulting allocations will be exactly the set $\{\delta_i'\}$. Since for each step the change in expected utility is at most zero, we have contradicted the initial assumption that the set of allocations $\{\delta_i'\}$ has a higher expected utility.
2) The expected utility $\mu_j(\delta_j)$ for a particular c_j is a function only of δ_j, the amount of deliberation time allocated. When that deliberation is performed is immaterial, as long as it happens before $\mathsf{occ}(c_j)$. We need to show that

$$\mathsf{occ}(s_j) + \delta_j \leq \mathsf{occ}(c_j).$$

Since deliberation is performed on c_1, then c_2, and so on,

$$\mathsf{occ}(s_j) = \mathsf{occ}(s_{j-1}) + \delta_{j-1} = \hat{t} + \sum_{i=1}^{j-1} \delta_i.$$

So it suffices to show that

$$\sum_{i=1}^{j-1} \delta_i \leq \mathsf{occ}(c_j) - \hat{t}.$$

Each iteration of the loop (1) allocates an amount of time equal to $\mathsf{occ}(c_i) - \mathsf{occ}(c_{i-1})$ for a given value of i. In that iteration, the only conditions considered for an allocation of deliberation time are $\{c_i, \ldots, c_n\}$. Time is never allocated for a condition c_i in any iteration that allocates time corresponding to an interval after $\mathsf{occ}(c_i)$. Therefore δ_i for any $i < j$ is allocated only in the iterations allocating time before $\mathsf{occ}(c_j)$ (because $\mathsf{occ}(c_i) < \mathsf{occ}(c_j)$).

Each iteration allocates an amount of time equal to $\mathsf{occ}(c_i) - \mathsf{occ}(c_{i-1})$, for some i, except for the last, which allocates time equal to $\mathsf{occ}(c_1) - \hat{t}$. The loop (1) varies i from k down to 1. So the total amount of time allocated by the iterations from some value j down to 1 equals $\mathsf{occ}(c_j) - \hat{t}$. These are the only iterations in which time is allocated for any $c_i, i < j$. So

$$\sum_{j=1}^{i-1} \delta_i \leq \mathsf{occ}(c_i) - \hat{t}$$

and all the allocation for any condition will be completed before that condition occurs.
3) The "error" arising from the use of a Δ greater than 0 is all due to one cause: some block(s) of time of width Δ should have been allocated partially to one anytime algorithm and partially to another. Since the procedure schedules deliberation for the conditions in $\mathcal{L}_t$ in contiguous chunks, there are at most $k-1$ of these blocks. The error attributable to each block can be bounded conservatively by the function $\epsilon = \gamma_{max} * \Delta$, where γ_{max} is the maximum gain for any allocation of time to any condition in $\mathcal{L}_t$ occurring after the block in question. So the total error is bounded by $\Delta * \sum_1^{k-1} \gamma_{max}$. This being a linear function of Δ, the error can be made as small as required by making Δ small. □

3.8.2 Proof that DS-2 generates optimal schedules

Proof: The proof requires that we demonstrate two points:

1. **DS-1** maximizes $\sum \mu_i(\delta_i)$ over all possible allocations of time.
2. Scheduling the allocated deliberation time in contiguous chunks (i.e., with no preemption among processes) causes no decrease in expected utility. This argument is exactly as in the proof for **DS-1**.

For the first point, we claim that the allocations $\{\delta_i, 1 \leq i \leq k\}$ made by **DS-2** maximize $\sum \mu_i(\delta_i)$. Suppose not. Then there is some set of allocations $\{\delta'_i, 1 \leq i \leq k\}$, such that $\sum \mu_i(\delta'_i) - \sum \mu_i(\delta_i) > 0$. We provide a transformation from $\{\delta_i, 1 \leq i \leq k\}$ to $\{\delta'_i, 1 \leq i \leq k\}$, and we show that for each step of this transformation, the expected utility decreases or stays the same. This contradicts the assumption that $\sum \mu_i(\delta'_i) > \mu_i(\delta_i)$. We assume that $\sum \delta'_i = \sum \delta_i$, because allocating additional deliberation time never decreases expected utility.

Another way to write the difference in expected utilities between the two sets of allocations is

$$\sum(\mu_i(\delta'_i) - \mu_i(\delta_i)) > 0$$

We consider only the conditions for which $\delta_i \neq \delta'_i$. Let $C^+ = \{c_i \in \mathcal{L}_t \mid \delta'_i > \delta_i\}$ and $C^- = \{c_i \in \mathcal{L}_t \mid \delta'_i < \delta_i\}$. Let c_{min} be the element of C^- minimizing $\gamma_i(\delta_i)$. There must be a subset of C^+ of conditions $\{c_{j_1}, \ldots, c_{j_k}\}$ such that

$$\delta_{\mathsf{min}} - \delta'_{\mathsf{min}} \leq \sum_{1}^{k} (\delta'_{j_\eta} - \delta_{j_\eta})$$

and

$$\gamma_{\mathsf{min}}(\delta_{\mathsf{min}}) \geq \gamma_{j_\eta}(\delta_{j_\eta}), \ 1 \leq \eta \leq n$$

A set with these properties can always be found. $\sum \delta_i = \sum \delta'_i$, so the net change in deliberation time allocated must be zero. Thus any deliberation time not used for c_{min} must have been allocated to other conditions. Given the restrictions on the procedure **allocate**, if $\gamma_{j_\eta}(\delta_{j_\eta})$ were greater than $\gamma_{\mathsf{min}}(\delta_{\mathsf{min}})$, then c_{j_η} and c_{min} could never have been elements of max at the same time, and there is no possible block of time that could have been allocated to c_{min} instead of c_{j_η}.

From here, the argument proceeds as in the proof of optimality for **DS-1**.

□

Chapter 4

The Robot Courier

4.1 Introduction

Say we are in charge of designing the control program for a robot courier for a Manhatten delivery service. The function of these couriers is to pick up small parcels and deliver them to specified locations. Assuming that a courier completes all of its assigned deliveries, the main factor in evaluating its performance will be the speed with which it accomplishes this task. The faster deliveries are made, the more deliveries a courier can make, and the more income it will generate.

This *Robot Courier* problem is very different from the time-dependent planning problems discussed in Chapter 3. There, deadlines are absolute, and the time taken for a response matters only if it causes the response to be tardy. For the present problem, there are no deadlines, and any increase in response time affects the utility of the robot's behavior. Similar problems arise in several domains, for example in the design of disk schedulers for operating systems. The key features of the problem are:

- Deliberation can be done while the system is performing some action (moving from location to location, for the Robot Courier).
- Tasks can be reordered.
- The utility of the system's performance depends only on when tasks are completed.

What kinds of computation might such a robot courier perform in order to do its job better? The robot must fix upon an order in which to visit the locations on its current delivery list. The ordering chosen (a *tour*) may have considerable impact on the length of time required to visit all the locations. We call the problem of choosing this ordering *tour-improvement* planning. Once the robot has an ordering for the locations,

it may spend time determining how best to get from one of them to another. We call this *path* planning.

The benefit to be expected from either tour-improvement or path planning depends on the current situation. Finding a tour that avoids the area around the Lincoln Tunnel at rush hour is most likely a good investment. On the other hand, if the next location to visit is straight down Fifth Avenue from the robot's current position, there may be only a minor benefit to refining a path to get there. In either case, time spent in planning may hurt more than it helps if it causes the robot to delay too long. *Deliberation scheduling* for the Robot Courier thus consists of allocating time to algorithms for tour improvement and path planning, on the basis of the expected improvement in the robot's performance.

In the next section, we define the Robot Courier problem more precisely and introduce some useful notation. Section 4.3 provides a formal specification in terms of the language developed in Chapter 2. Section 4.4 presents anytime decision procedures for path planning and tour improvement and the corresponding performance profiles. In Section 4.5, we discuss how to providing an optimal deliberation-scheduling policy for these anytime decision procedures. The remainder of the chapter explores the construction of explicit deliberation schedules as a way of doing deliberation scheduling, presenting deliberation scheduling algorithms for three restricted models of the Robot Courier problem and comparing their performance.

4.2 The Robot Courier

The Robot Courier operates in a simulated environment called the *gridworld*. The gridworld consists of a rectangular subset of the integer plane $\mathcal{Z}^2$, where each point is a location that may be occupied by the robot or something else, but not both. The gridworld instances in which the robot's behavior was simulated for the experiments reported here were square regions 250 locations on a side, with a fixed probability of .25 that any given location was occupied. We refer to these instances as *standard gridworlds*. Figure 4.1 shows an example of a 40 x 40 gridworld in which each location has a .25 probability of being occupied. This is not a trivial environment for doing path-planning or dead-reckoning navigation. Adjacent occupied locations produce large and complex obstacles to negotiate, in which blind alleys are a distinct possibility.

The robot can move onto any of the four neighbors of the point it currently occupies, provided that neighbor is not already occupied. The action the robot takes is to *attempt* to move in a specified direction. If the location in question is not occupied, the robot's location changes with a time cost of $1/v$, where v is the speed at which the robot moves. If the location is occupied, the robot's location does not change, but the time

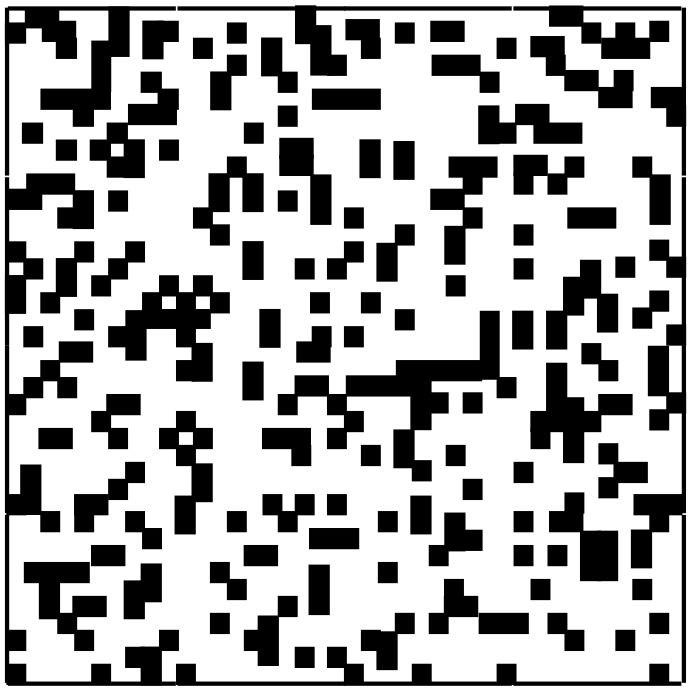

Figure 4.1: The gridworld

cost is $2/v$.[1] The robot's single sensor returns its current location. Its simple navigation routine is guaranteed eventually to reach a specified location, if it is possible to do so.[2] This routine does not require any use of the deliberation processor.

Requests specify a location that the robot is to visit. In the simplest version of the problem, the robot must visit each location requested.[3] Deliberation can be done while the robot is moving and consists of either choosing a new order in which to satisfy requests (visit the specified locations), or planning a path from one location to another, using a map stored in memory. This map can be consulted only in planning a path—the dead-reckoning navigator that requires no deliberation time cannot look at it. Reordering the locations to be visited may yield a shorter distance to travel, which may also take less time. Planning a path can be useful for two reasons. First, it may result in the robot traveling a shorter distance. Second, and as it turns out more significantly, having a planned path means not bumping into obstacles, which saves a lot of time.

[1] To see this, think of blocked intersections: one must travel almost up to the intersection to discover that it is blocked.

[2] This routine, based on Lumelsky's work [Lumelsky and Stepanov, 1987], uses a compass to head towards the goal, working around any obstacles encountered in a consistent direction (e.g., clockwise).

[3] In a more general class of problems, some value might be attached to visiting a particular location and the robot must decide whether or not it is worth it.

Before proceeding with our analysis, we introduce some terminology and notation:

- A *location* is a point in an instance of the gridworld. Each location l can be uniquely specified by specifying its x and y coordinates.
- $\mathcal{S} = \langle l_1, l_2, \ldots, l_m \rangle$ is a *tour*, an ordered sequence of locations in $\mathcal{L}$.
- $d_{l,l'}$ is the distance between locations l and l', and equals the sum of the difference in x and y coordinates between l' and l'. We use $d_{i,j}$ as a shorthand for d_{l_i,l_j} where appropriate.
- $D_{\mathcal{S}}$ is the length of the tour $\mathcal{S}$ ($D_{\mathcal{S}} = \sum_{i=1}^{m-1} d_{i,i+1}$).

4.3 Formal Statement of the Problem

The Robot Courier problem consists of

- A set of event types, $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$.
- A set of *requests*, $\mathcal{R} \subseteq \mathcal{T}$, corresponding to requests to the robot to visit a specified location.
- A set of *histories* $\mathcal{H} \subseteq 2^{\mathcal{T} \times \Re}$. Each element h of $\mathcal{H}$ is a set of ordered pairs $\{(\tau, t) | (\tau \in \mathcal{T}) \wedge (t \in \Re)\}$. We call these pairs *events*. Given an event $e = (\tau, t)$, the notation $\mathsf{type}(e)$ denotes the event type τ and $\mathsf{occ}(e)$ denotes the time of occurrence t relative to some global clock.
- A set of *actions* $\mathcal{A} \subseteq \mathcal{T}$. Four types of actions are available to the robot, corresponding to attempts to move north, south, east, or west. If the attempt succeeds, the robot's location changes appropriately. If the attempt fails (there was an obstacle), the robot's location stays the same. In either case, completion of the action is indicated by the occurrence of an event. For an event e corresponding to performance of an action, $\mathsf{comp}(e)$ denotes the event signaling completion of the action. $\mathsf{occ}(\mathsf{comp}(e))$ depends on $\mathsf{occ}(e)$ and on whether the location onto which the robot tries to move is occupied. Another attempt to move cannot be made until the last attempt has completed.
- A time t_0 such that $\forall h \in \mathcal{H}, \forall e \in h, t_0 \leq \mathsf{occ}(e)$.
- A set of *internal states* $\mathcal{S}^{int}$. Each internal state is a pair of vectors $\langle V_s, V_d \rangle$. V_s is the *sensor data*; the only sensor data available is the robot's current location. V_d is the *deliberation scratchpad* and initially includes a map of the world, including the location of obstacles.

- A set of *external states* $\mathcal{S}^{ext}$. The external state consists of the size of the world, the obstacles in it, and the robot's current location.
- A set of *initial states* $\mathcal{I}$. Each initial state i is an ordered pair $\langle s_{int}, s_{ext} \rangle$. s_{int} specifies the system's internal state at t_0. s_{ext} specifies the external state at t_0.
- A function $U : \mathcal{H} \times \mathcal{I} \rightarrow \Re$. Given an ordered list $(l_1, \ldots, l_n)$ of locations that the robot visits starting from an initial location l_0 at time t_0, there is some last action e_{last} such that the robot's location after $\mathsf{occ}(\mathsf{comp}(e_{last}))$ is l_n.

$$U(L) = -(\mathsf{occ}(\mathsf{comp}(e_{last})) - \hat{t})$$

An *instance* of the Robot Courier problem specifies the size of the world, the obstacles in it, the robot's initial location, and the times and locations for the requests that occur. A *solution* for the Robot Courier problem will consist of:

- A set of *anytime decision procedures* $\mathcal{P}$ for tour-improvement and path planning.
- An *agenda* of intended actions. The agenda is an ordered list of events corresponding to actions the system has committed to taking at the indicated time. Actions are added to the agenda only by deliberation. As time progresses ($\hat{t}$ increases), actions on the agenda are taken at the appropriate time.
- A *deliberation-scheduling policy.* This policy is implemented by a deliberation scheduler that makes allocations to the elements of $\mathcal{P}$.

4.4 Path Planning and Tour Improvement

The only forms of deliberation possible for the robot courier are to plan a path from one location to another or to change the order in which it will visit the locations requested. In order to construct a useful deliberation-scheduling policy, we need information about the procedures used to do this planning, in particular the effect of running those procedures on the robot's performance. In this section, we present anytime decision procedures for path planning and tour improvement. We use statistics on the robot's performance to construct performance profiles for these anytime decision procedures. The resulting profiles graph expectations on the effect on the robot's behavior of the time allocated to these procedures.

4.4.1 Path planning

A path from one location to another is a sequence of adjacent locations, starting at the initial location and working towards the destination. Operationally, the robot interprets

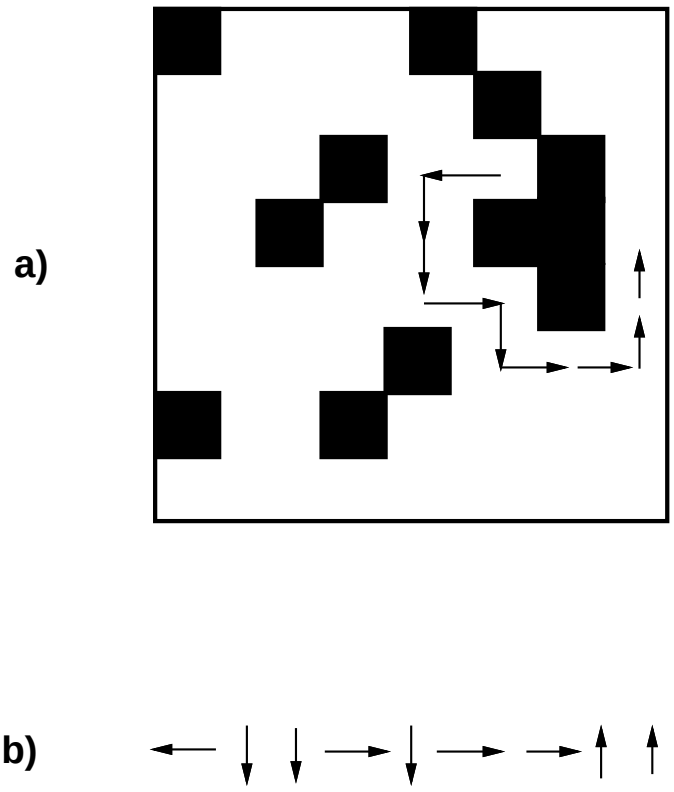

Figure 4.2: A simple gridworld path and its interpretation

a path as a sequence of actions to take, each corresponding to the direction from one location to the next. Figure 4.2a shows an example of a path from one location to another; Figure 4.2b shows the same path as interpreted by the robot. We assume that the robot can plan a path between two locations only before it starts moving between them. This is a stronger assumption than we might like, but it can be relaxed fairly easily at the cost of complicating the deliberation scheduler (by breaking the area to be traversed into planned and unplanned segments).

The procedure used to construct these paths performs a heuristic search, similar to an algorithm described in [Korf, 1987]. The algorithm we use is basically A* [Nilsson, 1980]. Nodes in the search tree correspond to locations, paths to branches from the root to a leaf. We use the remaining distance to the destination as a heuristic evaluation function. A *partial path* is a path from the starting location to some location intermediate between the starting location and the destination. A *complete path* is a path ending at the destination. The planning procedure terminates when it obtains a complete path. No attempt is made to find an optimal path. The procedure for performing a single iteration (searching an additional location) is given in Figure 4.3.

```
Procedure Plan_Path( open_list, dest):

L1:    node ← first( open_list)
       new_node ← Get_Next_Child( node)
       if new_node = FALSE
         then
             remove node from open_list
             if open_list is empty
                then
                    return FAILED
                else
                     goto L1
         else if new_node = dest
             then
                 return Make_Path( new_node)          ;All done.
             else
                 parent( new_node) ← node
                 add new_node to open_list
                 sort open_list by distance to dest
                 return open_list
```

Figure 4.3: Anytime decision procedure for path planning

The list open_list initially includes only the starting location, and is sorted by increasing distance to dest. The procedure Get_Next_Child(node) consults the map to find the location adjacent to node with the smallest distance between node and dest that is not occupied by an obstacle or already included in the search tree. If no such child can be found, then node is removed from the open list, because there is no path from node to dest that does not include a path through some other location in the search tree. Repeated calls to Plan_Path result in the construction of a path from the starting location to the destination if such a path exists. We assume that the robot never is requested to visit an inaccessible location, so a path can always be found. The procedure Make_Path takes a node as argument and constructs a path from the root (starting location) to the location corresponding to that node. The best partial path at any point can be obtained by calling Make_Path on the current head of the open list.

In order to construct expectations on the effect on the robot's behavior of time allocated to the path-planning procedure, we gathered statistics in a series of repeated trials, each in a standard gridworld (250 locations square, in which the probability of a location being blocked is 0.25). For each trial, starting and ending locations were picked at random, subject to the restrictions that the locations be distinct, that it be possible to get from one location to the other, and that neither location contain an obstacle. To get from the start to destination, the robot followed a partial path resulting from planning then moved from the end point of that path towards the destination using dead reckoning, bouncing off obstacles and finding its way around them. For each trial, we recorded the following statistics:

- The total number of planning steps (calls to Plan_Path) required to construct a complete path.
- The time the robot took to travel from start to destination, for a given number of planning steps. The robot can travel a planned path at a constant speed v. The total time required is the sum of the path length divided by v and the time required for the robot to go the rest of the way to the destination without a path.

To a very good approximation, the number of steps required to plan a complete path is linear in the distance from start to destination. The least-squares fit is given by the equation

$$y = 1.33d$$

where y is the estimated number of steps and d is the distance. The correlation coefficient for the data we gathered is 0.962, indicating that a linear approximation provides a good fit. The graph in Figure 4.4 shows the distribution of the observed data relative to this estimate.

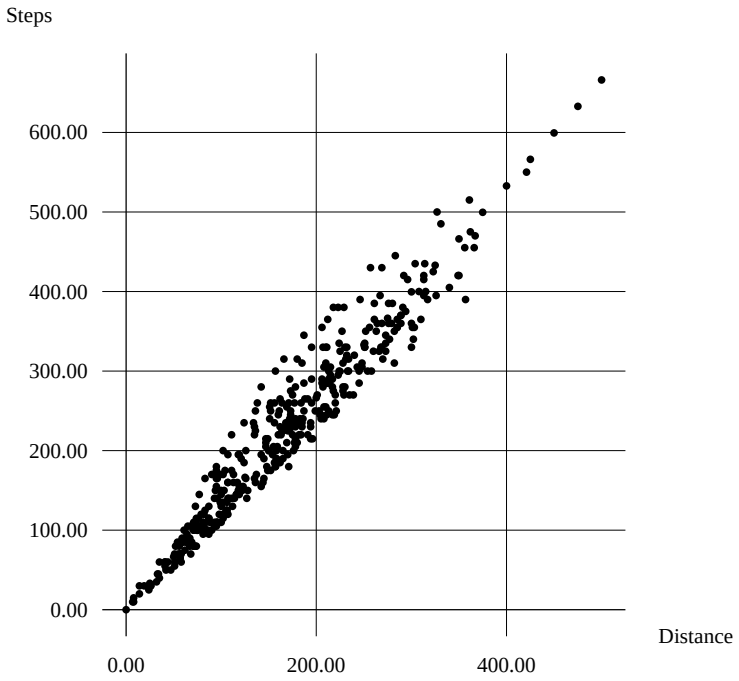

Figure 4.4: Estimating the number of steps to construct a complete path

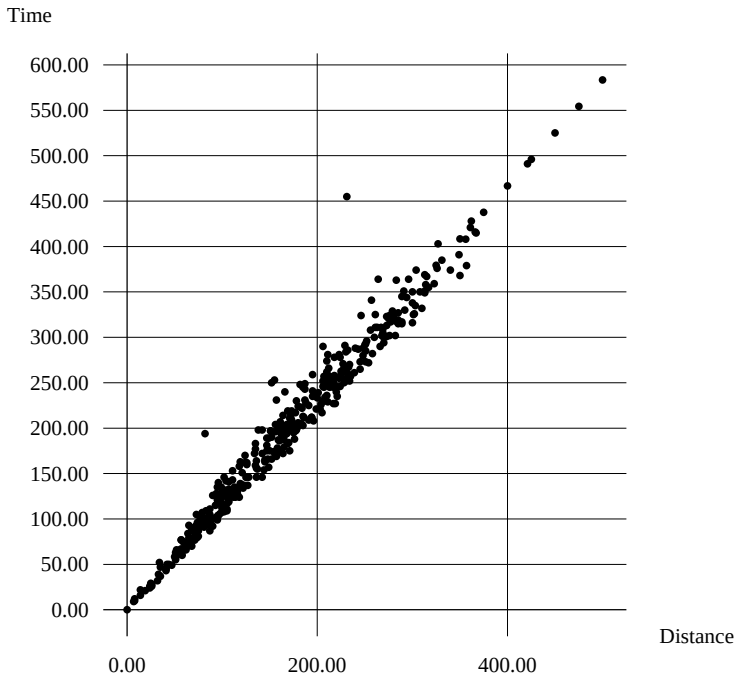

Figure 4.5: Estimating the time required given a complete path

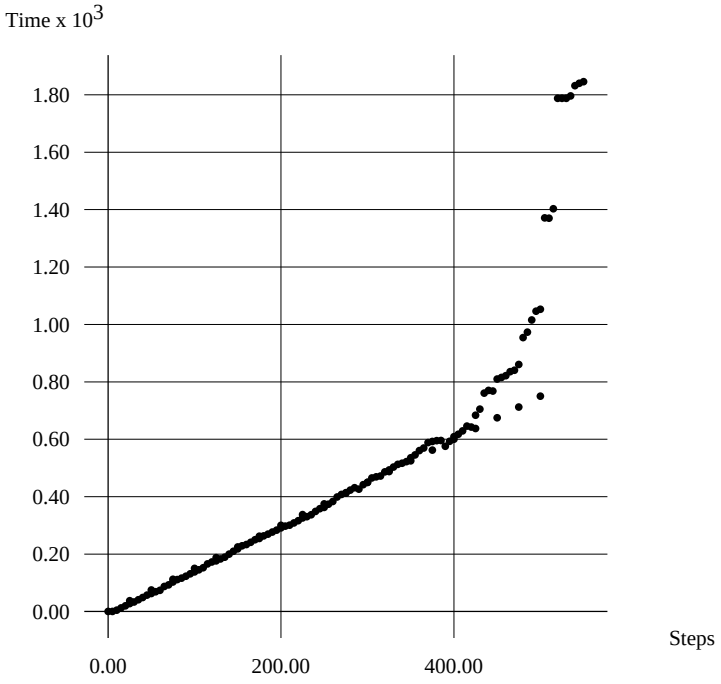

Figure 4.6: Estimating the time saved as a function of steps taken

The travel time required for a complete path as a function of distance can also be approximated quite well by a linear model. The function

$$y = 1.17d/v$$

estimates travel time from distance with a correlation of 0.98. Figure 4.5 graphs the observed data and the estimate for $v = 1$, and shows why we have not been overly concerned with providing an optimal path: there is not much room for improvement. Even an optimal path will have to deal with obstacles, and thus will usually be longer than the distance between the two points.

