

Anticipating Computational Demands when Solving Time-Critical Decision-Making Problems

Lloyd Greenwald, *Brown University, Providence, RI, USA*

Thomas Dean, *Brown University, Providence, RI, USA*

An agent embedded in a dynamic environment may need to respond in a timely manner to sequences of events outside the agent's control. By anticipating computational demands and allocating processing time accordingly the agent can avoid costly delays often arising from trying to respond to a dynamic environment with high-complexity decision procedures. Deliberation scheduling, the process of allocating processing time among competing decision procedures to explicitly account for the costs and benefits of computational delays, may aid an agent that must solve time-critical decision-making problems in which the time spent in decision-making affects the quality of the responses generated. The more accurate the agent is in anticipating the computational demands of forthcoming problems the more successful it can be in allocating its decision-making time. We present an approach to solving time-critical decision-making problems by taking advantage of domain structure to expand the amount of time available for processing difficult combinatorial tasks. Our approach uses predictable variability in anticipated computational demands to allocate on-line deliberation time and exploits problem regularity and stochastic models of environmental dynamics to restrict attention to small subsets of the state space. This approach demonstrates how slow, high-level systems (e.g. for planning and scheduling) might interact with faster, more reactive systems (e.g. for real-time execution and monitoring) and enables us to generate timely solutions to difficult combinatorial planning and scheduling problems such as the traffic control of multiple robot vehicles.

1 Introduction

This paper is concerned with anticipating computational demands to assist in allocating processing time to decision-making procedures in time-critical applications. The mechanisms for anticipating computational

demands are discussed in Section 2 and a particular approach to time-critical decision making called *response planning* is presented in Section 3. This approach focuses on solving time-critical decision-making problems in dynamic environments with large state spaces and resource limitations. Section 4 relates our response planning approach to the general idea of anticipating computational demands. In Section 5 we describe an example traffic-control problem for multiple robot vehicles that demonstrates the applicability of the response planning approach to solving time-critical problems. A general overview of this work motivated by an air traffic control problem is found in [10].

2 Anticipating Computational Demands

We are interested in the design of systems that make the best use of the time available for decision-making by explicitly accounting for the costs and benefits of computational delays. Applications such as multiple-vehicle traffic control involve solving high-complexity planning and scheduling problems in dynamic environments. These applications have time-critical properties that require timely responses to events outside the planner's control in order to satisfy hard or soft deadlines and to avoid costly delays.

The computational demand of a high-complexity planning and scheduling problem is the amount of processing time required to reach a given level of performance for a particular decision procedure applied to a particular problem instance. Each new state of the environment may be considered a unique problem instance. Alternatively, a group of states may be jointly considered as a problem instance. Dynamic events cause changes to the state of the environment and, thus, introduce new problem instances to be solved. An agent embedded in a dynamic environment may

need to respond to a sequence of problem instances and must determine how to best allocate its decision-making time for varying decision procedures and problem instances. The more accurate the agent is in anticipating the computational demands of forthcoming problems the more successful it can be in allocating its scarce decision-making time.

The agent can anticipate computational demands by predicting the problem instances for a forthcoming time period and determining the expected performance of different decision procedures given varying processing allocations. These expectations are typically derived from past performance of the decision procedures applied to similar problem instances. Anticipation requires domain knowledge and the ability to make predictions about forthcoming problem instances based on prior environmental states. No anticipation is possible in a completely nondeterministic environment or without knowledge of the domain. In some cases precursor events may give advance notice of upcoming states or domain regularity may deterministically restrict the possible sequences of upcoming events or states.

By anticipating computational demands and allocating processing time accordingly, the agent can avoid costly delays often arising from trying to respond to a dynamic environment with high-complexity decision procedures. The agent does so by initiating decision procedures prior to the occurrence of a problem instance, based on prediction and anticipated computational demands. The process of allocating processing time among competing decision procedures to explicitly account for the costs and benefits of computational delays is referred to as *deliberation scheduling*. Deliberation scheduling is a resource allocation problem where the resource is limited processing time. Deliberation scheduling aids an agent that must solve time-critical decision-making problems in which the time spent in decision-making affects the quality of the responses generated.