The expected travel time saved as a function of planning steps is linear to a reasonable approximation. The least-squares fit for time saved over planning steps taken is

$$t = \frac{1.50}{v}x$$

where t is the time saved, v is velocity, and x is the number of steps. The linear correlation for the data in this case is 0.744, which is good but not great. A comparison of this estimate to the experimental mean is given in Figure 4.6.

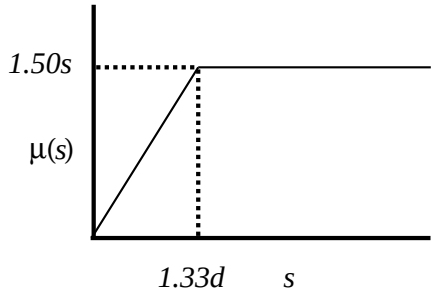

Figure 4.7: The expected savings in travel time for path planning

Several factors may influence the fit for this estimate. First, time saved is dependent on how long the robot takes without a path, which turns out to be highly variable. Let t_0 be the actual travel time required with no path, t_* be the actual time required with a complete path, and t be the actual travel time for the given number of planning steps. Taking

$$y = \frac{t_* - t}{t_* - t_0}, \quad x = \text{number of steps}$$

provides a correlation of 0.857. However, both t_0 and t_* are unknown (though we have a good estimate for t_0). This fsuggests that a limit to how good the estimate can be is imposed by the limits on the information available.

In addition, the observed mean has a highly nonlinear tail. The fact that this tail becomes significantly nonlinear only for numbers of steps between 400 and 500 suggests that some form of pathology is involved. The maximum possible distance that can separate two locations in the gridworld is 500. Few randomly chosen locations will be nearly that far apart. Thus paths requiring that many planning steps arise either from points close to the boundary (which creates difficulties for the dead reckoner) or from intervening obstacles that require an extremely convoluted path.

With these estimates, we can construct a performance profile that provides an estimate of the expected savings in travel time for planning steps taken. The expected savings in travel time is linear in the number of planning steps, up to a maximum number of steps dependent on the distance between the two locations. Let $\mu(s)$ be the expected travel time saved for s planning steps allocated (we assume that a single planning step is a constant-time operation. $\mu(s) = (1.50/v)s$ up to $s = 1.33d$, where d is the distance between the start and destination. The resulting profile is given in Figure 4.7.

What we are really interested in, however, is an estimate of expected travel time given a particular allocation of deliberation time. We use the following notation in referring to performance profiles for path planning:

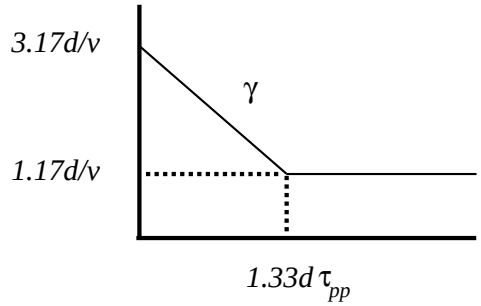

Figure 4.8: Expected travel time as a function of deliberation time

- τ_{pp} is the length of time needed for one planning step.
- δ is an amount of time allocated to planning a path between two locations.
- δ^* is the maximum useful time to spend in planning. For a pair of locations l_1 and l_2, $\delta^*_{1,2} = 1.33\tau_{pp}d_{1,2}$.
- $T(\delta)$ is the expected travel time between two locations, given δ spent planning a path. For a given pair of locations l_1 and l_2:

$$T(\delta_{1,2}) = \begin{cases} (3.17d_{1,2} - 1.50\delta/\tau_{pp})\frac{1}{v} & \delta_{1,2} < \delta^*_{1,2} \\ 1.17\frac{d_{1,2}}{v} & \delta_{1,2} \geq \delta^*_{1,2} \end{cases}$$

$$T_{1,2} = T_{1,2}(0) = 1.17d/v + \gamma\delta^*_{1,2} = 3.17d/v$$

- The *gain* $\gamma = 1.50/\tau_{pp}v$ is the rate of decrease in expected travel time as additional deliberation time is allocated up to the maximum δ^*.

A graph of $T(\delta)$ is given in Figure 4.8. One notable characteristic of the profile we have constructed is that the expected change in travel time for each planning step does not depend on distance or on the number of steps already taken, provided that number is below the estimated maximum. This is an approximation, but the close fit between these estimates and the observed data suggest that it is a reasonable one.

4.4.2 Tour improvement

The other form of deliberation the robot can do is to reorder the list of locations it must visit so as to decrease the time required to visit all of them. It is certainly possible to construct two tours such that the longer one is expected to take less time to traverse. However, gathering statistics on the length of time required to visit a set of n points shows that the expected time required decreases with decreasing tour length. In fact, we can make an accurate estimate of the time required to complete a randomly

```
Procedure 2_opt((l_1, l_2, ..., l_n)):

    for i = 1 to n - 2
        for j = i + 1 to n - 1
            t_1 = (l_1, ..., l_i)
            t_2 = (l_{i+1}, ..., l_j)
            t_3 = (l_{j+1}, ..., l_n)
            t_{i,j} = Min_Tour(t1, t2, t3)
    return arg min_{i,j}  length(t_{i,j})
```

Figure 4.9: The tour-improvement procedure **2_opt**

generated tour, given only its length and number of locations (see Section 4.6.2). In this section, we present and analyze an anytime decision procedure for tour improvement. Expected tour length is the parameter of the resulting tour we seek to minimize and for which we construct a performance profile.

Edge-exchange algorithms [Lin and Kernighan, 1973] produce tours that are progressively closer to optimal by exchanging small sets of edges (typically 2 or 3) so that the length of the overall tour decreases. It has been shown empirically that for large (100-city) tours, these algorithms average within about 8% of the optimal tour length when run to completion [Lawler *et al.*, 1985].[4]

The procedure 2-opt in Figure 4.9 exchanges pairs of edges, always making the exchange so as to maximize the decrease in tour length. Given an existing ordering, 2_opt tries all possible pairs of edges to remove. For each such pair, the procedure Min_Tour finds the way of reconnecting t_1, t_2, and t_3 that minimizes total tour length. l_1 is the current location, and thus Min_Tour cannot connect either t_2 or t_3 at that point. 2_opt returns the minimum-length tour over all possible pairs and reconnections. Repeated calls to 2_opt eventually result in a tour that cannot be shortened in this manner. This algorithm does not produce an optimal tour, but it comes close, and has the advantages of being cheap to run and being an anytime decision procedure. The average difference between the optimal tour for a randomly selected set of locations and the tour found using this algorithm is less than 2% for sets of between 10 and 15 locations.

To find the expected improvement in tour length as a function of the number of edge exchanges performed (calls to 2-opt), we repeatedly generated sets of from four to

[4]Note that a traveling-salesman tour requires a return to the starting location, whereas the "tour" we deal with here is an ordered list with a fixed starting point. Finding the optimal ordering is still an NP-hard problem.

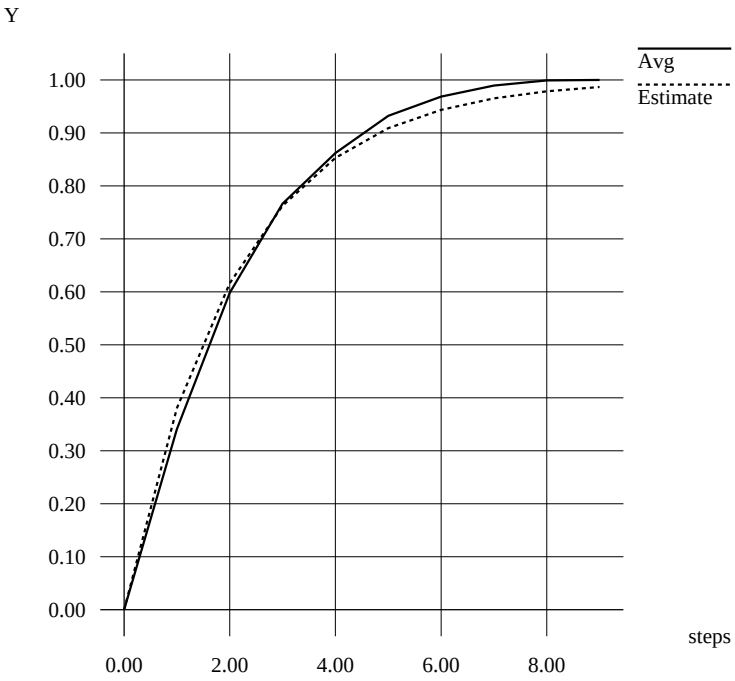

Figure 4.10: $n = 6$, $\lambda = -0.478995$

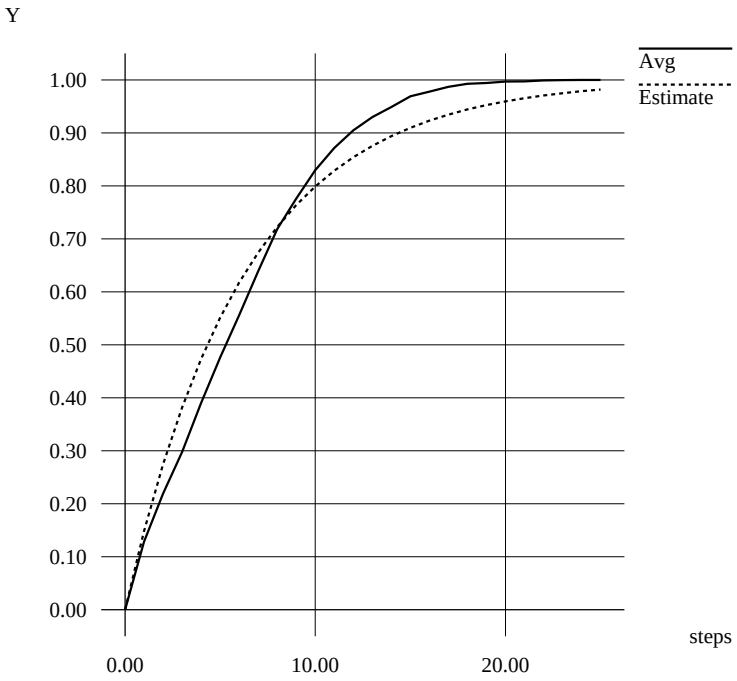

Figure 4.11: $n = 10$, $\lambda = -0.160275$

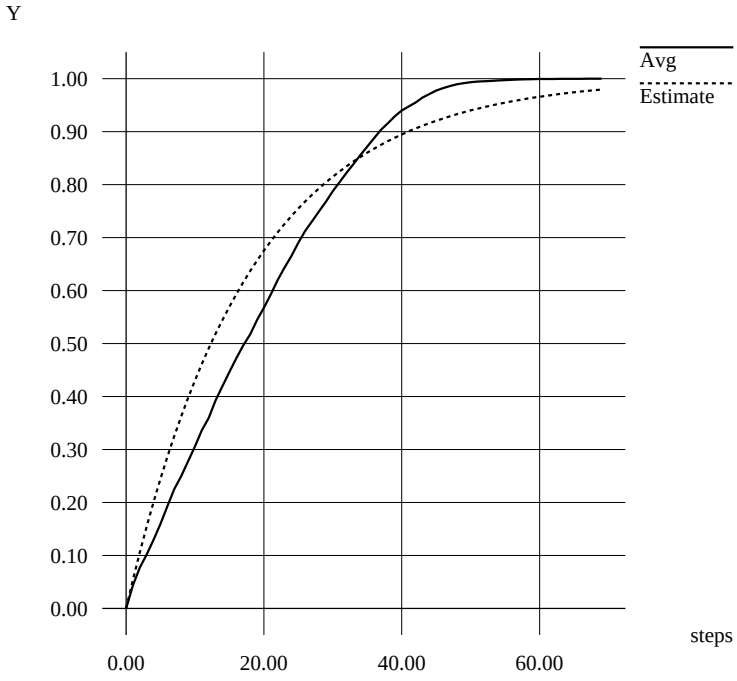

Figure 4.12: $n = 20$, $\lambda = -0.056250$

thirty random locations. For each set size, we gathered statistics on the improvement in tour length with additional exchanges. We assume pair exchange to be a constant-time operation, just as finding the next location was for path-planning. Let $L_{\mathcal{S}}$ be the length of the initial tour, L^* the length of the optimal tour (actually the length of the best tour produced by 2-opt), and $L(x)$ the length of the tour resulting from x pair exchanges. Then

$$y = E\left[\frac{L_{\mathcal{S}} - L(x)}{L_{\mathcal{S}} - L^*}\right] = f(x)$$

can be approximated by a function of the form $f(x) = 1 - e^{\lambda x}$, where λ is a function of the size of the tour. Figures 4.10 through 4.12 graph the LMS fit for functions of this form to the observed data, along with the average savings in length, for tours of three different sizes. The values of λ giving a best fit for each tour size are graphed in Figure 4.13.

It is apparent that the function f is a somewhat better fit for smaller tours. For larger tours, the expected savings in tour length as a function of the number of exchanges looks increasingly like a piecewise linear function. However, the standard error for this estimate is about 10% for any tour size up to 30.

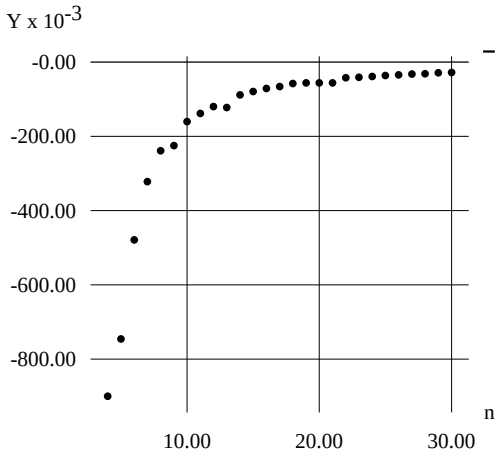

Figure 4.13: λ for different values of n

The parameter graphed on the y-axis in Figures 4.10–4.12 is $(L_{\mathcal{S}} - L(x))/(L_{\mathcal{S}} - L^*)$, the expected fraction of the possible decrease in length that has been realized in x pair exchanges. This is the statistic that we will compile and use for tour improvement. The actual improvement we can expect is dependent on λ, which we have tabulated, and also on the initial tour length $L_{\mathcal{S}}$ and the optimal tour length L^*. The value of $L_{\mathcal{S}}$ is easy to obtain. We gathered statistics to find a useful estimate for L^*. Using the same data as in estimating the expected decrease in tour length, we find that the length of the optimal tour (the one generated by 2_opt) can be approximated by the function

$$L^* = 1.42 S_T n^{0.41}$$

where S_T is a scaling factor. Let x_{max} be the greatest x coordinate for any location in the tour T and x_{min} be the least x coordinate for any location, and similarly for y_{max} and y_{min}. Then

$$S_T = \sqrt{(x_{max} - x_{min})(y_{max} - y_{min})}$$

This estimate is a fairly good one: the mean error between the estimate and the actual value of L^* over all n is less that 0.5% of the value of L^*, with a standard deviation of about 9%.

Suppose we are given a tour of size n and some initial ordering (we can pick one at random, if necessary). Using performance profiles as in Figures 4.10–4.12, the known values for the initial tour length $L_{\mathcal{S}}$ and the scaling factor S_T, and the estimate for L^*,

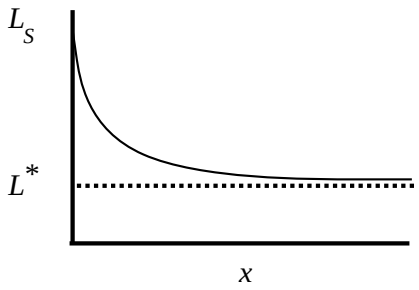

Figure 4.14: Expected tour length as a function of exchanges

we can calculate the expected length of the tour resulting from k edge exchanges:

$$\begin{aligned} E[L] &= L_{\mathcal{S}} - (1 - e^{\lambda_n k})(L_{\mathcal{S}} - L^*) \\ &= L_{\mathcal{S}} - (1 - e^{\lambda_n k})(L_{\mathcal{S}} - 1.42 S_T n^{0.41}) \end{aligned}$$

This gives us a tour-specific estimate of the tour length resulting from a proposed allocation of deliberation time. Estimated tour length as a function of edge exchanges is graphed in Figure 4.14.[5] With this result, we have the necessary expectations to do deliberation scheduling for the robot courier.

4.5 Expected Utility and Optimal Decisions

An instance of the Robot Courier problem specifies a fixed set of requests. We are interested in minimizing the time the robot spends in visiting the corresponding set of locations. From the robot's point of view, there are several sources of uncertainty:

- The types and times of arrival of requests is not known beforehand.
- The length of time necessary to travel between a given pair of locations is not known precisely, unless the robot already has a complete path planned from one to the other.
- The effect of deliberation on the robot's behavior is not known exactly. The two kinds of deliberation the robot can do are to plan a path between two locations or reorder the sequence of locations it intends to visit. The precise effect of either one of these on how long the robot takes to visit all the locations cannot be determined beforehand.

[5] To convert from exchanges to time requires multiplication by the constant τ_{ti}, the time required for a single exchange.

Each new instant of time provides information in the form of requests (or their absence), the success or failure of the robot's attempts to move in a given direction, and perhaps the progress of the anytime decision procedures being run. For each increment of time, the robot must decide whether to allocate time to reordering the sequence of locations it currently knows about or to extending a planned path between a particular pair of locations.

4.5.1 Optimal Allocation

In order to provide some intuitions regarding how new information changes the deliberation allocation problem, we work through a simple example. To simplify the analysis, we restrict the deliberation the robot can do to path planning and assume that the performance profiles we have compiled provide almost exact estimates (i.e., the error between an "estimate" and the actual resulting value is effectively 0). We also set the robot's velocity $v = 1$.

Consider the situation in Figure 4.15a. At time 0, the robot has received requests to visit locations l_1 and l_2, in that order, starting from its current location (call it l_0). Let the distance $d_{0,1}$ from l_0 to l_1 be 100, and let $d_{1,2}$ also be 100. We can derive the following parameters from the statistics gathered in the last section:

- The expected time required to plan a complete path to l_1, or from l_1 to l_2: $\delta_1^* = \delta_2^* = 133$
- The expected time required to get to l_1, or from l_1 to l_2, given a complete path: $T_1(\delta_1^*) = T_2(\delta_2^*) = 117$

The optimal allocation for path planning for these two events is shown in Figure 4.15b.[6] Allocating more to δ_1 will only increase the total time required (the total elapsed time for any greater allocation to δ_1 can be written as $\delta_1 + \delta_2 + T_2(\delta_2)$). The total expected time required $T = 122 + 133 + 117 = 372$.

Now suppose that at time $t = 245$, the robot does or does not receive a request to visit l_3 ($d_{2,3} = 100$). If the request does not occur, the situation is unchanged and the optimal allocation is as in Figure 4.15b. If the request does occur, however, the optimal allocation is as in Figure 4.15c. Note that this schedule differs from Figure 4.15b in the allocation δ_1, all of which must be executed before $t = 245$. Thus by the time the robot knows whether or not l_3 is to be visited, it may have overcommitted to deliberating about the first path and thus made a suboptimal (in hindsight) deliberation allocation.

The actual allocation that should be made depends on the probability that the request for l_3 will occur (call it p). If $p = 0$, the optimal allocation is $\delta_1 = 122$. If

[6] The optimal schedules shown for this example were generated using the algorithm in Section 4.6.1.

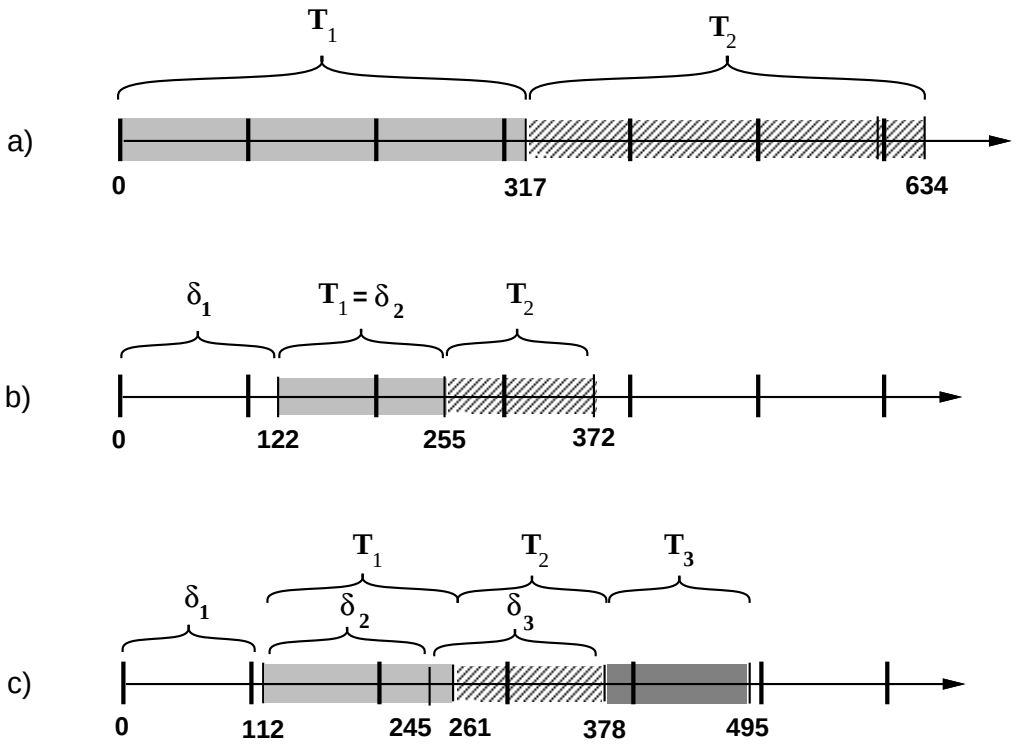

Figure 4.15: Optimal allocation for a simple deliberation problem

$p = 1$, the optimal allocation is $\delta_1 = 112$. For any other value of p, we need to find the expected *cost* of a given allocation, i.e., the extra time required compared to an optimal allocation. Given a particular allocation to δ_1, the cost for either case (i.e., l_3 occurs or not) is the difference in the total expected time T for

1. an optimal deliberation allocation, given the specified allocation to δ_1 (i.e., the best that can be done under the circumstances), and

2. an optimal deliberation allocation.

The expected cost for a given value of p and allocation δ_1 is p times the cost for the case where l_3 occurs plus $(1 - p)$ times the cost for the case where it does not.

Let R be the history including the occurrence of the request for l_3, and $\neg R$ be the history in which the request does not occur. The optimal allocation to δ_1 for $\neg R$ is 122 units of time. The total time required $T(\neg R) = \delta_1 + T_1(\delta_1) + T_2(T_1) = 122 + 133 + 117 = 372$. For R, the optimal allocation is $\delta_1 = 112$, and the total time $T(R) = \delta_1 + T_1(\delta_1) + T_2(\delta_2) + T_3(\delta_3) = 112 + 149 + 117 + 117 = 495$. For the following calculations, we assume that the new value of δ_1 will be in the range $112 - 122$, which is borne out by the form of the error calculations: changing δ_1 within that range trades off errors from R and $\neg R$, moving outside that range increases both errors. The change in $T(\neg R)$ for a different value of δ_1 is:

$$\begin{aligned}\Delta T(\neg R) &= (122 - \delta_1') + (T_1(122) - T_1(\delta_1')) \\ &= (122 - \delta_1')(1 - 1.5) \\ &= .5\delta_1' - 61\end{aligned}$$

For $\delta_1' \leq 122$, δ_2 and $T_2(\delta_2)$ will not change, because δ_2 already equals δ_2^*. The change in $T(R)$ for a different value of δ_1:

$$\begin{aligned}\Delta T(R) &= (112 - \delta_1') + (T_1(112) - T_1(\delta_1')) + (T_2(\delta_2) - T_2(\delta_2')) \\ &= (112 - \delta_1')(1 - 1.5 + .9) \\ &= -.4\delta_1' + 44.8\end{aligned}$$

In this case, δ_2 and T_2 will change, because decreasing T_1 decreases the time available. The derivation of the relation of the change in T_2 to the change in δ_1 is straightforward, starting from the equation

$$T_1 + T_2 = \delta_2 + \delta_3$$

which holds as long as $\delta_1 \geq 112$, and

$$T(\delta) = \begin{cases} 3.17d - 1.50\delta & \delta < \delta^* \\ 1.17d & \delta \geq \delta^* \end{cases}$$

derived in Section 4.4.1 (we assume here that v and $\tau_{pp} = 1$). The expected cost can now be written

$$E[C] = p(44.8 - .4\delta_1') + (1-p)(.5\delta_1' - 61)$$

Taking a partial derivative with respect to δ_1' gives

$$\frac{\partial E[C]}{\partial \delta_1'} = .5 - .9p$$

This indicates that $E[C]$ is minimized by choosing δ_1' from the range boundaries, depending on the value of p. Specifically, if $p \geq 5/9$, $\delta_1' = 112$. Otherwise, $\delta_1' = 122$.

4.5.2 Deliberation Scheduling

As demonstrated in the example above, optimal deliberation scheduling for the Robot Courier is a sequential decision problem, in which each allocation decision depends on expectations on what will happen in the future. Trying to construct an optimal deliberation-scheduling policy using dynamic programming leads to the same difficulties as in Chapter 3. Each instant provides new information (a new *state*) and requires a new allocation decision. The set of distinguishable states for which a decision must be cached is too large for generating and storing an optimal policy to be practical. Consider what features of the current state are known to the robot and might influence the optimal decision:

- The robot's current position.
- The current set of requests and their current ordering.
- How long the robot has spent planning paths among the requested locations, and which of those paths are complete.

It is possible that some of these states are equivalent from the point of view of deliberation scheduling, but determining that may not be easy.

Calculating a policy for a sequential decision model requires reasoning about expectations from a given state of the world. The uncertainty present in this problem complicates this calculation in that all of the possible next states must be examined, all of the possible states after the next one, and so on. If each state has a large number of possible next states (as is emphatically the case here—in the next instant, a request may occur for any of roughly $45,000$ locations), the size of the calculation required rapidly becomes impractical.

Taking the same approach as in Chapter 3, we deal with the difficulty of providing an optimal policy by simplifying the model for which a policy is to be provided. We simplify the problem in two ways:

- We use performance profiles that cache expectations rather than distributions on the answers returned for a given allocation of deliberation time.
- The expected utility of the robot's behavior is calculated using a decision model that takes into account only requests that have occurred, ignoring any expectations on future requests. This allows us to construct an explicit deliberation schedule once, and use it for deliberation scheduling thereafter (until a new request comes in, at which time a new schedule is generated).

In Section 4.6, we provide optimal deliberation schedulers for a series of increasing complicated deliberation-scheduling problems within this simplified model, and compare their performance on a set of static problems to determine what benefit is gained as the decision model is augmented. In Section 4.7.2, we make the same comparison, but in the dynamic case, in which new requests may arrive as the robot plans and moves about its world.

4.6 Deliberation Scheduling

In this section, we present deliberation scheduling algorithms for a simplified form of the Robot Courier problem. We restrict the problem in two ways:

- The performance profiles give expectations, not distributions, on the answers returned for a given allocation of deliberation time.
- The expected utility of the robot's behavior is calculated using a decision model that takes into account only requests that have occurred, ignoring any expectations on future requests.

The first restriction means that we provide "optimal" allocations by minimizing a function of expectations, rather than by generating the appropriate joint distribution. The second restriction allows us to construct an explicit deliberation schedule in which all requests arrive and all allocations are made before the robot starts deliberating or moving around. We investigate the problem of constructing deliberation schedules for a series of increasingly complex decision models within these restrictions:

Pr-I. Allocations can be made only for path planning.

Pr-II. Time for tour improvement can be scheduled only before any other deliberation or movement takes place.

Pr-III. The sequence of requests (the *tour*) can be divided into a prefix, for which only path-planning time is allocated, and a suffix, for which tour improvement can be

performed once, before any other deliberation or movement is done involving locations in that suffix sequence.

We provide optimal deliberation schedulers for each of these models and compare their performance on a set of static problems. In Section 4.7.2, we make the same comparison in the dynamic case, in which new requests may arrive as the robot plans and moves about its world.

4.6.1 Pr-I – Path Planning Only

Suppose the robot is given a tour $\mathcal{S} = \langle l_0, l_1, \ldots, l_n \rangle$, where l_0 is the robot's current location. Let δ_i be the amount of time allocated to deliberating about the path from l_{i-1} to l_i, for $1 \leq i \leq n$.[7] Let $\mathsf{begin}(l_i)$ and $\mathsf{end}(l_i)$ be the begin and end points, respectively, of the period of time over which the robot is traveling from l_{i-1} to l_i. This defines a time line with the following constraints:

- $\mathsf{end}(l_0) = \mathsf{end}(l_1) = 0$ — these are only relative time points
- $\forall l_i : 1 \leq i \leq n, \mathsf{begin}(l_i) = \mathsf{end}(l_{i-1})$
- $\forall l_i : 1 \leq i \leq n, \mathsf{end}(l_i) = \mathsf{begin}(l_i) + T_{i-1,i}(\delta_i)$ — δ_i initially 0

We will also use the following definitions from Section 4.4.1:

- τ_{pp} is the length of time necessary to perform one planning step.
- δ^* is the maximum useful time to spend planning. Traveling from l_{i-1} to l_i, $\delta_i^* = 1.33\tau_{pp}d_{i-1,i}$.
- The *gain* $\gamma = 1.50/\tau_{pp}v$ is the change in expected travel time as additional deliberation time is allocated, up to the maximum δ_i^*.
- $T_{i,j}(\delta)$ is the expected travel time from l_i to l_j, given δ spent planning a path:

$$T_{i,j}(\delta) = \begin{cases} 3.17d_{i,j}/v - \gamma\delta & \delta < \delta_{i,j}^* \\ 1.17d_{i,j}/v & \delta \geq \delta_{i,j}^* \end{cases}$$

$$T_{i,j} = T_{i,j}(0) = \frac{3.17d_{i,j}}{v}$$

- The expected savings in travel time, for an allocation of deliberation δ, is:

$$T_{i,j} - T_{i,j}(\delta_{i,j}) = \begin{cases} \gamma\delta_{i,j} & \delta_{i,j} < \delta_i^* \\ 2d_{i,j} & \delta_{i,j} \geq \delta_{i,j}^* \end{cases}$$

```
Procedure: Sched-1(S)
   begin
      t := end(l_n)
      for j = n to 1, step -1
          begin
              if t > begin(l_j), then
                  gap := min(γδ*_j, t - begin(l_j))
                  t = begin(l_j)
               else
                  gap := 0
              if t = begin(l_1), goto exit
(*)           δ_j := min(δ*_j, t - begin(l_1), 1/(γ+1)(t - begin(l_1) + gap))
              t := t - δ_j - max(0, γδ_j - gap)
          end
exit:: if γ > 1 and gap > 0, then
          Δ := min(δ*_j - δ_j, 1/γ * gap)
          δ_j := δ_j + Δ
      δ_i := 0, 1 ≤ i < j
      return Δ + Σ_1^n (T_{i-1,i}(δ_i))
  end
```

Figure 4.16: Constructing a deliberation schedule for path planning

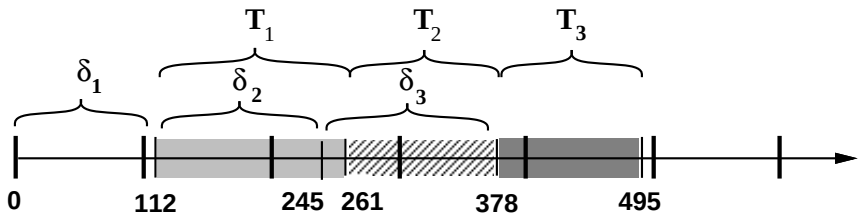

Figure 4.17: A deliberation schedule for path planning

The procedure Sched-1 in Figure 4.16 generates an optimal deliberation schedule for path planning. These schedules are optimal relative to a restricted decision model: performance profiles give expected values rather than distributions, and the possibility that new requests occur is ignored. Given this model, Sched-1 generates a deliberation schedule for $\mathcal{S}$ such that there is no other schedule for which the robot is expected to complete the tour in less time.[8] Sched-1 assigns values to all the δ_i. At run time, the robot deliberates on each path in turn. If the robot reaches the location l_{i-1} before completing the deliberation allocation δ_i, it sits and waits for the deliberation to complete before proceeding.

The procedure works backward from the last path to be traversed, ensuring that each path is allocated the maximum useful amount of time. There may be times where the robot is predicted to be traversing a path and all later paths have already been allocated as much time as will produce any improvement.[9] The length of the earliest such predicted idle time is the value of the variable gap. The algorithm is simplified considerably by the fact that the gain γ is the same for any path (i.e., as long as $\delta_i < \delta_i^*$, it doesn't matter which δ_i is allocated, for $1 < i < n$). Figure 4.17 shows the deliberation schedule constructed for the example in Section 4.5, in which $d_{0,1} = d_{1,2} = d_{2,3} = 100$. The shaded regions represent time allocated for deliberation.

A deliberation schedule for Pr-I can also be constructed by means of a linear program. See Section 4.9 for details.

Because Sched-1 works backward over the predicted schedule of deliberation and action, the resulting deliberation schedule assigns more deliberation time to later paths. For the static case, this does not matter. When we apply the algorithm in a dynamic environment, however, it probably makes more sense to allocate as much time as possible to earlier paths, since that part of the schedule is less likely to be modified as

[7] Given the tour $\mathcal{S}$, we use δ_i as a shorthand for $\delta_{i-1,i}$.

[8] Proofs that the algorithms in this section are optimal may be found in Section 4.9. The asterisk in Figure 4.16 marks a line that is important to the proof in that section, and has no other significance.

[9] Recall that the robot cannot work on a path for the pair of points between which it is currently in transit.

```
Procedure: Sched-Fix(⟨ l_0, l_1, ..., l_n⟩, Δ, {δ_i})
  begin
     i := 1
     j := i + 1
     t := 0
     while i < n − 1
         t = t + δ_i
         j := max(j, i + 1)
         while δ_i < δ_i^* ∧ j < n ∧ t < begin(l_i)
             x := min(δ_i^* − δ_i, begin(l_i) − t, δ_j)
             δ_i := δ_i + x
             δ_j := δ_j − x
             t := t + x
             j := j + 1
         i := i + 1
  end
```

Figure 4.18: Fixing up the deliberation schedule for a dynamic environment

time passes and new information becomes available. This modification to the deliberation schedules generated by Sched-1 can be made in a single sweep forward over the deliberation schedule in which time is reallocated to earlier paths to the extent possible while preserving the optimality of the resulting schedule. Procedure Sched-Fix in Figure 4.18 takes a tour and the corresponding allocations as arguments and performs this reallocation.

4.6.2 Pr-II

Suppose that we extend Pr-I by allowing the tour $\mathcal{S}$ to be reordered before any path planning is done. In other words, if the initial tour is far from optimal, we can now expend some effort improving it before doing path planning. Given an improved tour $\mathcal{S}'$, we can generate a deliberation schedule for path planning using Sched-1. So at run time, the robot will schedule deliberation on tour improvement, based on the expected effect on the time required to traverse the improved tour, and then do the scheduled deliberation. When tour improvement is completed, the robot will schedule path-planning time and then plan paths and move from location to location, just as in Pr-I.

To calculate the expected utility of a given allocation of time for tour improvement, we need to be able to estimate the time required, including path-planning time, to complete $\mathcal{S}'$. We have an estimate for the length of the tour $\mathcal{S}'$ resulting from allocating

a given amount of time to tour improvement. However, we do not know the distances between individual locations, and so cannot use Sched-1 to determine the expected time required, given an optimal schedule.

We provide an estimate for the expected travel time for $\mathcal{S}'$ given an optimal deliberation schedule for path planning by solving the problem under the assumption that all the distances between locations in $\mathcal{S}'$ are equal. Suppose we start with the tour $\mathcal{S} = \langle l_0, \ldots, l_n \rangle$ and allocate some time to tour improvement, achieving an improved tour $\mathcal{S}'$ with expected length $L_{\mathcal{S}'}$. In this case, the path-planning problem involves n paths, all of length $d = D_{\mathcal{S}'}/n$. The expected time to complete the tour given an optimal deliberation schedule can be expressed as a linear function of d (and thus of $L_{\mathcal{S}'}$). We use the same definitions as in the previous section. For $2 \leq i \leq n-1$, we drop the subscripts from t and δ, since they have the same values for any such i. δ^* and d are constant for any i, $1 \leq i \leq k$. We make use of the following estimates from section 4.4.1:

- $\gamma = 1.50/v\tau_{pp}$
- $\delta^* = 1.33d\tau_{pp} = 2.00d/\gamma v$
- $T(\delta) = 3.17d/v - \gamma\delta$
- $T = T(0) = 3.17d/v$
- $T(\delta^*) = 1.17d/v$

For $n = 1$ (i.e., there are only 2 locations in $\mathcal{S}$), there are two cases:

- If $\gamma \leq 1$, then $\delta_1 = 0$, and $T = 3.17d/v$.
- If $\gamma > 1$ then $\delta_1 = \delta^*$, and $T = \delta^* + 1.17d/v = (2.00/\gamma + 1.17)(d/v)$.

For larger tours, there are four cases, also depending on the value of γ:

- $\gamma \geq 1.71$. $\delta_1 = \delta_n = \delta = \delta^*$. Then $\mathcal{T} = \delta^* + nT(\delta^*) = (1.17n + \frac{2}{\gamma})(d/v)$.
- $\frac{(n-1)}{0.78n+0.55} \leq \gamma < 1.71$. $\delta_n = \delta = \delta^*$, $\delta_1 = (\frac{1.17n+0.73}{\gamma} - \frac{2(n-1)}{\gamma^2})(d/v)$.
  Then $\mathcal{T} = (1.17 + \frac{3.17n-1.17}{\gamma} - \frac{2(n-1)}{\gamma^2})(d/v)$.
- $\frac{1}{1.585(n-1)} \leq \gamma < \frac{(n-1)}{0.78n+0.55}$. $\delta_1 = 0, \delta_n = \delta^*$. We want δ such that $T+(n-2)T(\delta)-(n-2)\delta-\delta^* = 0$. Solving, $\delta = \frac{3.17(n-1)-\frac{2}{\gamma}}{(n-2)(1+\gamma)}(d/v)$. Then $\mathcal{T} = \frac{3.17n+1.17\gamma}{1+\gamma}(d/v)$.
- $\gamma < \frac{1}{1.585(n-1)}$. $\delta_1 = \delta = 0$, and $\delta_n = (n-1)T = (n-1)(3.17d/v)$.
  Then $\mathcal{T} = 3.17(n(1-\gamma)+\gamma)(d/v)$.

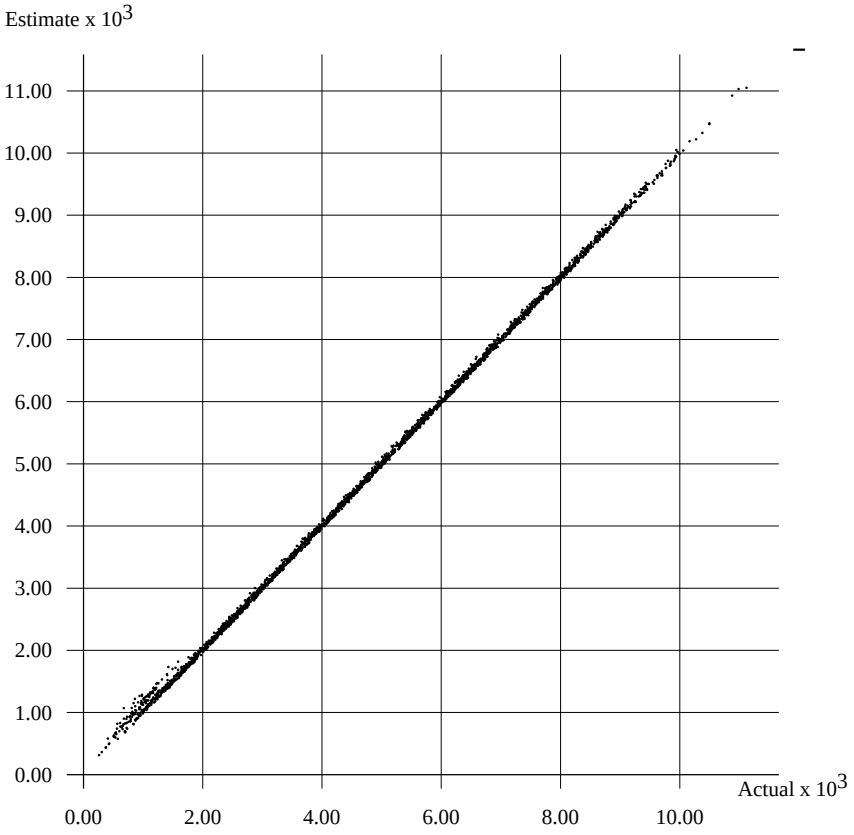

Figure 4.19: Comparing the estimate with the actual time required

The cases are distinguished by differing values of γ and n. In the first case, the expected benefit from path planning is so great that each path is allocated the maximum deliberation time. In the second case, the expected benefit is slightly less, so that setting δ_1 to δ^* might make the tour take longer. In the third case, the expected benefit of path planning is low enough that any value of $\delta_1 > 0$ has a negative utility, and only δ_n is automatically allocated the maximum amount of time. In the last case, the improvement to be expected is so small that only the last path in the tour is worth planning for, and that only while traveling among the other locations.

This estimate turns out to be an extremely good fit to the actual expected time, given a deliberation schedule generated by Sched-1. Figure 4.19 is a scatter plot of points comparing the estimated and actual times. For each point, a random tour of from 4 to 30 locations was generated. The x-coordinate is the actual time returned by Sched-1, and the y-coordinate is the time calculated using the estimate given above.

There are 2700 data points in the graph—100 for each tour size from 4 to 30. The slope of the least-squares line for this data is 1.001, and the correlation coefficient is 0.9998. These results do not show that this estimate provides a simpler way of generating near-optimal deliberation schedules for path-planning. What they do show is that an optimal schedule for a tour in which all the locations are the same distance apart has an expected travel time very close to the time for an optimal schedule for a tour of the same length but with a different distribution of inter-location distances. This makes sense, given that the gain γ is a constant for any pair of locations at any distance, up to the maximum allocation δ^*.

For a particular domain, γ and v are constants. n is fixed for a given set of locations. Let $f(d) = \alpha d$ be the appropriate estimate for expected travel time. From Section 4.4.2, we see that the estimated length of the tour resulting from spending δ' on improving the tour $\mathcal{S}$ is:

$$\begin{aligned} g(\delta') &= L_{\mathcal{S}} - (1 - e^{\lambda_{n+1}\delta'})(L_{\mathcal{S}} - L^*) \\ &= L_{\mathcal{S}} - (1 - e^{\lambda_{n+1}\delta'})(L_{\mathcal{S}} - 1.42 S_T (n+1)^{0.41}) \end{aligned}$$

where $L_{\mathcal{S}}$ = the length of the tour $\mathcal{S}$, S_T depends only on the set of locations in $\mathcal{S}$, and $(n+1)$ is the number of points in the tour. λ_{n+1} is a constant for a tour of size $n+1$. The expected time required to do tour improvement, then plan paths for and traverse the improved tour is:

$$\begin{aligned} T(\delta') &= \delta' + f(g(\delta')/n) \\ &= \delta' + \alpha(L_{\mathcal{S}} - (1 - e^{\lambda_{n+1}\delta'})(L_{\mathcal{S}} - 1.42 S_T (n+1)^{0.41}))/n \end{aligned}$$

Minimizing this value maximizes the expected utility. Taking the derivative with respect to δ' yields:

$$T' = 1 + \frac{\alpha \lambda_{n+1}}{n}(L_{\mathcal{S}} - 1.42 S_T (n+1)^{0.41}) e^{\lambda_{n+1}\delta'})$$

With $\alpha, n+1$, and $(L_{\mathcal{S}} - 1.42 S_T (n+1)^{0.41}) \geq 0$, and $\lambda_{n+1} < 0$, this function has a single root for $\delta' > 0$:

$$\delta' = \frac{1}{\lambda_{n+1}} \log \frac{-n}{\alpha \lambda_{n+1}(L_{\mathcal{S}} - 1.42 S_T (n+1)^{0.41})}$$

With the second derivative

$$T'' = \frac{\alpha \lambda_{n+1}^2}{n}(L_{\mathcal{S}} - 1.42 S_T (n+1)^{0.41}) e^{\lambda_{n+1}\delta'}) \geq 0$$

we know that this value for δ' minimizes $T(\delta')$.

The procedure Sched-2 in Figure 4.20 constructs an optimal deliberation schedule for Pr-II, for tours of three or more locations. Sched-2 determines the deliberation

Procedure: Sched-2($\mathcal{S}$)

begin

$\delta' := \frac{1}{\lambda_{n+1}} \log \frac{-n}{\alpha\lambda_{n+1}\ (L_{\mathcal{S}}\ -\ 1.42\ S_T\ (n+1)^{0.41})}$

return $\delta' + \alpha(L_{\mathcal{S}} - (1 - e^{\lambda_{n+1}\delta'})\ (L_{\mathcal{S}} - 1.42\ S_T\ (n+1)^{0.41}))/n$

end

Figure 4.20: Constructing a deliberation schedule for Pr-II

allocation for tour improvement that minimizes the expected time to traverse the tour, and returns that expected time. Once the tour-improvement algorithm has run, the robot runs Sched-1 on the resulting tour.

Figure 4.21 gives a qualitative description of the robot's operation using Sched-2 in a series of five snapshots illustrating the robot in various states of planning and deliberation scheduling. In each of the five snapshots, an asterisk indicates the time at which the snapshot is taken, t_0 to t_1 is the time spent in path planning before starting to travel to the first location in the current tour, and t_k to t_{k+1} (for $1 \leq k \leq n-1$) is the time spent traveling from the kth to the $k+1$st location. Figure 4.21-i depicts the situation in which the robot has an initial tour and λ_i is the expected time to traverse that tour. We represent the time required for constructing deliberation schedules by ϵ.

Figure 4.21-ii shows the robot's expectations after running Sched-2. The interval labeled δ is the amount of time allocated to tour improvement. The expected times spent in path planning and path traversal are represented by the interval λ_{ii}. Figure 4.21-iii shows the robot's expectations after actually performing tour improvement. At this point, the robot knows the exact order (and thus interpoint distances) for the improved tour. The interval λ_{iii} is the expected time needed to traverse the tour with no path planning (t_0 is identical to t_1).

Next, the robot calls Sched-1 to allocate time to planning each individual leg of the improved tour. Figure 4.21-iv shows the resulting deliberation schedule. The interval λ_{iv} is the expected time for carrying out both path planning and path traversal. Finally, Figure 4.21-v shows the actual schedule and elapsed time λ_v resulting when the robot traverses the tour. The actual tour may take more or less time than the robot's expectations.

4.6.3 Pr-III

Suppose now that we can interleave tour improvement and path planning, in the sense that we partition the tour into some number of sequences of locations for which we alternately do tour improvement followed by path planning, or just do path planning.

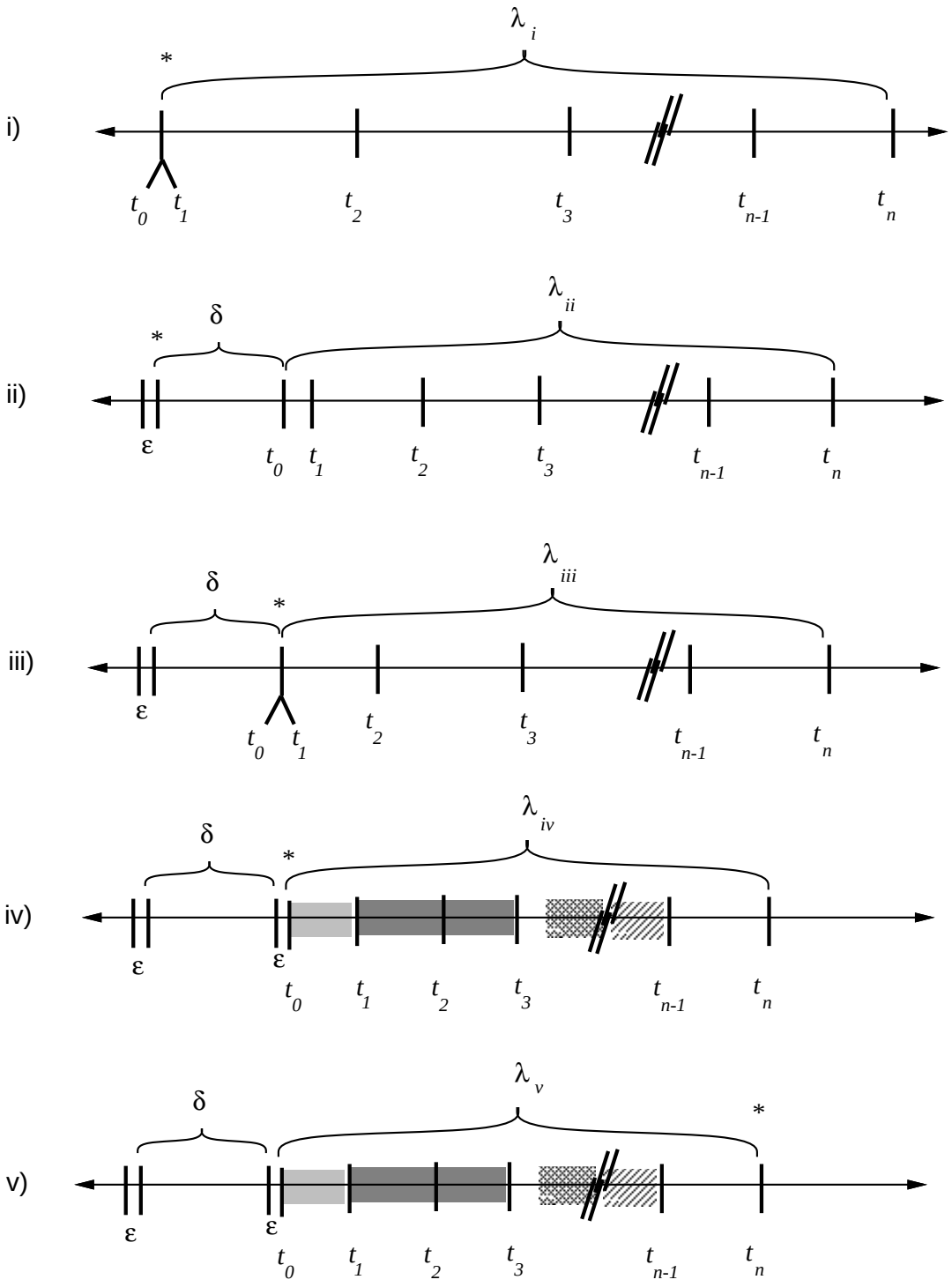

Figure 4.21: Deliberation scheduling example for Pr-II

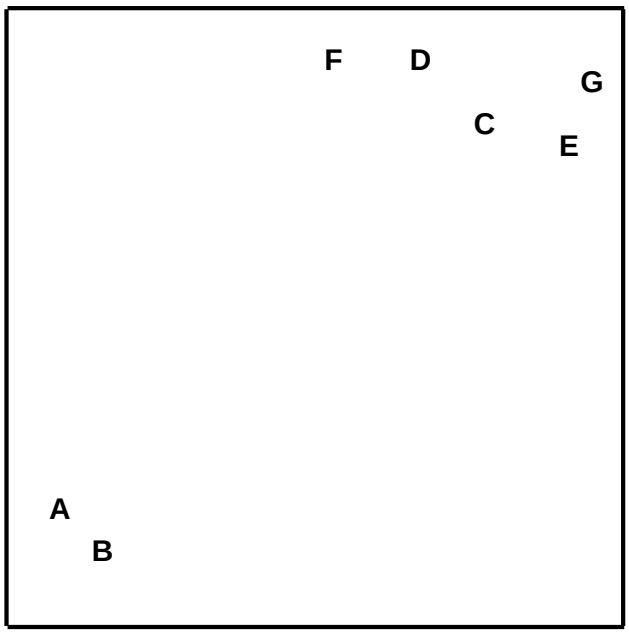

Figure 4.22: A Robot Courier tour

One immediate problem is that there is an exponential number of ways to do this partition. Clearly, trying all partitions to find the one minimizing traversal time for the tour would violate our assumption that deliberation scheduling is computationally very cheap. This kind of interleaving of tour improvement and path planning would also make it advisable to gather statistics about a different mode of tour improvement. The performance profile given in Section 4.4.2 was generated under the assumption that only the initial location of the tour was fixed. Interleaving in this way would require fixing both the initial and final locations in a sequence and reordering only the intervening locations.

Consider Figure 4.22. The robot is currently at A and intends to visit the locations B through G, in alphabetical order. The fact that the robot can deliberate while moving suggests that a good deliberation schedule might entail moving from A to B while planning a path to C, followed by tour improvement and path planning for locations C through G during the trip from B to C. We can generalize this to include an arbitrary number of locations (a prefix for the tour sequence) for which no tour improvement is to be done.

We define the scheduling problem Pr-III to consist of determining some number of locations for which we plan paths first, followed by the remaining locations, which are treated as an instance of problem Pr-II. Sched-3 in Figure 4.23 constructs an optimal

```
Procedure: Sched-3(S)
   begin
      t3 := ∞
      for i = 0 to n − 2
          begin
              s := Sched-3a(S, i)
              if t3 > s, then
                    t3 := s, pivot := i
          end
      return i*
  t1 := Sched-1(S)
      if t1 < t3
     return t1
        else
         return Sched-3a(S, pivot)
   end
```

Figure 4.23: Constructing a deliberation schedule for Pr-III

deliberation schedule for Pr-III for tours of three or more locations.[10] Sched-3 loops forward over the locations in the current tour $\mathcal{S}$. For each location l_i, Sched-3a (Figure 4.24) determines the traversal time required for an optimal deliberation schedule, assuming that l_i is the start of the part of the tour on which tour improvement will be done (the *pivot*). Having l_0 as the pivot yields a deliberation schedule equivalent to that generated by Sched-2. It is also possible, especially for smaller tours, that the optimal deliberation schedule allocates no time for tour improvement, so Sched-3 compares the expected time returned by Sched-1 to the time for each of the possible pivots. Sched-3 returns the expected time to complete the schedule, given the pivot and allocations set in Sched-3, Sched-3a, and/or Sched-1. The possible second call to Sched-3a is to regenerate the optimal deliberation schedule, which may have been changed by other calls to Sched-3a or Sched-1.

In the presentation of Sched-3a, δ' is the time spent in improving the part of $\mathcal{S}$ after the pivot, while moving among the locations before the pivot. Δ' is the time spent in tour improvement before any movement, and Δ is the time allocated to path planning on the part of the tour before the pivot, to be done before any movement. T_{pv} is the expected traversal time for the part of the tour after the pivot, given the time allocated to tour improvement. The function TI-est-dt takes as arguments a slope and the pivot, and returns the allocation such that the derivative of T_{pv} equals the slope.

[10] For two locations, no tour improvement is possible.

```
Procedure: Sched-3a(S, pv)
  begin
     δ' := end(l_pv) − begin(l_1)
     if pv = 0
        return Sched-2(S)
     t := δ'
     for j = pv to 1, step −1
        begin
           if t > begin(l_j), then
              gap := min(γδ*_j, t − begin(l_j))
              t := begin(l_j)
            else
              gap := 0
           if t = begin(l_1), goto exit
           x := (t − begin(l_1) + gap)/(1 + γ)
           y := max(0, (δ' − TI-est-dt(−γ/(γ+1), pv))/(1 + γ))
(*)        δ_j := min(δ*_j, t − begin(l_1), x, y)
           t := t − δ_j − max(0, γδ_j − gap)
           δ' := δ' − (1 + γ)δ_j
        end
exit:: if γ > 1 and gap > 0, then
        begin
           x := (max(0, δ' − TI-est-dt((1 − γ)/γ, pv)))/γ
           Δ := min(δ*_j − δ_j, gap/γ, x)
           δ_j := δ_j + Δ
           t := t − Δ
           δ' := δ' − γΔ
        end
     Δ' := max(0, TI-est-dt(−1.0, pv) − δ')
     return Δ' + Δ + Σ_1^pv T_{i−1,i}(δ_i) + T_pv(δ' + Δ')
end
```

Figure 4.24: Constructing a deliberation schedule for Pr-III

.

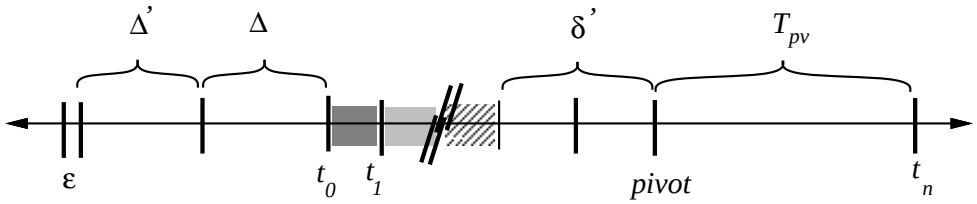

Figure 4.25: A Pr-III deliberation schedule

The procedure initially allocates all the travel time for the locations $l_0, \ldots, l_{\mathsf{pv}}$ to deliberation on an improved tour for $l_{\mathsf{pv}}, \ldots, l_n$, and allocates no time for path planning for $l_0, \ldots, l_{\mathsf{pv}}$. Using an algorithm almost identical to Sched-1, the procedure then sweeps back from $\mathsf{begin}(l_{\mathsf{pv}})$, determining for each increment of time whether spending time on tour improvement or path planning is expected to save more time in the final schedule. Finally (following the step marked exit), time is allocated before any movement is to be done. Time may be allocated to either path planning for $l_0, \ldots, l_{\mathsf{pv}}$ (Δ) or tour improvement for $l_{\mathsf{pv}}, \ldots, l_n$ (Δ'). The total expected traversal time for the deliberation schedule generated is returned.

Figure 4.25 shows what a deliberation schedule generated by Sched-3 looks like. First comes the time allocated for tour improvement and path planning without moving, then moving and further deliberation up to the location labeled "pivot." When the robot reaches that location, it will have an improved ordering for the part of the original tour following that point. At that time, it will schedule path planning time as in Sched-1 (using Sched-1, in fact). Figure 4.21 shows how the robot proceeds from here on.

4.7 Comparing the Deliberation Schedulers

Now that we have these three optimal schedulers for different decision models, an obvious question to ask is: what is the effect of the difference in models on performance? Sched-3 (for Pr-III) is more complex than Sched-2 (Pr-II), which is more complex than Sched-1 (Pr-I). In fact, there is a strict hierarchy in the sense that each scheduler makes use of the ones with smaller numbers. So the schedulers based on larger decision models should do better. But by how much?

Defining a metric for comparison is straightforward. We compare the time required to complete each of a large number of randomly generated sets of requests using each of the three schedulers. Other parameters of the problem must also be specified, such as the cost of tour improvement and path planning relative to movement or relative to each other. In this section, we investigate the effects of varying several of these parameters on the relative performance of the three schedulers. We do this for each of two problem

models. First, we investigate their behavior for static problems, in which the requests all arrive at time 0 and the robot then plans for and moves among the requested locations. Next, we make the same kinds of comparisons for dynamic problems, in which requests arrive over time and the robot must replan as new information becomes available.

4.7.1 Static Behavior

In order to compare the behavior of the deliberation schedulers in a static environment, we repeatedly generated random tours of fixed size on a randomly generated instance of a standard gridworld. Every fifty tours, a new gridworld instance was generated. Each scheduler was run on each tour, and the expected time to complete the tour using the resulting deliberation schedule was recorded. We also recorded the time needed to complete the tour with no planning at all. We did this in four series of experiments in which different features of the problem were varied systematically in order to investigate how the relative behavior of the schedulers changed. The features varied included the size of the tour and the expense of tour improvement or path planning relative to movement. The expense of planning was varied by introducing a multiplier that determined the number (or fraction) of planning steps that could be accomplished in the time the robot took to move one location.

In the first set of experiments, we varied only the tour size, fixing the path planning and tour improvement multipliers at 0.5 (i.e., it takes 2 time units to accomplish one step for either path planning or tour improvement). The results are shown in Figure 4.26. There are two significant features to the behavior of the schedulers as the number of points increases. First, the difference in the time required for planning versus not planning depends on whether tour improvement is allowed. The expected time using Pr-I is approximately proportional to the time required with no planning at all, which is what we would expect. For Pr-II and Pr-III, however, the time required is not proportional—the advantage to planning grows disproportionately with increasing tour size. The probable explanation for this behavior is that the advantage of doing tour improvement grows with increasing tour size, as is shown by the fact that our estimator for the optimal tour length (Section 4.4.2) is a sublinear function of tour size. The length of a randomly generated tour will varies roughly linearly with tour size, as is shown by the data for the no-planning case in Figure 4.26.

The other point to be made is that the gap in expected times between Pr-I and Pr-II is enormous—Pr-II has roughly the same advantage over Pr-I that Pr-I does over not planning at all—while the difference between Pr-II and Pr-III is effectively too small to see. The data show only a small though systematic advantage to using Pr-III. There should be an advantage, since the worst Sched-3 can do is equivalent to Sched-2.

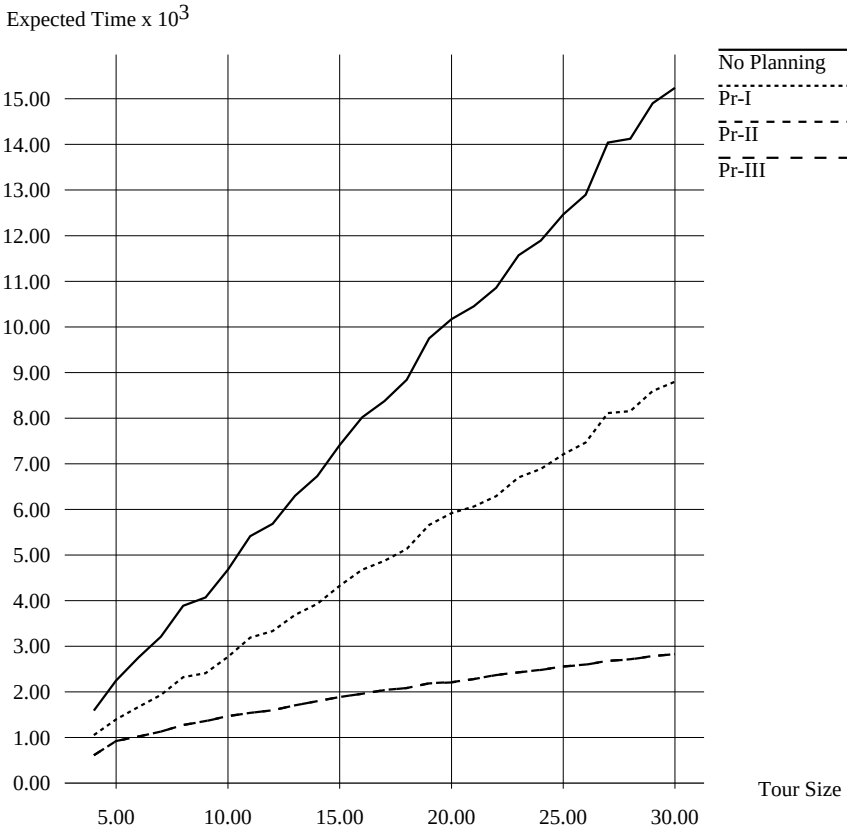

Figure 4.26: Varying tour size

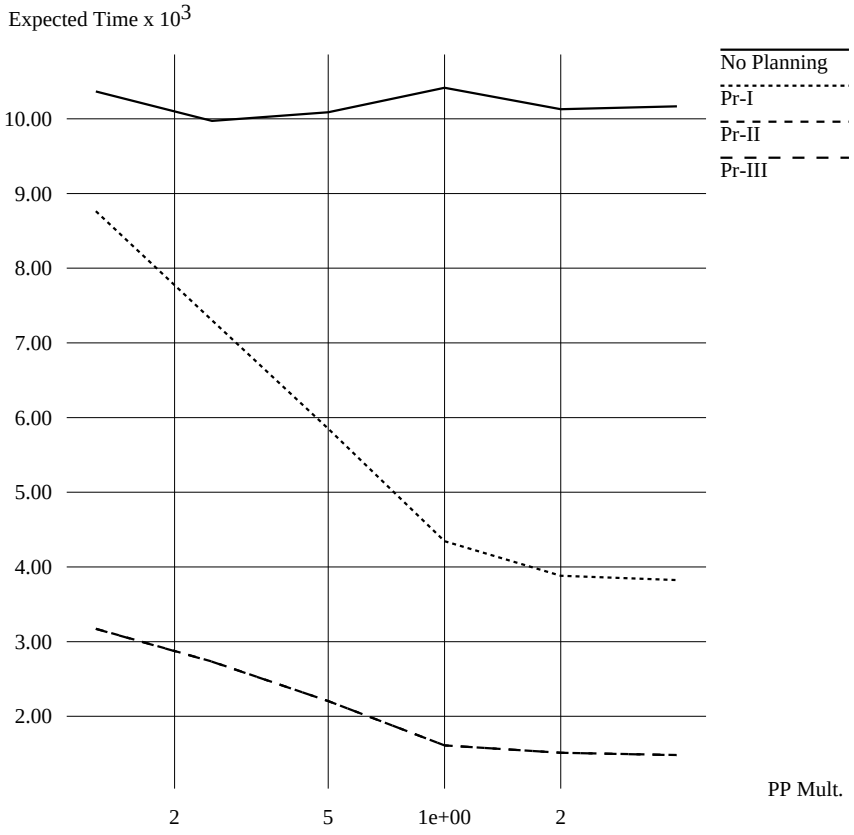

Figure 4.27: Varying path-planning rate

The fact that the advantage is so small indicates that there is rarely any benefit in tour improvement while moving among the first few locations on the tour. In other words, either situations like that in Figure 4.22 arise infrequently, or Pr-III has only a small advantage in such situations. Part of the reason for this behavior may be the fact that tour improvement is enormously more effective at reducing completion times than path planning. The derivative of the expected completion time for the first few time units allocated to tour improvement is between -50 and -100 for a tour of size 20. The gain γ for path planning is a constant—for this set of experiments, $\gamma = 0.75$. This is not the whole story, however. As is shown in Figure 4.28, even for a larger value of γ and a very small tour improvement multiplier, the difference between Pr-II and Pr-III is not large. In fact, Pr-III has the greatest relative advantage when *both* multipliers are small (Figure 4.29), which argues against the difference in gain for the two kinds of planning being an important factor.

The second set of experiments involved varying the multiplier for path planning, while holding the tour-improvement multiplier and tour size constant at values of 0.5 and 20, respectively. The results are shown in Figure 4.27, with a logarithmic scale on the x-axis. We hoped to gain two answers from this experiment. First, at what point does the problem "saturate," i.e., at what point does increasing the multiplier (decreasing the computational cost) for path planning bring little or no benefit? Second, how does varying the multiplier affect the relative performance of the three deliberation schedulers?

The graph in Figure 4.27 provides the information we seek. There is a clear decrease in expected time for any of the schedulers, up to a planning multiplier of about 2; after this there is little or no decrease, indicating that there is no further useful path planning to be done. As the multiplier decreases, the expected time required approaches the time for the no-planning case, as anticipated. Pr-II and Pr-III are again effectively identical in performance, and do not approach the no-planning case, because they still allocate time for tour improvement.

For the experiments graphed in Figure 4.28, we held the path-planning multiplier constant at 0.5 and the tour size constant at 20, and varied the tour-improvement multiplier over a range from 0.03 to 4.0. Again, the x-axis is logarithmic. As expected, varying the tour-improvement multiplier affected only Pr-II and Pr-III. Though there is not as clearly a point at which the problem "saturates" as when we varied the planning multiplier, the relative benefit of increasing computational power falls off fairly rapidly. As the tour-improvement multiplier approaches 0, Pr-II and Pr-III approach Pr-I in performance.

For the final set of experiments (Figure 4.29), we varied the path-planning and tour-improvement multipliers at the same time, starting both at 0.125 and doubling them at each iteration. The intent was to simulate a "clock rate" for the deliberation processor. The two previous sets of experiments tested the effect of varying the expense of path planning or tour improvement in isolation. In contrast, these data can give us some intuition for what would happen if we just installed a faster deliberation processor, without changing any of the anytime decision procedures or allocation strategies. For the first time there is a significant (visible, anyway) difference between Pr-II and Pr-III. Pr-III has a somewhat larger advantage when planning is very expensive (i.e., the multipliers are very small). This makes perfect sense, since there are situations that Pr-III can exploit that Pr-II cannot, and since the relative payoff for exploiting them is greater when computational resources are scarcer.

The other point clear from Figure 4.29 is that the system does exhibit the kind of "graceful degradation" we had intended. As the clock rate diminishes, the robot's performance smoothly approaches what it would be if no planning were done at all.

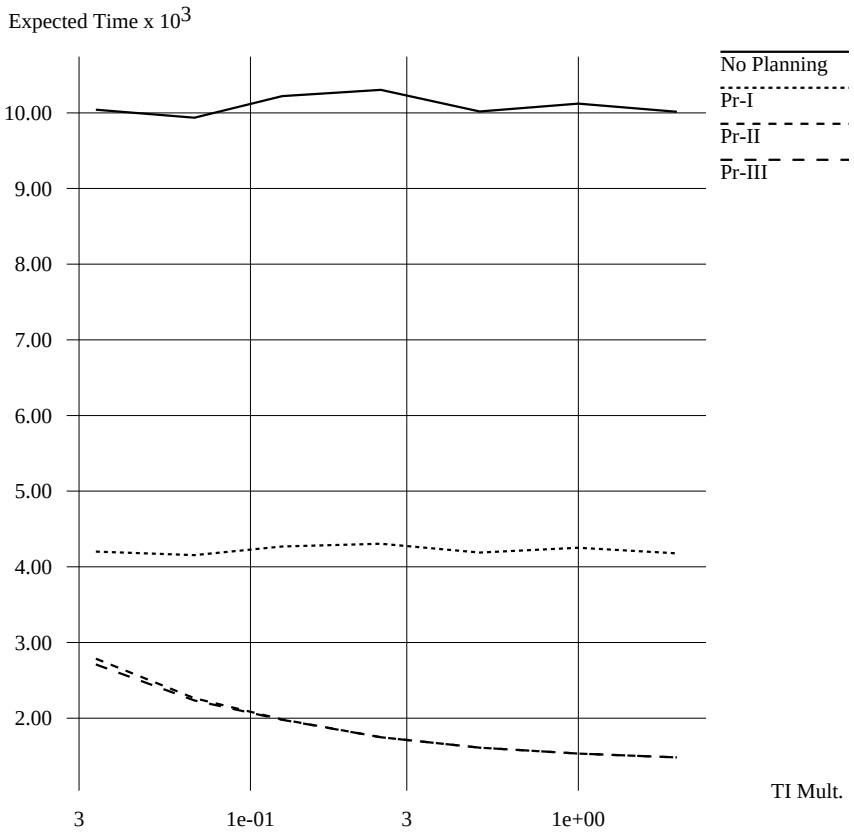

Figure 4.28: Varying tour improvement rate

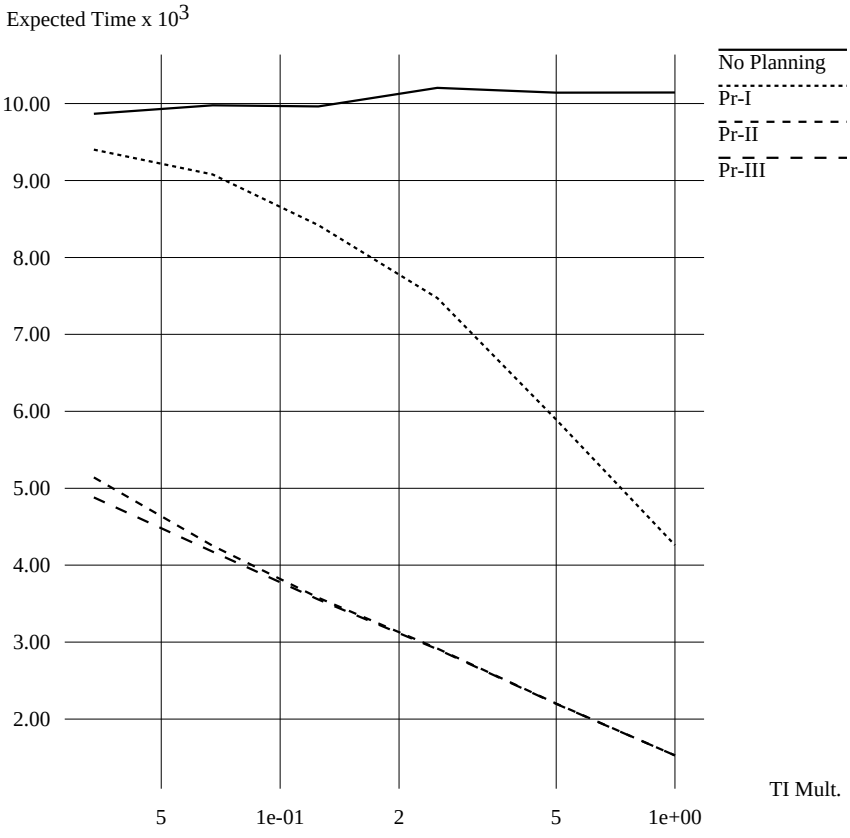

Figure 4.29: Varying "clock rate"

There is no point at which the resources available fall below some critical level and the system simply quits functioning. For systems interacting with highly unpredictable environment, this is a valuable trait.

The kinds of data we have gathered from these experiments could be very useful in the design of a time-dependent controller. Looking at the change in performance as the relative cost of deliberation changes makes clear the tradeoff between performance and computational power (which costs money, or battery power, or has some other form of opportunity cost).

4.7.2 Dynamic Behavior

The procedures in Section 4.6 provide optimal deliberation scheduling for a set of fairly restricted decision models. In particular, these models ignore the possibility of additional requests. In Section 4.5, we showed that optimal deliberation scheduling for a more general model is likely to be very expensive.

In Chapter 3, we showed that for time-dependent planning problems, using a similarly restricted model for which deliberation scheduling is computationally inexpensive produces only a small reduction in expected utility relative to an optimal (actually a super-optimal) deliberation scheduler.[11] Doing the same thing for the Robot Courier is a little more difficult, since designing a "perfect-information" scheduler is more difficult. There is also some question as to what constitutes "perfect information." In Chapter 3, the only additional information needed was the type and time of occurrence of future events. For the Robot Courier, using the distributions underlying the expectations compiled in the performance profiles in Section 4.4 may increase the expected utility. Compiling and employing these distributions will, however, complicate the problem considerably.

Our approach is instead to compare the performance of the three schedulers directly to each other, to the robot's performance doing no planning at all, and to their performance in the static case. In this section, we compare the performance of the schedulers for Pr-I, Pr-II, and Pr-III on a large sample of randomly generated dynamic problems in which requests are received over time and the robot must deal with new information as it becomes available. As the robot moves and deliberates, several types of new information become available. The robot may reach the next location earlier than expected, or fail to reach a location by the expected time. New requests may arrive. We can define a set of events corresponding to the first availability of information of each of these types:

[11] By "super-optimal" we mean an optimal deliberation scheduler for a decision model that includes more information than is actually available to the system, and is thus guaranteed to construct deliberation schedules at least as good as those constructed by an "optimal" deliberation scheduler.

- The arrival of a new request.
- The arrival of the robot at its destination.
- The failure of the robot to arrive at its destination within some ϵ of the expected time.

In the dynamic version of the Robot Courier problem, the deliberation schedulers are run only on the occurrence of one of these events. The rest of the time, the robot plans and moves according to the deliberation schedule and the tour ordering it currently has. If one of the anytime decision procedures completes a planning task ahead of time, the robot starts the next allocated planning task early.

Each set of experiments graphed below involved different fixed values for the planning multipliers and a varied distribution on the occurrence of requests over time. For each combination of values for these three parameters, 50 sets of requests were generated, each with a random time and location, and the behavior of the robot simulated as time passed, the robot planned and moved, and new requests arrived. The time the robot took to complete all the requests was recorded for each set and for each deliberation scheduler, as well as for the case where no deliberation was scheduled at all.

In the experiments whose results are graphed in Figure 4.30, the path-planning and tour-improvement multipliers were both set to 0.5. The probability of a request in any time unit was varied from 0.0025 to .025. The high variance in average times for the no-planning case can be attributed to the high variance in how long the dead reckoner actually takes to reach a goal. Every once in a while, it treats the gridworld boundary as an obstacle, and follows *it* clockwise for a space (say half way around). The same variation appears in the Pr-I data, and for the same reason—the robot is traversing the same set of points, apparently unable to do enough planning to avoid the long delays caused by the dead reckoner. This explanation is supported by the behavior of the Pr-I scheduler in Figure 4.31, where the path-planning multiplier has been doubled: here the variation shown for the no-planning case does not affect the Pr-I data. To see what is happening, consider what happens to the robot when requests are spaced so far apart that there are occasionally only two points in the tour (the current location and the last request). For a path-planning multiplier of 0.5, the gain γ is 0.75. For a tour of two locations, this means that no path planning is done at all. For a multiplier of 1.0, $\gamma = 1.5$, and the robot always plans a complete path between the two locations.

Comparing the results for Pr-II and Pr-III in Figures 4.30 and 4.31 shows that those schedulers are affected by this variation as well, though perhaps not as much. An interesting feature apparent in Figure 4.30 is that the dead reckoner's misbehavior can

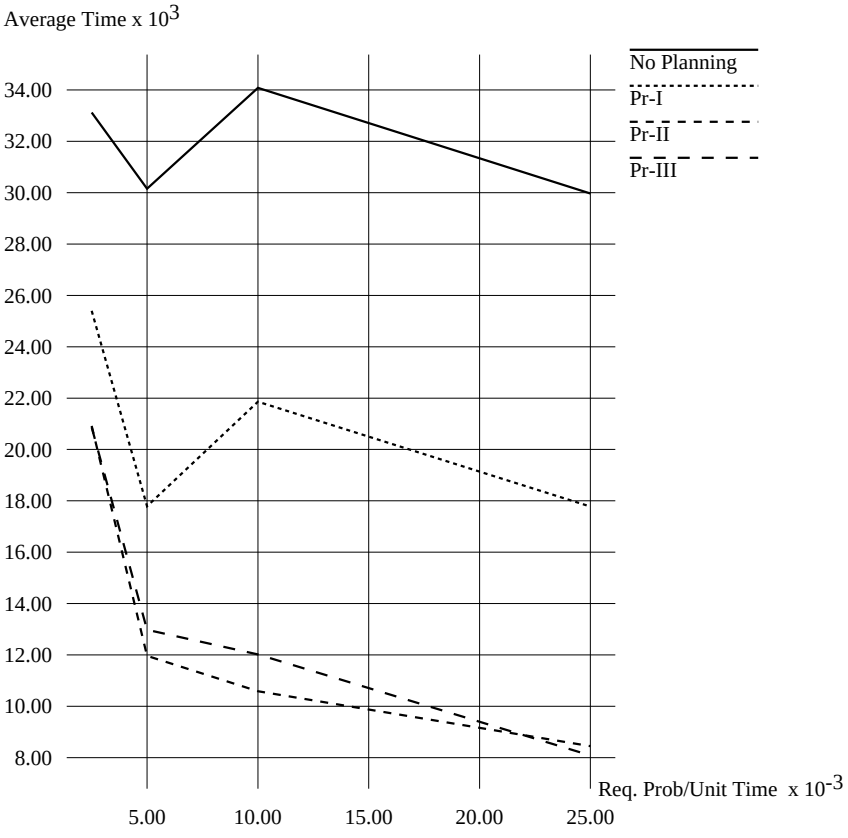

Figure 4.30: Dynamic deliberation scheduling

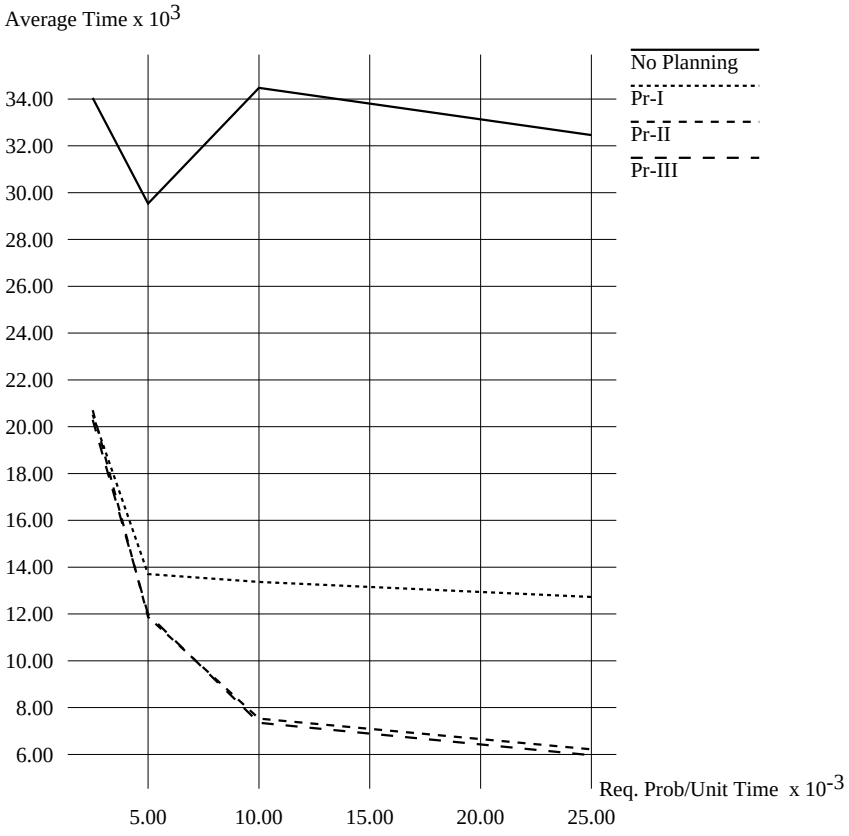

Figure 4.31: Making path planning less expensive

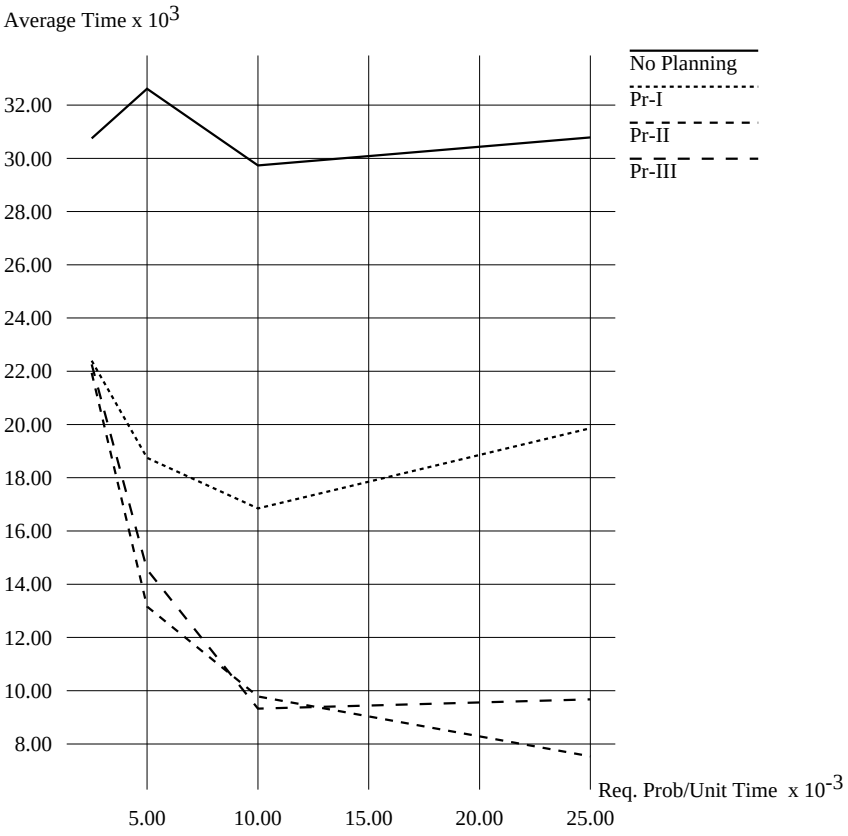

Figure 4.32: Making tour improvement more expensive

cause Pr-III to behave worse than Pr-II, since Pr-III is more dependent on accurate expectations on how long moving from location to location will take.

The effects of changing the tour-improvement multiplier are not as pronounced. In Figure 4.32, the path-planning and tour-improvement multipliers are 0.5 and 0.125, respectively, and in Figure 4.33, they are 1.0 and 0.125. Comparing those graphs to Figures 4.30 and 4.31 fails to reveal any pronounced difference in the behavior of any of the schedulers: Pr-I and Pr-III continue to be affected by variations in the behavior of the dead-reckoning system, and Pr-II is not, or not as much.

As the probability of requests occurring drops (the mean time between requests rises), all three schedulers realize smaller and smaller gains over not planning at all. This is because they have smaller tours to work with at any time. In the limit, when the request probability is very small, the robot sits at one location, waiting for a request. When the request arrives, the robot moves to that point, and then waits for the next

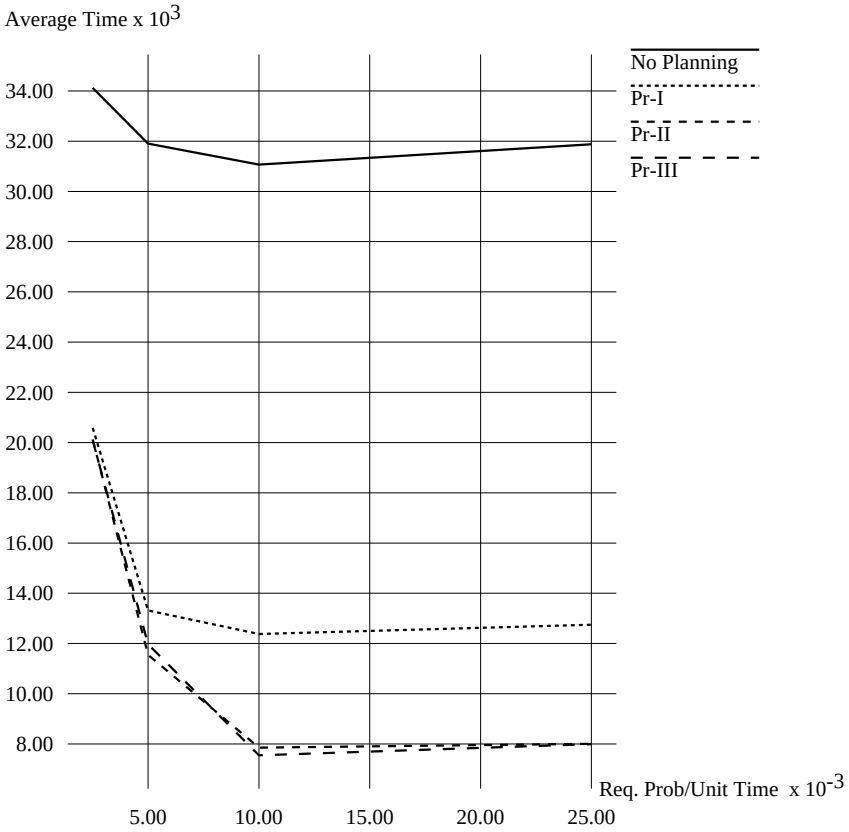

Figure 4.33: Expensive tour improvement and cheap path planning

request. No tour improvement is ever scheduled, and path planning time is allocated only if $\gamma > 1$, and then only for a single path at a time.

One reason for these experiments was to investigate how much a dynamic environment hurts the performance of each of these schedulers. The answer to this question depends on the mean time between requests. We have already discussed how the performance of all three schedulers falls off as the request probability drops. So let us choose an intermediate value, say 0.01 (mean time between requests of 100 time units, which is on the same order as the mean time to travel between locations), and compare the schedulers' performance to their performance in the static case. Comparing the graphs in Figures 4.27 and 4.30, we see that for path-planning and clock multipliers of 1.0 and 0.5, the gain in the average time required for Pr-I is less than 10%. Pr-II and Pr-III do not fare as well, suffering about a 40% gain in average completion time. From Figures 4.28 and 4.33, we see that Pr-I loses about 5% in the dynamic case, while

Pr-II and Pr-III suffer performance losses under 15%. One possible explanation for the difference in loss between Pr-I and Pr-II or Pr-III is the qualitative difference between path planning and tour improvement: new requests can render useless most or all of the path planning done so far, if they make it advisable to reorder the tour, while new requests simply added to the end of the current tour have less effect on what earlier path-planning allocations should be.

4.8 Summary and Conclusion

In this chapter, we have discussed a class of time-dependent problems, known collectively as the Robot Courier problem, in which there are no hard deadlines and tasks can be reordered. For this class of problems, utility is defined strictly in terms of how long the robot takes to complete a set of tasks. We define appropriate anytime decision procedures for deliberation, and construct performance profiles to provide estimates of the effect on the robot's performance of time allocated to these anytime decision procedures. We show that for the Robot Courier, just as for time-dependent planning problems, providing an optimal deliberation-scheduling policy for any decision model general enough to take into account the dynamic and uncertain nature of the problem is likely to be very difficult.

For any reasonably complex deliberation scheduling problem (we view the Robot Courier as a relatively simple problem), it seems likely that similar difficulties will arise in defining or calculating the "optimal" policy. We deal with this difficulty empirically: by generating optimal or near-optimal deliberation-scheduling policies for restricted decision models and testing those policies on real or simulated instances of the problem.

In this chapter, we provide deliberation schedulers for three increasingly complex decision models, all of which use expectations rather than distributions for performance profiles and ignore the possibility that additional requests (tasks) may occur. It is no surprise that these deliberation schedulers do not perform as well on dynamic problems (in which requests arrive as time passes) as on static problems (in which the requests are all made at time 0), but the average loss in performance in dynamic situations is fairly small.

In the process of comparing the various deliberation schedulers, we generate data of a sort that is likely to be very useful in the design of any time-dependent controller. We examine the change in performance as the relative cost of deliberation changes. This makes clear the tradeoff between performance and computational power (which costs money, or battery power, or has some other form of opportunity cost). We also saw the system does exhibit the kind of "graceful degradation" we had intended. As the clock rate diminishes, the robot's performance smoothly approaches what it would

be if no planning were done at all. There is no point at which the resources available fall below some critical level and the system simply quits functioning. For systems interacting with highly unpredictable domains, this is a valuable trait.

These results are particularly encouraging because this is the first empirical demonstration that expectation-driven iterative refinement is a useful framework for analyzing and solving time-dependent problems.[12] We start with a specific problem requiring certain kinds of deliberation and generate anytime decision procedures, performance profiles, and deliberation schedulers. We demonstrate that these anytime decision procedures and deliberation schedulers yield a system that functions well under widely varying resource demands and in the presence of considerable uncertainty. We also show that it is easy to obtain the information required to "tune" such a system for a particular application.

4.8.1 Extensions

The class of problems we have investigated in this chapter could be extended in many ways. As with the time-dependent planning problems explored in the last chapter, we could construct a lot of different problem classes that have the same fundamental structure. However, many of these classes may require fundamentally different deliberation scheduling strategies, and it is by no means clear how useful an exhaustive catalog of them would be, were it feasible to produce one.

Instead, we suggest some possible modifications to the basic problem and discuss why these modifications are interesting, and how they would affect deliberation scheduling. One obvious possibility that has repeatedly been suggested is to restrict ourselves to a single processor, so that deliberation and action must contend for resources. For the Robot Courier, this actually makes the optimization problem easier. We can consider each path and the attendant planning to be a separate task, where the minimum task length is a function only of the distance between locations and γ. Since γ is fixed, the problem reduces to figuring out how much tour-improvement to do. There is no real "scheduling" involved in allocating path-planning time.

There are several more interesting generalizations that could be made. In particular:

- Allow the utility function to include costs for missing deadlines for reaching specified locations (the Domino's Pizza delivery model). This changes the problem substantially in that however "tour improvement" is to be defined, it is not as simple as finding a minimum-length tour. One possible approach would be to use branch-and-bound search as an anytime algorithm for tour improvement.

[12] In Chapter 3, we assumed that we had the necessary anytime decision procedures and performance profiles.

- Allow fixed orderings on pairs of locations (suppose the robot picked things up to be dropped off somewhere else). This problem is equivalent to the Robot Courier problem as analyzed in this chapter, except that some of the tours returned by the current tour-improvement anytime decision procedure would have to be excluded, and the estimates used to construct the tour-improvement performance profile would have to be changed.

- Allow locations to have different payoffs, and allow the robot to put off visiting any particular location as long as it wants. A further complication would be to make the payoff depend on when the robot reached the location. This would change the problem substantially, especially in the dynamic case, where optimizing the robot's behavior might now be viewed as maximizing the payoff per some unit of time.

A difficulty we have successfully avoided so far is the construction of cumbersome joint distributions. This will be a difficulty both in constructing more complete decision models for this class of problems and in extending this class in certain ways (e.g., allowing the introduction of deadlines with a rapidly increasing tardiness cost). Throughout this chapter, we have constructed estimates for the expected value of various random variables, shown that the estimates were reasonable ones, and proceeded to combine them as though they represented deterministic values. This is not in general correct, but, for this problem, we noted that fact and proceeded, and were rewarded when the resulting systems behaved reasonably well.[13] Problem classes in which the deadline costs are more complex, or that involve causal reasoning with uncertain information, are likely to require constructing joint distributions for effective deliberation scheduling.

4.8.2 Future Work

The range of problems similar to the Robot Courier problem is extensive. Any problem involving tasks arriving over time, in which throughput is the main consideration and there is some latitude for reordering the tasks in hand should have similar characteristics (e.g., disk or task scheduling for an operating system).

The particular version of the Robot Courier problem investigated here has analyzed sufficiently to show that we have produced a useful solution using expectation-driven iterative refinement. Further work on problems in this class is likely to be application-driven. Given our demonstration that problems with this structure are amenable to solutions using expectation-driven iterative refinement, the next step appears to be for

[13] Henrion has shown [Henrion, 1984] that there are cases where using means rather than distributions does no damage at all (e.g., if utility is a linear function of the random variables).

someone to try applying it to some "real world" problem, or to let us know about the problem so that we can try it.

4.9 Proofs

4.9.1 Proof that Sched-1 is optimal

Let $\mathcal{T}$ be the total expected time for the robot to complete a tour, including planning time, for a given deliberation schedule. An optimal deliberation schedule allocates deliberation time so as to minimize $\mathcal{T} = \sum_{i=1}^{n} T_{i-1,i}(\delta_i)$. From Section 4.4.1 we have:

$$T_{i-1,i}(\delta_i) = \begin{cases} 3.17d/v - \gamma\delta_i & \delta_i < \delta_i^* \\ 1.17d/v & \delta_i \geq \delta_i^* \end{cases}$$

We make use of the following properties of deliberation scheduling for this particular problem:

1. Any optimal deliberation schedule can be transformed into a deliberation schedule in which the robot deliberates about each path in turn, in the order in which they occur, without increasing $\mathcal{T}$. This is because there are no release times: the robot knows all of the requests at the time the deliberation schedule is generated, and can allocate time to any path at any point in time before it starts moving on that path.

2. Any optimal deliberation schedule can be transformed into a deliberation schedule in which the robot plans without moving only at the very beginning of the tour, without increasing $\mathcal{T}$. We transform the deliberation schedule as for property 1, then delay the start of any motion such that there is a time following that motion in which the robot is deliberating but not moving.

3. For any locations l_i and l_j, the change in *travel time* in the tour with deliberation time δ_i is the same as for time allocated to δ_j, provided $\delta_i < \delta_i^*$ and $\delta_j < \delta_j^*$. This is because γ is a constant.

The proof has two parts. First, we trace the execution of the algorithm to show that the deliberation schedule generated is a local minimum for $\mathcal{T}$. Then we show that this local minimum must be a global minimum.

Proof:

We start with small cases: $n = 1$:

When **Sched-1** reaches exit, $t = \mathsf{begin}(l_1)$, $\mathsf{gap} = \gamma\delta_1^*$, $j = 1$, and $\delta_1 = 0$. $\mathcal{T} = \delta_1 + T_{0,1}(\delta_1) = \delta_1 + \max(d/v, 3.17(d/v) - \gamma\delta_1)$. $\gamma\delta_1^* = 2(d/v)$. Thus $\mathcal{T}$ is minimal for

$\delta_1 = \delta_1^*$ if $\gamma > 1$ and minimal for $\delta_1 = 0$ otherwise. The code following exit will allocate δ_1 appropriately, depending on the value of γ.
$n = 2$.

In the main loop in **Sched-1**, for $j = 2$ the step marked with an asterisk will allocate δ_2 such that either:

1. $\mathsf{begin}(l_2) - \delta_2 > \mathsf{begin}(l_1)$, and $\delta_2 = \delta_2^*$, or
2. $\mathsf{begin}(l_2) - \delta_2 = \mathsf{begin}(l_1)$.

In the first case, there is no benefit to increasing δ_2. The interval between $\mathsf{begin}(l_1)$ and $\mathsf{begin}(l_2) - \delta_2$ cannot be used for deliberating about the path to $l1$. If $\gamma > 1$, there may be some allocation made to δ_1. For $j = 1$, $\mathsf{gap} > 0$, and so δ_1 is allocated in the code following exit so that either the idle time gap disappears, or $\delta_1 = \delta_1^*$. If $\delta_1 \neq \delta_1^*$, $\mathsf{begin}(l_1) + \delta_2 = \mathsf{begin}(l_2)$, and so $\mathcal{T} = \delta_1 + \delta_2 + T_{1,2}(\delta_2)$. Any further allocation to δ_1 will increase $\mathcal{T}$.

In the second case, we go to exit with $j = 2$. $\mathsf{gap} > 0$, so we consider increasing δ_2. Since in this case $\mathsf{begin}(l_2) - \delta_2 = \mathsf{begin}(l_1)$, $\mathcal{T} = \delta_1 + \delta_2 + T_{1,2}(\delta_2)$. Using the same argument as for $n = 1$, we see that if $\gamma > 1$, $\mathcal{T}$ is minimal for $\delta_2 = \delta_2^*$, and otherwise δ_2 should not be increased. Any value of δ_1 greater than 0 will increase $\mathcal{T}$.

For arbitrary n, the problem is only slightly more complicated. Consider the schedule when the procedure arrives at exit: for any $i > j$, $\delta_i = \delta_i^*$, and so no expected savings in travel time will result from increasing any of those allocations. We have to justify not increasing $\delta_i, i \leq j$. But we know that $\mathcal{T} = \sum_1^j \delta_i + \sum_j^n T_{m-1,m}(\delta_m^*)$. Unless $i = j$, adding to δ_i can only increase $\mathcal{T}$. If $\mathsf{gap} > 0$ and $\gamma > 1$, δ_j will be allocated so that either the idle time gap disappears or $\delta_j = \delta_j^*$. Any additional allocation to δ_j would increase $\mathcal{T}$. Property 3 above ensures that reallocating time to δ_j from some other δ_i will not decrease $\mathcal{T}$, either.

The local minimum for $\mathcal{T}$ achieved using the deliberation schedule generated by this algorithm is also a global minimum. The deliberation scheduling problem can be framed as a problem of minimizing a linear function of n variables

$$T = \Delta + \sum_1^n 3.17d/v - \gamma(\delta_i)$$

under a set of linear constraints:

- $0 \leq \delta_i \leq \delta_i^*$
- $\delta_m + \sum_1^{m-1}(1+\gamma)\delta_i - \Delta \leq (3.17/v)\sum_1 m - 1d_{i-1,i}\ \ 1 \leq m < n$
- $\Delta \geq 0$

Any consistent set of linear constraints defines a convex polytope within which the solution must lie. We are minimizing a linear function over a convex polytope, and thus any local minimum is also a global minimum [Papadimitriou and Steiglitz, 1981]. □

4.9.2 Proof that Sched-2 is optimal

From Section 4.6.2, $g(\mathcal{S}, \delta')$ is the expected length of the improved tour $\mathcal{S}'$, given δ' units spent on improving tour $\mathcal{S}$. $\alpha * g(\mathcal{S}, \delta')/k$ is the estimate for how long it will take to traverse $\mathcal{S}'$, given an optimal deliberation schedule for path planning, which can be produced by running **Sched-1** on $\mathcal{S}'$. The expected value of $\mathcal{T}$ given δ' time units spent on tour improvement is $\delta' + \alpha * g(\mathcal{S}, \delta')/(n-1)$, where n is the number of locations in $\mathcal{S}$. **Sched-2** picks the value of δ' such that the expected time T is minimized (expected utility is maximized).

Minimizing the function $\delta' + (\alpha * g(\mathcal{S}, \delta')/(n-1))$ is straightforward. The function g has the form $(L_0 - L^*)(1 - e^{\lambda\delta'/\tau_{ti}})$, where L_0, L^*, λ, and τ_{ti} are all constants, given a tour $\mathcal{S}$. τ_{ti} is the time required to do an edge-exchange on an existing tour (see Section 4.4.2 for more details). So we want the value of δ' minimizing the value of a function of the form $f(\delta') = a + b\delta' + ce^{d\delta'}$, where a, b, and $c > 0$, and $d < 0$. The first derivative $f' = b + cde^{d\delta'} = 0$ for some value of δ' ($cde^{d\delta'}$ approaches 0 from below for increasing δ', and $b > 0$). The second derivative $f'' = cd^2e^{d\delta'} > 0$, and so there is a single global minimum for $\delta \geq 0$.[14] □

4.9.3 Proof that Sched-3 is optimal

Sched-3 checks every possible interleaving of tour improvement and path planning and traversal allowed by the statement of the problem Pr-III (i.e., every possible value for the pivot dividing the two parts of the tour). It remains to be shown that **Sched-3a** correctly estimates the expected utility of each alternative. Having the pivot at l_n or l_{n-1} is equivalent to using Pr-I—no tour improvement will be done—but more expensive, since the deliberation time for paths after the pivot will never be allocated before the pivot. Having the pivot at l_0 results in behavior identical to running **Sched-2**.

Proof:

The algorithm initially allocates the time spent travelling between l_0 and the pivot l_p to tour improvement. Since this time figures in the expected completion time for the tour, this allocation will certainly decrease the expected completion time. What the bulk of the procedure accomplishes is to decide for each increment of time whether

[14] In fact, we can find that minimum analytically, by solving $f'(\delta') = b + cde^{d\delta'} = 0$.

the expected completion time would decrease if that increment were spent on planning some path instead of on tour improvement. This results in a local minimum for the expected time T, which we show is also a global minimum.

The central loop in **Sched-3a** allocates time for path planning exactly as does **Sched-1**, and thus has the property that, given that path planning has a higher expected utility, deliberation time is allocated to the correct path. What is different is the comparison of $\mathcal{T}'$ and γ to determine which type of planning to spend the time on. Given a pivot l_p, for any δ_m, $1 < m \leq p$, $\delta_m < \delta_m^*$, allocating an increment dt to δ_m instead of to δ' changes the expected time to complete the tour by $\mathcal{T}'\mathsf{dt} - \gamma\,\mathsf{dt} + \gamma\mathcal{T}'\mathsf{dt}$. Thus the statement marked with an asterisk takes time from tour improvement for path planning just in case that decreases the expected completion time. The code following exit first determines whether or not to allocate time to path planning before traveling, making the same checks as **Sched-1** and also determining that the net effect of decreasing the time available for tour improvement is in the right direction. It may then allocate time to tour improvement, also before traveling, if the net effect will be to decrease the expected completion time. Summing the resulting travel times and the time allocated to planning before the robot moves gives the expected utility for the current choice of pivot p.

The local minimum thus generated is also a global minimum. Just as for **Sched-1**, minimizing T involves minimizing a function that has a single global minimum over a convex polytope. Gradient descent will always reach the minimum, and so the local minimum found by **Sched-3a** must be a global minimum. The function is of the form:

$$T = \Delta' + \Delta + \sum_{1}^{p} T_{i-1,i}(\delta_i) + a + b(\delta' + \Delta') + ce^{d(\delta'+\Delta')}$$

where a, b, c ¿ 0, and $d < 0$. Δ' is the time spent on tour improvement before moving, Δ is the time spent on path planning before moving, and δ' is the time spent on tour improvement while moving to the pivot. a is the time spent on path-planning before moving after reaching the pivot l_p, and b and c are constants determined by the part of the tour $\mathcal{S}$ after the pivot l_p. The constraints defining the polytope are:

- $0 \leq \delta_j \leq \delta_j^*$
- $\delta' \geq 0$
- $\Delta \geq 0$
- $\Delta' \geq 0$
- $\delta_m + \sum_1^{m-1}(1+\gamma)\delta_i - \Delta \leq (3.17/v)\sum_1 m - 1 d_{i-1,i},\ 1 \leq m \leq p$

□

Chapter 5

The Shooting Gallery

5.1 Introduction

Suppose that a robot stands in front of an arcade shooting gallery. Targets appear at irregular intervals on one side of a screen some distance in front of the robot and cross to the other side. The robot uses a gun to try to hit these targets. Hitting one of the targets with a projectile has a known value. The robot also has two kinds of sensors. The first sensor provides information essentially for free, but is not very useful; it tells the robot when a new target appears, but only roughly where, and can also be used to determine when a target disappears off the other side of the screen. The second sensor is expensive to use in that there is a delay between asking for a sensor reading and obtaining the information. Since the sensor cannot be used again during this delay, there is a fixed minimum interval between sensor readings. This second sensor can be used to find the approximate position of a specified target. It returns a triple of time, x and y values, where the x and y values include gaussian noise. The value of each target the robot hits is added to its score. The robot's task is to accumulate as many points as possible from a fixed set of targets. This domain, the *Arcade*, is shown in Figure 5.1.

The resulting time-dependent problem (the optimization problem faced by the robot) is the *Shooting Gallery* problem. The Shooting Gallery problem is designed to have the following features:

- Prediction – the robot uses expectations regarding the future course of events. If the robot uses no other expectations, the delay involved in aiming the gun and the projectile's travel from the gun to the screen requires the robot to "lead" the targets.

- Incomplete information – the robot knows about the location and movement of

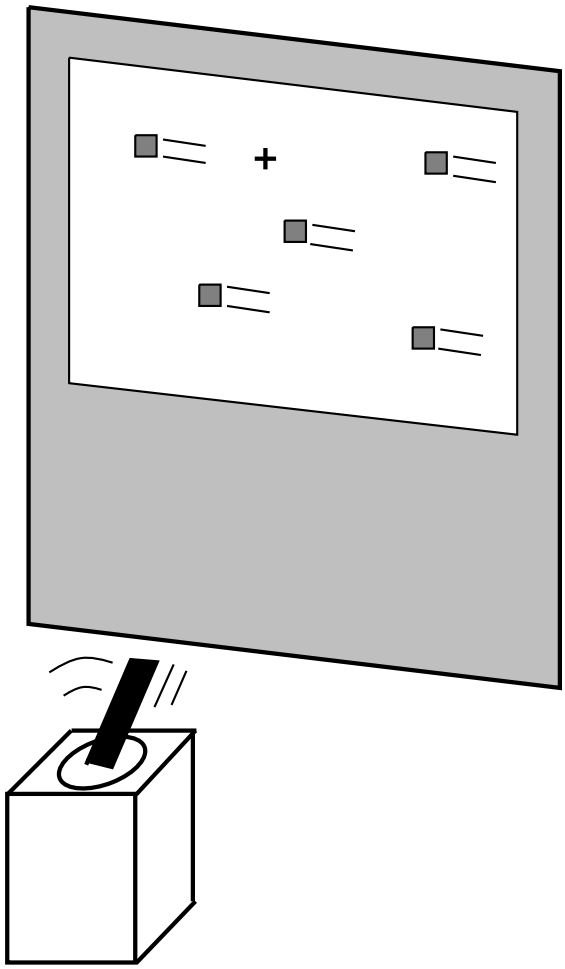

Figure 5.1: The Arcade

targets only through the information returned by its sensors. More information is available, but the robot must decide how to make use of scarce resources for sensing.

- Deadlines – each target must be hit before it runs off the left edge of the screen.
- Local optimization – hitting each target is a separate problem, with separate possibilities for optimization.
- Global optimization – the separate problems of hitting each target are interdependent. For example, the order in which the robot tries to hit targets influences how far the aim of the gun must move between one target and the next.

The Shooting Gallery problem is considerably more complex than either time-dependent planning or the Robot Courier. As we show in this chapter, the Shooting Gallery is substantially different from the first two problems. For the earlier problems, using a decision model that ignored the future occurrence of events had only a small effect on the expected utility of the resulting schedules. For the Shooting Gallery, the cost of ignoring future events is no longer negligible (see Section 5.7).

In the next section, we define the Shooting Gallery problem more precisely. Section 5.3 gives a formal specification in terms of the language developed in Chapter 2. In Section 5.4, we derive some necessary estimation and probability calculations. Section 5.5 presents a simple control system for the robot using no deliberation. In Section 5.6, we define anytime decision procedures for target and sensor scheduling. Section 5.7 presents a simple deliberation scheduler for these procedures that is optimal for a restricted decision model. Section 5.8 discusses the calculation of expected utility and optimal deliberation scheduling, including some conclusions drawn from the results of Section 5.7.

5.2 The Shooting Gallery

The *target area* is a rectangle of known dimensions (580×400 units) placed directly in front of the robot at a distance of 1000 units. The robot controls a *gun* that can be aimed at any part of the target area. The aim of the gun (where it's pointing in the target area) moves at a fixed velocity of 15 units per time unit, and only parallel to the x and y axes. The projectiles it fires are ballistic—they hit the point the gun was aiming at when they were fired. There is no uncertainty in where the gun is aiming, how long the projectiles take to get to the screen (40 time units), or where they hit. The gun has a maximum firing rate of once every 20 time units.

At intervals governed by an exponential distribution (mean time between targets is 50 time units), targets appear on the right edge of the target area, moving with x and y velocities determined by independent uniform distributions over fixed ranges (-1.5 to -3.0 and -0.25 to 0.25, respectively). Targets are rectangular (56×36 units). The value associated with a target is either 10 or 50, with probabilities of .75 and .25 respectively. Targets move across the target area from right to left, disappearing when they reach the left edge or when hit by a projectile from the gun.

The only direct information the robot has about the state of the world comes from two sensors. The robot has one sensor that operates for free and informs it when a target appears or disappears. For a target that has appeared, the sensor also provides the target's value and a very noisy estimate of its position; this estimate is a triple of x and y values and the time the target appears. The x and y values include gaussian noise with a standard deviation of 100 units. The expected error in the target's initial position is thus of the same order as the size of the target area.

The robot has a second sensor that takes as input a target to look for and, after a fixed delay, returns a triple of x, y and time values. Gaussian noise is added independently to the x and y values, with a standard deviation of 10 units. Using the initial position estimate (from the first sensor) and some number of readings from this sensor, we can derive x and y estimates for a target's position at any future time. The expected error for these estimates is gaussian, with mean 0 and known variance (see Section 5.4).

The following calculations may yield some intuitions about this domain:

- The mean time required for a target to cross the target area: $500/2.25 = 222$ time units.
- The mean number of targets in the target area at any one time: approximately $222/50 = 4.4$.
- The mean number of sensor readings the robot can take per target: $50/20 = 2.5$.
- The time required to move the aim of the gun from the top to the bottom of the target area: $400/15 = 26.7$ time units.
- The approximate probability of hitting a target with one shot given one recent sensor reading: 0.07.
- The approximate probability of hitting a target with one shot given five sensor readings: 0.45.
- The approximate probability of hitting a target with one shot given ten sensor readings: 0.9.

The probabilities are estimates, because they depend on when the readings are taken and when the robot fires at the target. These probabilities and the mean number of readings per target make it clear that the robot cannot gather enough sensor data to ensure a high probability of hitting every target that appears. In addition, the time required to move the gun in one direction across the target area is over half the expected time between targets. These two observations form the basis for our deliberation procedures. One procedure determines which targets to shoot at and when. The other determines what sensor readings to take to maximize the expected value of the targets hit, given a particular schedule of targets to shoot at.

5.3 Formal Statement of the Problem

The Shooting Gallery problem consists of

- A set of *event types* $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$. The only interesting events that are not robot actions are the appearance and disappearance of targets and the impact of projectiles on a target (in which case the target disappears) or on the target area.

- A set of *actions* $\mathcal{A} \subseteq \mathcal{T}$. At any one time, the robot can:

  - Change the motion of the gun. There are five possibilities: up, down, left, right, and none.
  - Use the sensor on a specified target, if sufficient time has elapsed since the last time. If it has not, nothing happens.
  - Fire the gun, if sufficient time has elapsed since the last time. If it has not, nothing happens.

  These actions "take time" in that sensor information is returned after a fixed delay (during which no further sensor actions have any effect), and in that there is a minimum delay between successive firings of the gun.

- A set of *histories* $\mathcal{H} \subseteq 2^{\mathcal{T} \times \Re}$. Each element h of $\mathcal{H}$ is a set of ordered pairs $\{(\tau, t) | (\tau \in \mathcal{T}) \wedge (t \in \Re)\}$. We call these pairs *events*. Given an event $e = (\tau, t)$, $\mathsf{type}(e)$ denotes the event type τ, and $\mathsf{occ}(e)$ denotes the time of occurrence t relative to some global clock.

- A time t_0 such that $\forall h \in \mathcal{H}, \forall e \in h, t_0 \leq \mathsf{occ}(e)$.

- A set of *internal states* $\mathcal{S}^{int}$. Each internal state is a pair of vectors $\langle V_s, V_d \rangle$. V_s is the *sensor data*, including the gun's current position, the last reading the active

sensor returned, and any new appearance or disappearance of a target. V_d is the *deliberation scratchpad.*

- A set of *external states* $\mathcal{S}^{ext}$. The external state consists of the current set of targets crossing the target area, the aim of the gun, any projectiles in flight, the last time the gun was fired, and the last time an active sensor reading was taken.

- A set of *initial states* $\mathcal{I}$. Each initial state i is an ordered pair $\langle s_{int}, s_{ext} \rangle$. s_{int} specifies the system's internal state at t_0. s_{ext} specifies the external state at t_0. Initially, the gun is centered in the target area, there are no targets on the screen, and the gun and sensor are ready to be used.

- A utility function $U : \mathcal{H} \times \mathcal{I} \rightarrow \Re$. If targets $\{x_1, \ldots, x_n\}$ appear in the target area and the robot manages to hit the targets $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \subseteq \{x_1, \ldots, x_n\}$, then
$$U = \frac{\sum_{j=1}^{k} \mathsf{value}(x_{i_j})}{\sum_{j=1}^{n} \mathsf{value}(x_j)}$$

An *instance* of the Shooting Gallery problem specifies a set of target appearances, including their time of appearance, values, and x and y velocities. A *solution* for the Shooting Gallery problem includes:

- A set of *anytime decision procedures* $\mathcal{P}$ for deliberation. See Section 5.6.

- An *agenda* of intended actions. The agenda is an ordered list of events corresponding to actions the system has committed to taking at the time indicated. Actions are added to the agenda only by deliberation. As time progresses ($\hat{t}$ increases), actions on the agenda are taken at the specified times.

- A *deliberation-scheduling policy.* This policy is implemented by a deliberation scheduler that makes allocations to the elements of $\mathcal{P}$.

5.4 Calculations From Sensor Data

In order to make decisions about what to do, the robot needs to be able to calculate the effects of those decisions. In this section, we derive estimates on the motion of the targets, given a set of sensor readings. We also show how to calculate the probability of hitting a given target by aiming at a specified point in the target area and firing at a specified time.

5.4.1 Estimation

Target motion is defined by the equations:

$$X(t) = X_0 + M_x(t)$$

$$Y(t) = Y_0 + M_y(t)$$

The available information consists of an initial position (x_0, y_0, t_0) with Gaussian noise added independently to x_0 and y_0, and a set of sensor readings $\{(x_1, y_1, t_1), \ldots, (x_n, y_n, t_n)\}$, again with Gaussian noise in x_i and y_i. The variance of the initial position error is not the same as the variance of errors in the active sensor.

We need estimates for $X(t)$ and $Y(t)$. We also need to know how the estimates are distributed around the correct values for $X(t)$ and $Y(t)$. Because the x and y calculations are identical and independent, we discuss only the estimation of $X(t)$ in the rest of this section. In Section 5.4.2, we combine the results of the x and y estimations to find the probability of hitting a target.

$X(t)$ is linear in t, so we use a minimum-squared-error (MSE) estimator. If the variances in the individual x estimates were all equal (suppose we did not have the pair (x_0, t_0)), the MSE estimate would be:

$$\hat{x} = \overline{x} + \frac{\sum_1^n x_i t_i - n\overline{x}\,\overline{t}}{\sum_1^n t_i^2 - n\overline{t}^2}(t - \overline{t})$$

where $\overline{x} = \frac{1}{n}\sum_1^n x_i$ and $\overline{t} = \frac{1}{n}\sum_1^n t_i$. However, the initial reading x_0 is distributed with a different variance. Let σ^2 be the variance of the n readings obtained from the active sensor and σ_0^2 be the variance of the initial position x_0. Let $w_0 = 1/\sigma_0^2$, $w = w_{i>0} = 1/\sigma^2$, and define the weighted means:

$$\begin{aligned} \overline{x} &= \frac{\sum_0^n w_i x_i}{\sum_0^n w_i} = \frac{w_0 x_0 + w\sum_1^n x_i}{w_0 + nw} \\ \overline{t} &= \frac{\sum_0^n w_i t_i}{\sum_0^n w_i} = \frac{w_0 t_0 + w\sum_1^n t_i}{w_0 + nw} \end{aligned}$$

Then, following [Green and Margerison, 1978], the MSE estimator for $X(t)$ is:

$$\begin{aligned} \hat{x}(t) &= \frac{\sum_0^n w_i x_i}{\sum_0^n w_i} + \frac{\sum_0^n w_i x_i t_i - (\sum_0^n w_i x_i)(\sum_0^n w_i t_i)/(\sum_0^n w_i)}{\sum_0^n w_i t_i^2 - (\sum_0^n w_i t_i)^2/(\sum_0^n w_i)}(t - \overline{t}) \\ &= \overline{x} + \frac{x_0 w_0 t_0 + w\sum_1^n x_i t_i - (w_0 x_0 + w\sum_1^n x_i)\overline{t}}{w_0 t_0^2 + w\sum_1^n t_i^2 - (w_0 + nw)\overline{t}^2}(t - \overline{t}) \end{aligned}$$

We can use this equation to show that $\hat{x}(t)$ is a linear combination of the Gaussian variables $\{x_0, x_1, \ldots, x_n\}$, and is thus itself Gaussian. $\hat{x}(t)$ is an unbiased estimator for

$X(t)$, so $E(\hat{x}(t)) = X(t)$. We can also calculate the variance of $\hat{x}(t)$:

$$\begin{aligned}
\hat{x}(t) &= \frac{\sum_0^n w_i x_i}{\sum_0^n w_i} + \frac{\sum_0^n w_i x_i t_i - (\sum_0^n w_i x_i)(\sum_0^n w_i t_i)/(\sum_0^n w_i)}{\sum_0^n w_i t_i^2 - (\sum_0^n w_i t_i)^2/(\sum_0^n w_i)}(t - \overline{t}) \\
&= \sum_0^n w_i x_i \left(\frac{1}{\sum_0^n w_i} + \frac{t_i - (\sum_0^n w_i t_i)/(\sum_0^n w_i)}{\sum_0^n w_i t_i^2 - (\sum_0^n w_i t_i)^2/(\sum_0^n w_i)}(t - \overline{t}) \right) \\
&= \sum_0^n w_i x_i \left(\frac{1}{\sum_0^n w_i} + \frac{(t_i - \overline{t})(t - \overline{t})}{\sum_0^n w_i t_i^2 - (\sum_0^n w_i)\overline{t}^2)} \right) \\
V(\hat{x}(t)) &= V\left[\sum_0^n w_i x_i \left(\frac{1}{\sum_0^n w_i} + \frac{(t_i - \overline{t})(t - \overline{t})}{\sum_0^n w_i t_i^2 - (\sum_0^n w_i)\overline{t}^2)} \right) \right] \\
&= \sum_0^n V\left[w_i x_i \left(\frac{1}{\sum_0^n w_i} + \frac{(t_i - \overline{t})(t - \overline{t})}{\sum_0^n w_i t_i^2 - (\sum_0^n w_i)\overline{t}^2)} \right) \right] \\
&= \sum_0^n V(x_i) w_i^2 \left(\frac{1}{\sum_0^n w_i} + \frac{(t_i - \overline{t})(t - \overline{t})}{\sum_0^n w_i t_i^2 - (\sum_0^n w_i)\overline{t}^2)} \right)^2 \\
&= \frac{1}{\sum_0^n w_i} + \sum_0^n w_i * \frac{2}{\sum_0^n w_i} * \frac{(t_i - \overline{t})(t - \overline{t})}{\sum_1^n w_i t_i^2 + (\sum_0^n w_i)\overline{t}^2} \\
&\quad + \sum_0^n w_i \frac{(t_i - \overline{t})^2 (t - \overline{t})^2}{w \sum_0^n t_i^2 - (\sum_0^n w_i)\overline{t}^2)^2} \\
&= \frac{1}{w_0 + nw} + \frac{(t - \overline{t})^2}{w_0 t_0^2 + w \sum_1^n t_i^2 - (w_0 + nw)\overline{t}^2}
\end{aligned}$$

Since $\hat{x}(t)$ is Gaussian, we now know the distribution of the error between the estimate and the actual position of the target at time t: $\hat{x}(t) - X(t)$ is a Gaussian random variable with mean 0 ($\hat{x}$ is an unbiased estimator) and variance as given above.

Suppose we have taken the first k sensor readings (i.e., we know the values of $\{x_1, \ldots, x_k\}$) and want to know how an additional $n - k$ readings will change $\hat{x}(t)$. Because $\hat{x}(t)$ is dependent on when readings are taken, we need to specify when these additional readings will be taken (the values of $\{t_{k+1}, \ldots, t_n\}$). Let $\hat{x}_k(t)$ be the estimate calculated from $\{(x_1, t_1), (x_2, t_2), \ldots, (x_k, t_k)\}$. Then $\hat{x}_k(t) - X(t)$ is a Gaussian random variable with mean 0 (because $\hat{x}(t)$ is an unbiased estimator). $\hat{x}(t) - \hat{x}_k(t)$ is a random variable dependent only on $\{x_{k+1}, \ldots, x_n\}$ and is thus independent of $\hat{x}_k(t) - X(t)$ (which depends only on $\{x_0, \ldots, x_k\}$). So

$$E(\hat{x}(t) - \hat{x}_k(t)) = E(\hat{x}(t) - X(t)) - E(\hat{x}_k(t) - X(t)) = 0 - 0 = 0$$

Thus $E(\hat{x}(t)) = \hat{x}_k(t)$, given the first k sensor readings. To calculate the variance, we use the fact that the only random variables involved in calculating $\hat{x}(t)$ are $\{x_{k+1}, \ldots, x_n\}$:

$$V_k(\hat{x}) = w \sum_{k+1}^n \left(\frac{1}{w_0 + nw} + \frac{(t_i - \overline{t})(t - \overline{t})}{w_0 t_0^2 + w \sum_1^n t_i^2 + (w_0 + nw)\overline{t}^2} \right)^2$$

$V_k(\hat{x})$ is the variance of a random variable corresponding to how far the new estimate $\hat{x}(t)$ will be from the old estimate $\hat{x}_k(t)$ (for the same time t). This is useful for determining how the readings $\{x_{k+1}, \ldots, x_n\}$ are likely to affect the results of deliberation on target scheduling and sensor scheduling.

5.4.2 Calculating Probabilities

The results of the previous section provide distributions on the errors involved in the estimates of a target's x and y positions at a specified time, given sensor readings that the robot has taken or will take. We need to calculate the expected utility of the schedule of movement, firing, and sensor readings that describe the system's performance. One question we need to answer is: what is the probability of hitting a target at time t by aiming at $(\hat{x}(t), \hat{y}(t))$, given the information then available? Arcade targets are rectangles of fixed size. Let r_x = half the x dimension of a target and r_y = half the y dimension. Then the probability of hitting a target at time t by aiming at $(\hat{x}(t), \hat{y}(t))$ is:

$$\begin{aligned} P &= P(|X(t) - \hat{x}(t)| < r_x) * P(|Y(t) - \hat{y}(t)| < r_y) \\ &= [\Phi_{x,t}(r_x) - \Phi_{x,t}(-r_x)] * [\Phi_{y,t}(r_y) - \Phi_{y,t}(-r_y)] \end{aligned}$$

$\Phi_{x,t}$ and $\Phi_{y,t}$ are Gaussian distribution functions having mean 0 and variance calculated as in the last section, using the x and y sensor variances and the time t. If the robot fires the gun without being in precisely the right place (according to its own estimates), we also need the probability of hitting a target at time t by aiming at $(\hat{x}(t) + dx, \hat{y}(t) + dy)$:

$$\begin{aligned} P &= [\Phi_{x,t}(r_x - |dx|) - \Phi_{x,t}(-r_x - |dx|)] * \\ &\quad [\Phi_{y,t}(r_y - |dy|) - \Phi_{y,t}(-r_y - |dy|)] \end{aligned}$$

5.5 Acting Without Deliberating

At each point in time, the only decisions available to the robot are: whether or not to fire the gun, in what direction to move the gun's aim, and whether or not to employ the active sensor and if so on which target.

The robot has a simple control system that operates in the absence of any scheduled deliberation, using of a simple extrapolation routine that returns the estimated position of a target at a specified time (this is just the MSE estimate from Section 5.4). There is also a procedure that determines how the gun should be moved so as to fire at the target's estimated position (at the time when the projectile reaches the target area). The motion of the target and the aim of the gun can be described in terms of linear

functions of the passage of time, so it is straightforward to solve for the firing position that takes minimum time to reach.

The procedure Find_Gun_Movement in Figure 5.2 returns the amount of time to be spent moving the gun in each direction (recall that the gun can be moved only parallel to the x or y axis). The variables $\overline{t}$, $\overline{x}$, $\overline{y}$ are the weighted means of the t, x, and y values returned by the sensor readings taken so far (Section 5.4). x' and y' are the estimated x and y components of the target's velocity. The value of $delay$ is the difference between the weighted mean $\overline{t}$ of the times at which sensor readings have been taken, and the time at which a projectile fired at the present moment would reach the target area. The value of v is the speed at which the aim of the gun can be moved along either the x or y axis.

Varying s_x and s_y, we check four cases, corresponding to the four ways in which the gun might move (e.g., to the left and up). Moving to the right ($s_x = 1$) is favored, because the first shot will be fired sooner (the targets move from right to left). If the signs of t_x and t_y are not equal to those of s_x and s_y, there is no solution (i.e., no way to make the gun meet the target) moving in that direction.

The movement and firing of the gun are controlled by the procedure Aim_Control shown in Figure 5.3, which determines the direction to move to intercept the oldest target as soon as possible. If it can determine that the target cannot be intercepted before disappearing from the target area, it moves to the next-oldest target. When the gun is pointing at the best estimate of where to shoot to hit the target, it fires. All sensor readings are taken on the target the gun is currently moving to intercept, or failing that, the oldest target in the target area. The procedure Sensor_Control (Figure 5.4) is run every time a new sensor reading can be taken and returns the target to take a reading of.

In a series of 10 trials, each consisting of a set of target appearances with a total point value of approximately 10000 (mean number of targets 5000), the average fraction of the available points gained by the robot was 0.44. We show in the rest of this chapter that the behavior of this no-deliberation control system can be improved upon.

5.6 Deliberation

We split the scheduling problem into two pieces: deciding what target to shoot at next and when, and deciding what sensor readings to take in order to maximize the probability of hitting the targets aimed at.

```
Procedure: Find_Gun_Movement(target,time,x,y)
begin
    delay := time + 40.0 - t̄
    v := 15

    s_x := 1.0
    s_y := 1.0
    t_x := (x̄ - x + x'(s_y (ȳ + y' * delay - y)/(v - s_y x') + delay) /
           (v - s_x x' - s_x s_y x' y'/(v - s_y x'))
    t_y := (ȳ - y + y'(delay + s_x t_x)) / (v - s_y y')
    if sign(t_x) = s_x and sign(t_y) = s_y then
        return t_x, t_y

    s_x := -1.0
    t_x := (x̄ - x + x'(s_y (ȳ + y' * delay - y)/(v - s_y x') + delay) /
           (v - s_x x' - s_x s_y x' y'/(v - s_y x'))
    t_y := (ȳ - y + y'(delay + s_x t_x)) / (v - s_y y')
    if sign(t_x) = s_x and sign(t_y) = s_y then
        return t_x, t_y

    s_x := 1.0
    s_y := -1.0
    t_x := (x̄ - x + x'(s_y (ȳ + y' * delay - y)/(v - s_y x') + delay) /
           (v - s_x x' - s_x s_y x' y'/(v - s_y x'))
    t_y := (ȳ - y + y'(delay + s_x t_x)) / (v - s_y y')
    if sign(t_x) = s_x and sign(t_y) = s_y then
        return t_x, t_y

    s_x := -1.0
    t_x := (x̄ - x + x'(s_y (ȳ + y' * delay - y)/(v - s_y x') + delay) /
           (v - s_x x' - s_x s_y x' y'/(v - s_y x'))
    t_y := (ȳ - y + y'(delay + s_x t_x)) / (v - s_y y')
    if sign(t_x) = s_x and sign(t_y) = s_y then
        return t_x, t_y
end
```

Figure 5.2: Figuring out how to move the gun to hit a target

```
Procedure: Aim_Control()
begin
    time := Get_Current_Time()
    x := Current_X_Aim
    y := Current_Y_Aim
    v := 15
    current_target := null
    for target in List_of_targets
        t_x, t_y := Find_Gun_Movement(target, time, x, y)
        t_hit := abs(t_x) + abs(t_y) + 40.0 + time
        if X_Estimate(target, t_hit) < 0.0
            continue
        current_target := target
        if t_x < 0
            move := left
        else if t_x > 0
            move := right
        else if t_y > 0
            move := up
        else if t_y < 0
            move := down
        else
            move := fire
    return move
end
```

Figure 5.3: No-deliberation control for moving and firing the gun

```
Procedure: Sensor_Control()
begin
    target := current_target
    if target = null
        target := first( List_of_targets)
    return target
end
```

Figure 5.4: No-deliberation sensor control

5.6.1 Target Scheduling

One kind of deliberation the robot can do is to decide which targets to go after when. This helps first in allowing the robot to decide not to pursue some targets if there are too many present, and second in allowing the robot to optimize the movement of the gun.

The no-deliberation control system generates an implicit schedule. This system goes after the oldest target in the target area that it can get to before it disappears. If several targets are on the screen, as is frequently the case, this may not be a very good ordering, since it may be possible to reorder the targets so as to minimize the time spent in moving the gun around. In addition, consider a situation in which there are several targets that all appeared at about the same time, with about the same velocity. All of these targets may appear to be reachable before they disappear, though in fact it may be possible to reach only one or two of them before they all disappear at roughly the same time. In this case, all but one or two of the targets in this group should be removed from consideration (e.g., no sensor readings should be wasted on them). Constructing an explicit schedule that takes into account the time required to get from one target to the next will allow the robot to avoid this error.

We have implemented a target scheduler that determines a schedule of targets to try to hit and fixes the maximum number of shots to be taken at each target. Constructing such a schedule involves estimating a target's trajectory in order to determine when it will disappear and when and where to shoot at it. Given only the initial sensor information for a target, we can estimate roughly when it will disappear by using an expectation on the target's x velocity calculated from the behavior of earlier targets. This information is insufficient, however, to influence significantly the robot's chances of hitting the target. For instance, 100 time units after the target appears, if the robot has not taken a sensor reading on it, the standard error of its estimate of the target's x position is about 130 and standard error of the y estimate is about 104. This gives a probability of a successful hit of about 0.02. With an increasing number of sensor readings, the robot's estimate of the target's trajectory becomes increasingly accurate, allowing more precise scheduling and increasing the probability of hitting the target shot at.

The target-scheduling algorithm in Figure 5.5 incrementally modifies an existing schedule. At each iteration, a different target is considered for insertion before the current point. Each target is considered in turn. The procedure keeps track of the expected time required to shoot each target. If any target under consideration cannot be reached before it disappears, starting from the gun's expected position after shooting at the last target, it is removed from the schedule.

This is a greedy algorithm; we add to the current prefix of the target schedule

```
Procedure: Target_Sched(ix)
begin
    start := ⟨ first( Target_schedule)⟩
    rest := Concatenate( Target_schedule − start, List_of_targets − Target_Schedule)
    while length(rest) > 0
        if ix < 1
            break
        S_curr = Concatenate(start, rest)
        V_curr := Calc_Expected_Utility(S_curr)
        V_max := V_curr
        tgt_max := first(rest)
        for tgt_2 in rest
            if ix < 1
                break
            S_new = Concatenate(start, ⟨ tgt_2 ⟩, rest − ⟨ tgt_2 ⟩)
            V_new := Calc_Expected_Utility(S_new)
            if V_new > V_max
                tgt_max := tgt_2
                V_max := V_new
            ix := ix − 1
        start := Concatenate(start, ⟨ tgt_max⟩)
        rest := rest − tgt_max
    return Concatenate(start, rest)
end
```

Figure 5.5: Generating a schedule of targets to shoot at

(*start*) the target that maximizes the expected utility of the resulting schedule. The procedure shown here is simplified by omitting the calculations for determining the maximum number of shots to be taken at each target. In practice, we set this limit at the expected number of shots required to hit each target, given the sensor readings scheduled to be taken before each shot.

The oldest part of the schedule is considered first, because time is passing as scheduling is being done. The passage of time as scheduling is performed is significant in another way: if the scheduler will not finish an iteration until after the robot shoots at the next n scheduled targets, those targets must be removed from the part of the schedule that can be rearranged. In the current implementation, this is dealt with by not allowing a substitution at the beginning of the target schedule, and by doing no deliberation at all if it is impossible to complete at least one iteration of the scheduler before the current target is expected to be hit.

5.6.2 Sensor Scheduling

For a given target schedule, the robot can allocate time to scheduling the sensor so as to maximize the expected value of the targets hit. Since the sensor is a scarce resource, we model it as a set of "slots" separated by the fixed sensor delay, each corresponding to a separate reading. Sensor scheduling involves deciding what target will get a reading in each of these slots. It has some interesting properties:

- Any delay between sensor readings and shooting at a target decreases the accuracy of the position estimate (and thus the probability of hitting it).
- Additional readings increase the accuracy of the estimate.
- More widely spaced sensor readings increase the accuracy of the estimate.

These properties suggest that effective sensor scheduling may have a substantial effect on the utility of the robot's performance.

The sensor scheduler iterates over the available sensor slots, starting with the earliest ones. For each slot, it calculates the change in expected utility if that slot were to be used for a different target. The slot is allocated to the target maximizing the resulting change in expected utility. The scheduler iterates over future sensor slots until it runs out of either deliberation time or targets to schedule readings of.

5.7 Deliberation Scheduling

Deliberation scheduling for the Shooting Gallery consists of determining when to do target scheduling and when to do sensor scheduling. The two forms of scheduling are interdependent: sensor readings provide additional information that makes target scheduling more reliable. Reliable target scheduling means that scheduled sensor readings are more likely to be useful, because the target involved will in fact be fired on at the expected time. As a first test of the effect of these two schedulers, we ran a series of trials, each consisting of a fixed set of target appearances. For each set, we noted the utility of the robot's performance (the value of the targets it hit) for four different cases: doing no deliberation, doing only target scheduling, doing only sensor scheduling, and doing both target and sensor scheduling. We ignored the cost of deliberation in these experiments; the object was to establish upper bounds on the improvement in the robot's behavior resulting from time allocated for deliberation using these two schedulers. The results are summarized in Table 5.1.

Sensor scheduling does not appear to be very effective: its improvement on the expected utility of the robot's performance is only 2%, even when we ignore the cost of deliberation and run the scheduler to completion. As we show later in this section,

Deliberation	Avg Utility
None	0.44
Sensor only	0.45
Target only	0.53
Both	0.54

Table 5.1: The benefits of deliberation

the poor performance of the sensor scheduler may be due to the fact that the robot's expectations are a poor model of the environment's behavior.

Given these results, we will not use the sensor scheduler at all. Deliberation scheduling is thus trivial: we run the target scheduler until one of three things happens:

- A new target appears.
- A target is hit.
- A target disappears without being hit.

These events modify the set of targets the robot knows about, and the target scheduler is restarted with the new information.

The benefit of target scheduling is shown in Figure 5.6. The leftmost value on the graph is the expected utility when the robot does no deliberation at all. For the rightmost, deliberation is free. In between, the cost of deliberation increases to the left. This graph has one very interesting feature: the cost of the approximations made in constructing expectations has become significant. There is a point at which the expected utility of the robot's performance is actually worse for having done some deliberation than it is for no deliberation at all. This suggests that the expectations the robot uses are not a good model for the actual behavior of the system. The expectations the robot uses involve the same simplifications as for the Robot Courier: future events are ignored and expected values, not distributions, are combined, but the cost of these approximations has become significant.

5.8 Expected Utility and Optimal Decisions

5.8.1 Expected Utility

The Shooting Gallery is the most complicated of the three examples we analyze. Using the calculations in Section 5.4, we can calculate the probability of hitting a given target by firing at a particular spot at a particular time, given a set of sensor readings the robot has taken or will take. Since we know the value of the target, this gives us the expected gain in utility. The next step is to calculate the expected value of the targets

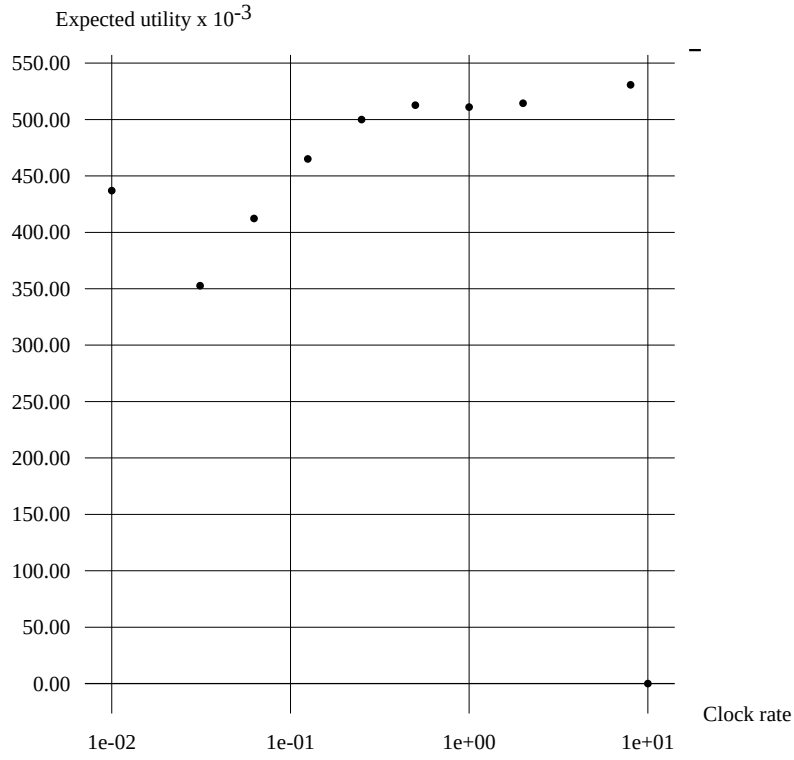

Figure 5.6: Performance profile for target scheduling

hit in a sequence of moving, firing, and sensor readings. This is more difficult: when we we fire on the second target depends on when we hit the first (unless we just fire at the first target once and move on, which seems a bad idea). This uncertainty grows with each successive target. For the current system, this calculation is done by summing the expected time required to hit each target. As shown in the previous section, this may not be a good approximation to the behavior of the system.

5.8.2 Optimal Decisions

As for time-dependent planning and the Robot Courier, optimal decision-making involving expectations on the future course of events is a hard problem in the Shooting Gallery, and as for the other two problems, the difficulty arises from two properties of the domain. First, the future occurrence of events influences the value of a decision made at the current time. These "occurrences" can include the fact that an event does *not* happen (e.g., a target does not appear) at a given time. Second, the number of different possible occurrences in the next increment of time is enormous: no target may appear, or a target may appear with any of a wide range of initial position and velocities. Each of these possible positions and velocities affects the optimal decision at the current point. Thus generating an optimal policy by dynamic programming, even for a short sequence of decisions, requires considering an enormous number of different states.

Consider the situation in Figure 5.7. Suppose that at time t, the robot is currently considering in what order it should pursue the two targets shown. Suppose further that we know that a target will appear at time $t + 1$, but not where or with what x and y velocities. It is clear that the utility of the decision made at time t may depend on where the target appears.

Calculating a policy for a sequential decision model requires reasoning about expectations from a given state of the world. The uncertainty present in this problem complicates this calculation in that all of the possible next states must be examined, and all of the possible states after the next one, and so on. If each state has a large number of possible next states (consider the range of possible initial positions and velocities for a new target), the size of the calculation required rapidly becomes impractical.

The fact that sensor readings affect deliberation scheduling introduces a cycle that complicates the utility calculations in a way we have not dealt with. The deliberation scheduler must schedule time for deliberation on the basis of the expected effects of that allocation. Among those effects are additional sensor information that can be used to do deliberation scheduling at some later point. This suggests that calculating an optimal policy may be not only computationally infeasible, but conceptually difficult as well.

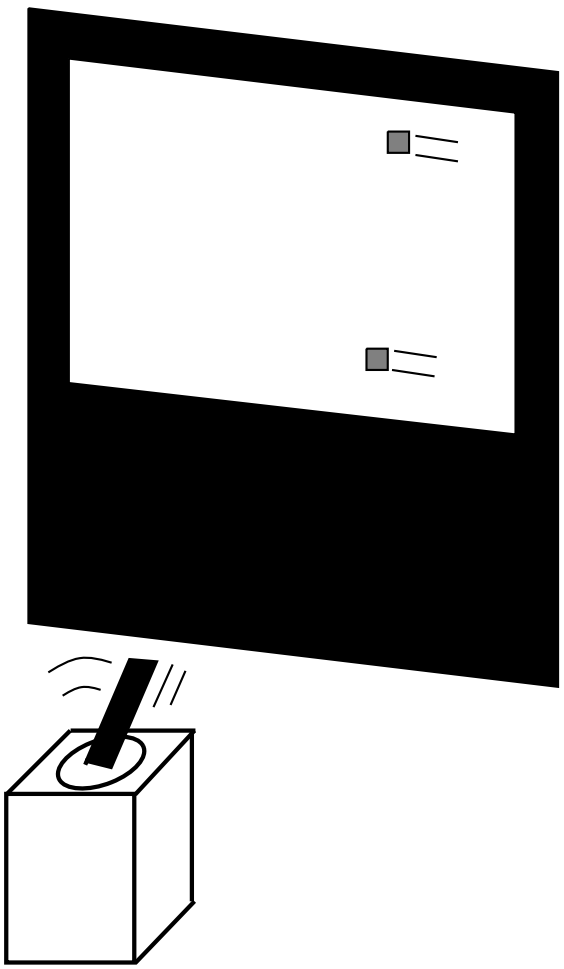

Figure 5.7: Target scheduling for the Shooting Gallery

5.9 Summary and Conclusions

The solution presented here for the Shooting Gallery is preliminary. We present a pair of algorithms for scheduling targets and sensor readings and show that, given the expectations the robot uses, sensor scheduling makes effectively no improvement in the robot's performance. The solution we present has the desired features of being sensitive to the current situation and the available resources, and serves as another demonstration that expectation-driven iterative refinement is an effective framework for generating solutions to time-dependent problems. However, it also makes it clear that in this domain, more sophisticated calculations of the expected behavior of the robot and its environment might be useful.

5.9.1 Extensions and Future Work

This is the most complicated of the three examples we analyze and the one on which the most work remains to be done. For time-dependent planning and the Robot Courier, we present exhaustively analyzed solutions. For the Shooting Gallery, on the other hand, there is considerable room for further analysis of the current problem. For example, we have done no explicit analysis of the interaction between sensor readings and target scheduling, or between sensor reading and deliberation scheduling. The fact that sensor readings affect deliberation scheduling introduces a cycle that complicates the utility calculations in a new way.

Nor is the set of anytime decision procedures we have defined clearly the "best" set, either in terms of the problems addressed or the algorithms employed. But it is not clear how a set of anytime decision procedures could be shown to be the "best" other than empirically: by doing a better job than any other proposed candidates.

Other issues that we may address in future work involve the robot's expectations on its environment. It is clear that the simplifying assumptions that worked well for the first two problems are not as useful in this domain, but how do we relax them and retain efficient deliberation scheduling? In this chapter, the robot is given the distributions that govern the appearance and velocities of targets. What if constructing these distributions was deliberation that had to be scheduled? A related question that is also of interest: what if estimating a target's trajectory was deliberation that had to be scheduled as well?

Chapter 6

Contributions and Future Work

6.1 Contributions

The initial motivation for the work described here was the desire to build a robot that was both responsive to the world around it, and capable of abstract problem solving to achieve a set of goals (i.e., capable of *planning*). This motivation guides us to look at particular kinds of time-dependent problems. In general, we are interested in problems in which the system must make tradeoffs among several competing demands on its resources, in the face of widely varying and uncertain task loads and resource availability.

The main contributions we have made to the analysis and solution of such time-dependent problems can be classed as theoretical analysis, the empirical solution of specific problems, and the generation of a methodology for solving time-dependent problems. The theoretical issues we address include finding or constructing appropriate anytime decision procedures, generating expectations on the behavior of those anytime decision procedures, and calculating the effects of deliberation on expected utility. We also suggest some ways in which the decision model can be restricted so as to simplify the design of a deliberation scheduler.

We describe in detail the process of providing solutions for three different time-dependent problems. The first of these examples is the class of time-dependent planning problems, time-dependent problemsthat involve absolute deadlines and independent responses to individual requests. The second is the Robot Courier problem, in which there are no hard deadlines and tasks can be reordered. The third is the Shooting Gallery problem, in which the robot uses sensors to gather information in order to complete a set of highly interdependent tasks with uncertain deadlines.

Generating these solutions, we develop a methodology for solving time-dependent problems: finding or generating the necessary anytime decision procedures, gathering

expectations, and framing and solving the resulting optimization problem. We also show that in the process of providing a solution, we generate data useful for exploring the tradeoff between performance and computational power, and that our solutions do exhibit the kind of "graceful degradation" we had intended in the face of increasingly scarce resources. The rest of this section explores these contributions in more detail.

6.1.1 A Framework for Solving Time-Dependent Problems

We are interested in constructing controllers for robots interacting with complex and dynamic environments. By "dynamic environment" we mean an environment whose state evolves over time as events occur or processes progress. Planning systems operating in simple simulated environments have generally done their planning "off-line" relative to the passage of time in the environment, and the environment can also be made free of agents or processes other than the planner. Robots operating in the real world, however, cannot stop the passage of time while they decide what to do. Nor can they afford to ignore the course of events around them. Any time-consuming computation the robot does is at the cost of something else it could be doing with the same time, and is also likely to be interrupted by the occurrence of some unexpected event.

Any system, no matter how large and sophisticated, has finite computational resources. The need to synchronize the system's behaviour with other events and the limited computational resources available for deliberation means that the system must trade off the potential benefits of deliberation against the cost of delaying action. If the system has more than one thing to think about at any one time, it must make tradeoffs in how much time is allocated to each task. A solution to a time-dependent problem is a strategy for determining when the system should deliberate and on what, and when it should act (i.e., a strategy for deliberation scheduling).

Expectation-driven iterative refinement gives us a framework for solving time-dependent problemsthat has several advantages. First, it simplifies the problem by allowing us to break it into smaller pieces. We define the kind of deliberation that the system will do, find or construct anytime decision procedures to do that deliberation, construct expectations on the behavior of those procedures, and finally provide a procedure for scheduling the available anytime decision procedures, on the basis of the current and expected state of the environment. Anytime decision procedures have some advantages in dynamic and uncertain environments. They allow the system to make effective use of as much time as is available, and if interrupted unexpectedly (e.g., by the occurrence of some event urgently requiring a response) still provide a reasonable answer. The use of probability and decision theory as a basis for deliberation scheduling allows us to compare one solution to another in a consistent way (on the basis of the expected utility of the resulting system behavior). It also enables us to

characterize the cost of the assumptions and simplifications we make in providing a deliberation scheduler.

6.1.2 Theoretical Results

In Chapter 2, and to a lesser extent elsewhere in the thesis, we address theoretical issues relevant to any attempt to solve time-dependent problems using expectation-driven iterative refinement. We provide a formal specification for what we mean by a time-dependent problem. This provides us with a common means of specifying different classes of time-dependent problems, making it easier to determine their similarities and differences.

We also define the form of a solution to a time-dependent problem generated using expectation-driven iterative refinement. These solutions require us to specify a set of anytime decision procedures for doing deliberation. A great number of known algorithms, capable of solving a wide variety of problems, either are already framed as anytime algorithms or can easily be adapted to be anytime algorithms. In Section 2.4.3, we show how to combine known anytime decision procedures to generate anytime decision procedures for solving more complex problems. The methods discussed in that section are local in scope and therefore limited in their application, but will suffice for simple combinations.

In Section 2.4.2, we explore the problem of gathering and using expectations on the behavior of the anytime decision procedures used for deliberation. Using expected values rather than distributions on the answers returned for a given allocation of deliberation time simplifies the calculation of expected utility (and thus deliberation scheduling). However, the resulting system behavior may have a lower expected utility than if we had used the distributions. Another issue to be dealt with is that the parameter of the answer returned that is considered most significant will change with use to which the answer is put.[1] The form of the performance profile used for deliberation scheduling may need to change not just from problem to problem, but perhaps from situation to situation.

We show that constructing a deliberation-scheduling policy involves solving a sequential decision problem, and that providing an optimal deliberation-scheduling policy for a reasonably complex decision model may be very difficult. As an alternative strategy, we suggest some ways in which the decision model can be restricted so as to simplify the problem of providing a deliberation-scheduling policy. In particular we discuss eliminating expectations on the occurrence of events and generating an explicit deliberation schedule on the occurrence of a restricted set of significant events. For

[1] Horvitz [1988] calls this property *multiple attributes of utility*.

some problems (e.g., time-dependent planning problems and the Robot Courier), this approach can provide computationally inexpensive deliberation scheduling for which the expected utility is only slightly less than for deliberation scheduling using a more complete (but completely intractable) decision model.

6.1.3 Empirical Results

Time-dependent planning problems involve absolute deadlines and independent responses to individual requests. For the static case, in which the decision model does not include expectations on future events, we provide an optimal deliberation-scheduling policy, implemented by a deliberation scheduler that constructs an explicit deliberation schedule and then makes allocations according to that schedule as time passes. We show that the deliberation scheduler used in the static case can be applied to the dynamic problem, in which new observations occur as time passes, with only a slightly lower expected utility than an optimal "perfect-information" deliberation scheduler that knows the future occurrence of events. We also show that providing an optimal deliberation-scheduling policy for a decision model that includes expectations on the future occurrence of events may be computationally difficult, in that the number of states for which a policy must be determined may be very large.

In Chapter 4, we present the Robot Courier problem, in which there are no hard deadlines and tasks can be reordered. For this class of problems, utility is defined strictly in terms of how long the robot takes to complete a set of tasks. We define appropriate anytime decision procedures for deliberation and construct performance profiles to provide estimates of the effect of allocations to these anytime decision procedures on the robot's performance. We show that for the Robot Courier, just as for time-dependent planning, providing an optimal deliberation-scheduling policy for any decision model that takes into account the dynamic and uncertain nature of the problem is likely to be very difficult.

For any reasonably complex deliberation scheduling problem (we view the Robot Courier as a relatively simple problem), it seems likely that similar difficulties will arise in defining or calculating the "optimal" deliberation-scheduling policy. We deal with this difficulty empirically: by generating optimal or near-optimal deliberation-scheduling policies for restricted decision models and then testing those policies on real or simulated instances of the problem. For the Robot Courier, we provide deliberation schedulers for three increasingly complex decision models, all of which use expectations rather than distributions for performance profiles and ignore the possibility that additional requests (tasks) may occur. These deliberation schedulers do not perform as well on dynamic problems (in which requests arrive as time passes) as they do on static problems (in which the requests are all made at time 0). But the average loss in

performance in dynamic situations is fairly small.

In the process of comparing the various deliberation schedulers, we generate data of a sort that is likely to be very useful in the design of any time-dependent controller. We examine the change in performance as the relative cost of deliberation changes. This makes clear the tradeoff between performance and computational power (which costs money, or battery power, or has some other form of opportunity cost). The other point that becomes clear in this analysis is that the system does exhibit the kind of "graceful degradation" we had intended. There is no point at which the resources available fall below some critical level and the system simply quits functioning. These results are especially encouraging because this is the first empirical demonstration that expectation-driven iterative refinement is a useful framework for analyzing and solving time-dependent problems.

The Shooting Gallery is the most complex of the problems we investigate. It is designed to have a specific set of characteristics, including requiring the use of expensive and noisy sensors to gather information, interaction between several possible kinds of deliberation (target scheduling and sensor scheduling, for the Shooting Gallery), and a set of separate but interdependent tasks to be accomplished by uncertain deadlines. These characteristics create a tradeoff among deliberation, acting, and gathering information. We do not have a final answer on how that tradeoff should be made. We do provide a solution that makes good use of the time available and improves the robot's performance substantially. One interesting feature of the solution we provide is that the cost of the simplifications made in the decision model used for deliberation scheduling is high enough that there are situations where the expected utility of the robot's behavior is highest doing no deliberation at all, even if time is available.

6.1.4 Towards a Methodology for Generating Solutions

A time-dependent problem is specified in terms of domain characteristics (what kinds of events happen, what kinds of actions the system can take, and a utility function on the resulting sequence of events). We then determine the anytime decision procedures to be used to perform the required computation. We gather expectations on the behavior of those anytime decision procedures and how that behavior affects the expected utility of the resulting system behavior. We use expectations on the future course of events and the answers returned by the anytime decision procedures, and seek to maximize the expected utility of the resulting system behavior. A solution to a time-dependent problem is a deliberation scheduler that provides a (perhaps approximate) solution to this optimization problem at run-time.

Several lessons become clear from our work on how to go about solving a time-dependent planning problem. For one, we do not "solve" any planning problem for any

of the three examples discussed in this thesis. Rather, we provide approximate answers and some measure of the errors to be expected. An interesting point that arises in the generation of performance profiles is that we need a fast estimate of how good an answer is obtainable. This estimation will of necessity introduce some error and we should know how much. The form of the solution we provide is strongly determined by the structure of the environment within which the robot operates. The environment determines the performance profiles of the anytime algorithms the robot employs for planning, and to a lesser extent the algorithms themselves. For even the simplest of our three time-dependent problems, providing an optimal deliberation scheduler for a decision model that includes expectations on the occurrence of events appears to be very difficult. For all three problems, we ended up simplifying the decision model and investigating the resulting loss in expected utility.

In practice, solving even a moderately complex time-dependent planning problem requires combining the results of several deliberation procedures. In some problems, several unrelated subproblems may competing for deliberation time. For the Robot Courier in Chapter 4, this would correspond to allocating planning time for the individual paths on a tour. In addition, the system is likely to have several anytime decision procedures to which it can allocate time, and the results of these anytime decision procedures are not necessarily independent. In order to allocate deliberation time for tour improvement (reorder the locations in a tour), the Robot Courier needs to take into account the effects of both path planning and tour improvement. The overall time required to complete the improved tour depends on the length of the tour and also on the results of planning paths from one location to another. For the Shooting Gallery (Chapter 5), target scheduling and sensor scheduling interact in complex ways. Sensor scheduling determines the information that will be available for target scheduling. Target scheduling determines which targets should get additional sensor readings. This interdependence among anytime decision procedures further complicates the problem of calculating expected utility from performance profiles, in addition to the issues discussed in Section 2.5.3.

6.2 Further Work

Providing solutions to time-dependent problems of the sort addressed here (as opposed to the kinds of problems control theory treats, for example) is a comparatively untouched research area. For the most part, we have brought together techniques from other fields, rather than building on existing work addressing the same problems. There are certainly previous and parallel efforts that address similar problems (see Section 1.3): the field is large enough, and these efforts few enough, that the overlap

among them and with our work is limited.

A unified view of time-dependent problems would make comparing and combining the results of this related work easier. The formal specification in Chapter 2 is directed to this end, as is Russell's definition of meta-reasoning [1989b]. These two views of time-dependent problems are distinct and may not be easy to reconcile. Russell's formalism makes representing deadlines difficult and models deliberation in terms of discrete chunks, whereas our work emphasizes on events and their temporal relationships (including deadlines of various kinds), and views deliberation time as being allocated in values drawn from a continuous range.

There are some indications that a truly general framework for stating and solving time-dependent problems may be hard to find. Already entire disciplines are devoted to solving particular classes of time-dependent problems (e.g., control theory, real-time systems, and job-shop scheduling). In addition, consider what has happened in the work on scheduling: there are a myriad of separate problems, distinguished by minor differences in problem characteristics, for which the difficulty of providing a scheduler (and the best way of doing so) varies enormously [King and Spachis, 1980]. The most general notion of deliberation scheduling subsumes the scheduling problem, and thus will be no easier to provide a general solution for.[2]

We deal with the multifarious and diverse nature of time-dependent problems by identifying the kinds of time-dependent problems we wish to address. We characterize the problems we are interested in, first by specifying the circumstances in which they may arise (controlling robots in complex environments), and also by defining a language whose expressive limitations circumscribe the problems we will be concerned with. In the rest of this section, we discuss some possible extensions to the work presented here, followed by some more general issues related to the class of time-dependent problems we are studying.

The work described here could be extended in several directions. We divide them roughly into further investigation of theoretical issues, and possible extesions of our experimental results for particular problems.

6.2.1 Theoretical Issues

Several theoretical issues touched on here could benefit from further investigation. For example, the process of providing solutions to our example problems has consisted in large part of formulating an appropriate decision model for deliberation schedulingand then providing a deliberation scheduler that implements a corresponding policy. Many issues arise in the construction of these decision models. First and foremost, what

[2]This is one of the advantages of using anytime decision procedures: the resulting scheduling problems are more likely to be easy to solve.

kinds of simplifications can we make, and at what cost? We need to model such things as Henrion's expected value of including uncertainty [Henrion, 1984], the value of gathering additional information, and the cost of excluding some kinds of expectations from the decision model (e.g., expectations on future events). In addition, if the system must generate its own performance profiles and expectations on the future occurrence of events, that computation will probably qualify as deliberation that must be scheduled. Then we must include in our utility calculations the effects of time spent refining the expectations used in deliberation scheduling.

One might also investigate the tradeoff between run-time and compile-time or design-time computation. For time-dependent problems, there are actually two such tradeoffs to be made. The first is the reduction of run-time deliberation by compile-time computation. To make this tradeoff in a principled way, we need to be able to model the costs and benefits of reducing the run-time decision-making capabilities of the system [Heckerman *et al.*, 1989, Russell, 1989, Kanazawa and Dean, 1989]. The second tradeoff is the reduction of run-time computation for deliberation scheduling in the same way. These two problems are interdependent: the benefit of modifying the decision procedures used for deliberation depends on the expected utility resulting from using a particular deliberation scheduling strategy, while the kind of deliberation scheduler that is appropriate depends on the behavior of the anytime decision procedures available.

An issue we have mentioned but not explored is the use of deliberation schedulers that are computationally expensive. In this thesis, we have dealt with the issue of taking into account the time required for deliberation scheduling by ensuring that our deliberation schedulers ran fast, and looking at the cost of that speed in terms of the expected utility of the resulting behavior. But this is only one way to address the problem, and it is by no means certain to be the best way for any given time-dependent problem. We have suggested two other approaches: running a relatively expensive deliberation scheduler infrequently, and explicitly scheduling deliberation scheduling time, using an anytime deliberation scheduler and expectations on the effects of time allocated for deliberation scheduling. What kind of decision model should we use for scheduling deliberation scheduling? What aspects of the problem should we leave out, so as to make this "meta-meta-reasoning" efficient?

6.2.2 Extending the Examples

The three examples for which we have provided solutions could be extended in many ways.[3] Which of these possible extensions we find interesting is most likely to be

[3] For a more detailed description of possible extensions, see the concluding sections of the appropriate chapters.

determined either by a particular application we wish to address or by some theoretical point we wish to illuminate.

There is a wide range of possible applications that are similar to the Robot Courier. Any problem involving tasks arriving over time, where throughput is the main consideration and there is some possibility of reordering the tasks in hand should have similar characteristics (e.g., disk or task scheduling for an operating system). Several interesting generalizations could be made. In particular:

- Allow the utility function to include costs on missing deadlines for reaching specified locations (the Domino's Pizza delivery model).

- Allow fixed orderings on pairs of locations (suppose the robot picks things up are to be dropped off somewhere else).

- Allow locations to have different payoffs and allow the robot to put off visiting any particular location as long as it wants. A further complication would be to make the payoff depend on when the robot reached the location.

The solution we have presented for the Shooting Gallery could be extended in a number of ways, including making better use of the available information for deliberation scheduling, tuning the procedures used for deliberation, and doing a better job of incorporating the effects of further sensor readings into deliberation scheduling.

6.3 Open problems

There are some interesting directions for future research starting with the approach developed here. One obvious possibility is to find an application and see how well the approach holds up. This thesis is a proof-of-concept for expectation-driven iterative refinement: we have demonstrated that expectation-driven iterative refinement can be employed to provide solutions for moderately complex time-dependent problems. The next step is to try it on a real application. Some recent work may simplify this process considerably. Several people have recently proposed planning or scheduling methods explicitly designed for implementation as anytime decision procedures[Elkan, 1990, Drummond and Bresina, 1990, Zweben *et al.*, 1990]. The enormous advantage to these methods is that they can be used directly in a solution for a time-dependent problem constructed using expectation-driven iterative refinement. All we need to do is gather some statistics on their performance in a particular domain.

Another issue that may arise when this framework is applied to more complex problems is representing and reasoning with temporal information. Planning inevitably

involves reasoning about the occurrence of events and the passage of time. In a dynamic environment known only through a set of sensor readings, some of the information available is uncertain. Temporal reasoning with uncertain information is a hard problem. The most common responses to this fact have been either to limit the expressivity of the language [Berzuini *et al.*, 1989, Dean and Kanazawa, 1988, Haddawy, 1990] or to use the structure of the application being addressed to constrain the inference performed [Wellman, 1988, Hanks, 1990]. It would be interesting to extend the examples analyzed in this thesis to include uncertain information and to see what combination of these approaches might prove effective. An additional issue that would almost certainly have to be dealt with in doing this is the explicit allocation of deliberation time for projection.

Some other possible directions for further work include designing a system that learns policies for deliberation scheduling (using an approach similar to [Barto *et al.*, 1989]), automating the calculation of expected utility from performance profiles, as in Section 2.5.3, and extending the start we have made on providing a general framework for constructing complex anytime decision procedures from simpler ones (Section 2.4.3).

Bibliography

[Agogino *et al.*, 1988] A. M. Agogino, S. Srinivas, and K. Schneider. Multiple sensor expert system for diagnostic reasoning, monitoring, and control of mechanical systems. *Mechanical Systems and Signal Processing*, 2(2):165–185, 1988.

[Andersson, 1988] Russell L. Andersson. *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control.* MIT Press, Cambridge, Massachusetts, 1988.

[Barto *et al.*, 1989] Andrew G. Barto, R.S. Sutton, and C.J.C.H Watkins. Learning and sequential decision making. Technical Report 89-95, University of Massachusetts at Amherst Department of Computer and Information Science, 1989.

[Bellman, 1957] R.E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[Berzuini *et al.*, 1989] C. Berzuini, R. Bellazzi, and S. Quaglini. Temporal reasoning with probabilities. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 14–21, 1989.

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *IJCAI89*, 1989.

[Bollinger and Duffie, 1988] John G. Bollinger and Neil A. Duffie. *Computer Control of Machines and Processes.* Addison-Wesley, Reading, Massachusetts, 1988.

[Breese and Fehling, 1988] John S. Breese and Michael R. Fehling. Decision-theoretic control of problem solving: Principles and architecture. In *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, 1988.

[Breese and Horvitz, 1990] J. S. Breese and E. J. Horvitz. Ideal reformulation of belief networks. In *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, pages 64–72, 1990.

[Breese, 1990] J. S. Breese. Construction of belief and decision networks. Technical Report 30, Rockwell International Science Center, 1990.

[Brooks, 1985] Rodney A. Brooks. A robust layered control system for a mobile robot. A.I. Memo No. 864, MIT Artificial Intelligence Laboratory, 1985.

[Carraway *et al.*, 1989] R. L. Carraway, T. L. Morin, and H. Moskowitz. Generalized dynamic programming for stochastic combinatorial optimization. *Operations Research*, 37:819–829, 1989.

[Chapman and Agre, 1987] David Chapman and Phil Agre. Abstract reasoning as emergent from concrete activity. In Michael P. Georgeff and Amy L. Lansky, editors, *The 1986 Workshop on Reasoning About Actions and Plans*, pages 411–424. Morgan-Kaufman, 1987.

[Cohen *et al.*, 1988] P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 9(1):34–48, 1988.

[Cooper, 1987] Gregory F. Cooper. Probabilistic inference using belief networks is np-hard. Memo KSL-87-27, Stanford Knowledge Systems Laboratory, 1987.

[Dantzig, 1963] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-88*, pages 49–54. AAAI, 1988.

[Dean and Kanazawa, 1988] Thomas Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *Proceedings AAAI-88*, pages 524–528. AAAI, 1988.

[Dean and Siegle, 1990] T. Dean and G. Siegle. An approach to reasoning about continuous change for applications in planning. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 132–137, 1990.

[Dean *et al.*, 1987] Thomas L. Dean, R. James Firby, and David P. Miller. The forbin paper. Technical Report 576, Yale University Department of Computer Science, 1987.

[Dean, 1989] Thomas Dean. Decision-theoretic control of inference for time-critical applications. Technical Report CS-89-44, Brown University Department of Computer Science, 1989.

[Dorf, 1989] Richard C. Dorf. *Modern Control Systems.* Addison-Wesley, 1989.

[Drummond and Bresina, 1990] Mark Drummond and John Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 138–144, 1990.

[Einav, 1990] David Einav. Computationally-optimal real-resource strategies for independent, uninterruptible methods. In *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, pages 73–81, 1990.

[Elkan, 1990] Charles Elkan. Incremental, approximate planning. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 145–150, 1990.

[Etzioni, 1989] Oren Etzioni. Tractable decision-analytic control. In *First International Conference on Principles of Knowledge Representation and Reasoning*, pages 114–125. Morgan-Kaufmann, 1989.

[Farmer *et al.*, 1984] D. Farmer, T. Toffoli, and S. Wolfram. *Cellular Automata.* North Holland, 1984.

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

[Firby, 1987] R. James Firby. An investigation in reactive planning in complex domains. In *Proceedings AAAI-87*, pages 196–201. AAAI, 1987.

[Georgeff and Ingrand, 1989] Michael P. Georgeff and Francois F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978. IJCAI, 1989.

[Georgeff *et al.*, 1987] M. P. Georgeff, A. L. Lansky, and M. J. Schoppers. Reasoning and planning in dynamic domains: an experiment with a mobile robot. Technical Report 380, SRI International, 1987.

[Good, 1976] I. J. Good. *Good Thinking.* University of Minnesota Press, 1976.

[Green and Margerison, 1978] J. R. Green and D. Margerison. *Statistical Treatment of Experimental Data.* Elsevier, New York, NY, 1978.

[Green, 1969] C. C. Green. Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1969.

[Haddawy, 1990] Peter Haddawy. Time, chance, and action. In *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, pages 147–154, 1990.

[Hageman and Young, 1981] L.A. Hageman and D.M. Young. *Applied Iterative Methods.* Academic Press, 1981.

[Hager, 1988] Gregory D. Hager. *Active Reduction of Uncertainty in Multi-Sensor Systems.* PhD thesis, University of Pennsylvania, July 1988.

[Hanks and Firby, 1990] Steve Hanks and R. James Firby. Issues and architectures for planning and execution. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.* DARPA, 1990.

[Hanks, 1990] Steven Hanks. *Projecting Plans for Uncertain Worlds.* PhD thesis, Yale University, January 1990.

[Hansson and Mayer, 1988] Othar Hansson and Andrew Mayer. The optimality of satisficing solutions. In *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, 1988.

[Harel, 1987] David Harel. *ALGORITHMICS: The Spirit of Computing.* Addison-Wesley, 1987.

[Hayes-Roth *et al.*, 1989] Barbara Hayes-Roth, Richard Washington, Rattikorn Hewett, Michael Hewett, and Adam Seiver. Intelligent monitoring and control. In *Proceedings IJCAI 11*, pages 243–249. IJCAI, 1989.

[Heckerman *et al.*, 1989] David E. Heckerman, John S. Breese, and Eric J. Horvitz. The compilation of decision models. In *UW89*, pages 162–173, 1989.

[Heckerman, 1985] David Heckerman. Probabilistic interpretations for mycin certainty factors. In *Proceedings of the 1985 AAAI/IEEE Sponsored Workshop on Uncertainty and Probability in Artificial Intelligence*, 1985.

[Hendrix, 1973] Gary Hendrix. Modeling simultaneous actions and continuous processes. *Artificial Intelligence*, 4:145–180, 1973.

[Henrion, 1984] M. Henrion. *The Value of Knowing How Little You Know.* PhD thesis, Carnegie Mellon University, 1984.

[Henrion, 1986] M. Henrion. Propagating uncertainty by logic sampling in bayes' networks. In *Proceedings of the Second Workshop on Uncertainty in Artificial Intelligence*, 1986.

[Hinton and Sejnowski, 1983] G.E. Hinton and T.J. Sejnowski. Optimal perceptual inference. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 448–453, 1983.

[Horvitz *et al.*, 1988] Eric J. Horvitz, John S. Breese, and Max Henrion. Decision theory in expert systems and artificial intelligence. *Journal of Approximate Reasoning*, 1988.

[Horvitz *et al.*, 1989] E. J. Horvitz, H. J. Suermondt, and G. F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, 1989.

[Horvitz, 1987] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, 1987.

[Horvitz, 1988] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings AAAI-88*, pages 111–116. AAAI, 1988.

[Howe *et al.*, 1990] A. E. Howe, D. M. Hart, and P. R. Cohen. Addressing real-time constraints in the design of autonomous agents. Technical Report 90-06, Department of Computer and Information Science, University of Massachusetts, 1990.

[Kaelbling, 1988] Leslie P. Kaelbling. Goals as parallel program specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 60–65, 1988.

[Kanazawa and Dean, 1989] K. Kanazawa and T. Dean. A model for projection and action. In *Proceedings IJCAI 11*, pages 985–990. IJCAI, 1989.

[King and Spachis, 1980] John R. King and Alexander S. Spachis. Scheduling: Bibliography and review. *International Journal of Physical Distribution and Materials Management*, 10(3):105–132, 1980.

[Korf, 1987] Richard E. Korf. Real-time path planning. In *Proceedings DARPA Knowledge-Based Planning Workshop*, 1987.

[Korf, 1990] Richard Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2):189–212, 1990.

[Kraus *et al.*, 1990] S. Kraus, M. Nirkhe, and D. Perlis. Planning and acting in deadline situations. In *Proceedings of the 1990 Workshop on Planning in Complex Domains*. AAAI, 1990.

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

[Lawler *et al.*, 1985] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Travelling Salesman Problem.* Wiley, New York, NY, 1985.

[Lesser and Corkill, 1983] V.R. Lesser and D.D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4:15–33, 1983.

[Lesser *et al.*, 1988] V.R. Lesser, J. Pavlin, and E. Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9:49–62, 1988.

[Levitt *et al.*, 1988] Tod Levitt, Thomas Binford, Gil Ettinger, and Patrice Gelband. Utility-based control for computer vision. In *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence*, 1988.

[Lin and Kernighan, 1973] S. Lin and B. W. Kernighan. An effective heuristic for the travelling salesman problem. *Operations Research*, 21:498–516, 1973.

[Locke, 1986] C. Douglass Locke. *Best-Effort Decision-Making for Real-Time Scheduling.* PhD thesis, Carnegie-Mellon University, May 1986.

[Lumelsky and Stepanov, 1987] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 1987.

[Manohar, 1988] S. Manohar. Supercomputing with vlsi. Technical Report CS-88-14, Brown University Department of Computer Science, 1988.

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

[Mead and Conway, 1980] C.A. Mead and M.A. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[Mok, 1989] Aloysius P. Mok. Formal analysis of real-time equational rule-based systems. In *Proceedings of the Real-Time Systems Symposium.* IEEE, 1989.

[Nilsson, 1980] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, 1980.

[Papadimitriou and Steiglitz, 1981] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization.* Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[Pearl, 1985] Judea Pearl. *Heuristics.* Addison-Wesley, 1985.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufman, Los Altos, California, 1988.

[Raiffa, 1968] Howard Raiffa. *Decision Analysis: Introductory Lectures on Choices Under Uncertainty*. Addison-Wesley, Reading, Massachusetts, 1968.

[Ramadge and Wonham, 1989] Peter Ramadge and Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[Ramamritham *et al.*, 1987] Krithi Ramamritham, John A. Stankovic, and Wei Zhao. Meta-level control in distributed real-time systems. In *International Conference on Distributed Computer Systems*, 1987.

[Ramamurthi and Agogino, 1988] K. Ramamurthi and A. M. Agogino. Real time expert system for fault-tolerant supervisory control. In *1988 ASME International Computers in Engineering Conference*, pages 333–340, 1988.

[Robert, 1986] F. Robert. *Discrete Iterations: A Metric Study*. Springer-Verlag, 1986.

[Rosenschein and Kaelbling, 1987] Stan Rosenschein and Leslie Pack Kaelbling. The synthesis of digital machines with provable epistemic properties. In Joseph Y. Halpern, editor, *Theoretical Aspects of Reasoning About Knowledge, Proceedings of the 1986 Conference*, pages 83–98. Morgan-Kaufman, 1987.

[Ross, 1983] Sheldon Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, New York, NY, 1983.

[Russell and Wefald, 1989a] Stuart J. Russell and Eric H. Wefald. On optimal game-tree search using rational meta-reasoning. In *Proceedings IJCAI 11*, pages 334–340. IJCAI, 1989.

[Russell and Wefald, 1989b] Stuart J. Russell and Eric H. Wefald. Principles of metareasoning. In *First International Conference on Principles of Knowledge Representation and Reasoning*. Morgan-Kaufmann, 1989.

[Russell, 1989] Stuart J. Russell. Execution architectures and compilation. In *Proceedings IJCAI 11*, pages 15–20. IJCAI, 1989.

[Russell, 1990] Stuart Russell. Fine-grained decision-theoretic search control. In *Proceedings of the Sixth Workshop on Uncertainty in Artificial Intelligence*, pages 436–442, 1990.

[Sacerdoti, 1977] Earl Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier Publishing Company, Inc., 1977.

[Shachter, 1986] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, November/December 1986.

[Shekhar and Dutta, 1989] S. Shekhar and S. Dutta. Minimizing response times in real time planning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978. IJCAI, 1989.

[Shih *et al.*, 1989] W-K. Shih, J. W. S. Liu, and J-Y. Chung. Fast algorithms for scheduling imprecise computations. In *Proceedings of the Real-Time Systems Symposium*. IEEE, 1989.

[Simon and Kadane, 1975] H. A. Simon and J. B. Kadane. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.

[Smith, 1989] David E. Smith. Controlling backward inference. *Artificial Intelligence*, 39:145–208, 1989.

[Sproull, 1977] Robert F. Sproull. Strategy construction using a synthesis of heuristic and decision-theoretic methods. Technical Report CSL-77-2, Xerox PARC, 1977.

[Stankovic, 1988] John A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 18:10–18, 1988.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings IJCAI 5*. IJCAI, 1977.

[Tompkins and Wilson, 1969] C.B. Tompkins and W.L. Wilson. *Elementary Numerical Analysis*. Prentice-Hall, 1969.

[Treleaven *et al.*, 1982] Philip Treleaven, David Brownbridge, and Richard Hopkins. Data-driven and demand-driven computer architecture. *Computing Surveys*, 14(1):93–143, 1982.

[Varga, 1962] R. S. Varga. *Matrix Iterative Methods*. Prentice-Hall, Englewood Cliffs, NJ, 1962.

[Vere, 1983] Steven Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:246–267, 1983.

[von Neumann and Morgenstern, 1947] John von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, Princeton, 2nd edition, 1947.

[Vrbsky *et al.*, 1990] S. V. Vrbsky, J. W. S. Liu, and K. P. Smith. An object-oriented query processor that returns monotonically improving approximate answers. Technical Report UIUCDCS-R-90-1568, Department of Computer Science, UIUC, 1990.

[Wellman, 1988] Michael P. Wellman. Formulation of tradeoffs in planning under uncertainty. Technical Report MIT/LCS/TR-427, MIT AI Laboratory, 1988.

[Wilkins, 1984] David Wilkins. Domain independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269–302, 1984.

[Wilkins, 1989] David E. Wilkins. Can ai planners solve practical problems? Technical Report TR-468, SRI International, 1989.

[Zweben *et al.*, 1990] M. Zweben, M. Deale, and R. Gargan. Anytime rescheduling. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.* DARPA, 1990.