The task of allocating decision-making time is simplified by the use of decision procedures that can be interrupted at any time to return a solution whose quality improves with more computation time. These interruptible algorithms are often referred to as *anytime algorithms* because they can output some result, of varying quality, at any time. The flexibility inherent in these decision procedures allows for a more flexi-

ble allocation of processing time to satisfy an overall performance criterion. Anytime algorithms are sometimes referred to as *flexible computations* [11] or *imprecise computations* [20]. Zilberstein [22] contains a nice comparison of related work in anytime algorithms.

An anytime algorithm may also take the form of a *contract algorithm*. A non-interruptible contract algorithm associates a specific processing time allocation (contract time) with any given performance level. As opposed to an interruptible algorithm, a contract algorithm does not guarantee results at any time and may, in fact, not return anything for processing times less than the allocated contract time and may not achieve improved performance for actual processing time greater than the allocated contract time.

The relationship of decision procedure performance (output quality) to allocated processing time is one special case of meta-level knowledge that can be represented by a *performance profile*. In general, these profiles capture the anticipated computational demands of an algorithm under differing problem parameters and inputs. One particularly relevant parameter is the desired level of performance (output quality). Additionally, the anticipated computational demands of an algorithm may be related to both the quality level of its input and the desired performance level of its output. This is represented by a *conditional performance profile*. Conditional performance profiles may be generalized to encompass other problem parameters such as resource limitations and time window of applicability of decision procedure output. Deliberation scheduling is a special case of the more general problem of composing anytime algorithms with conditional performance profiles to satisfy overall performance measures. This is the problem of *compiling* anytime algorithms introduced by Zilberstein [22].

2.1 Performance Profiles

Anticipated computational demands for a given problem and decision procedure are captured in the form of a performance profile. In the standard usage introduced by [4, 1] a performance profile is a mapping from processing time to expected performance (quality) for a decision procedure, derived from past experience. Performance profiles are used to allocate computational resources among anticipated problem instances and competing decision procedures. In this section we gener-

Anticipating Computational Demands

alize the performance profile notion and extend some of the discussions of [22]. In Section 2.2 we expand upon methods for conditioning profiles based on differing problem parameters and inputs.

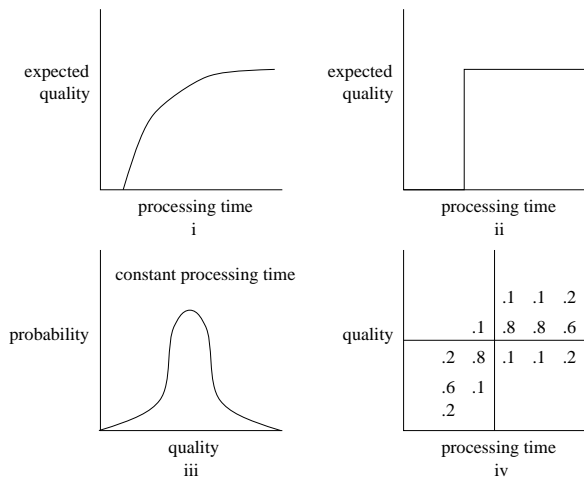


Figure 1: Basic Performance Profiles

A typical performance profile for an anytime algorithm is depicted in Figure 1.i. This figure indicates that the expected quality of the decision procedure summarized by this profile increases with diminishing returns (decreasing slope) as processing time increases. Another way to look at this profile is that the anticipated computational demands (processing time) increase as desired level of performance increases. Compare this to a typical performance profile for a traditional algorithm depicted in Figure 1.ii. The expected output quality of this decision procedure takes on exactly two values. It is zero until some threshold processing time and then a constant value thereafter.

By *expected quality* we are referring to a statistical measure of the output quality drawn from past experience with the decision procedure over varying problem instances. This corresponds to the actual observed quality only for deterministic algorithms in which quality is dependent only on processing time. However for nondeterministic algorithms and those for which quality may vary with the particular problem instance, actual observed quality may deviate from this expected value.

The variance in observed algorithm quality for varying problem instances can be interpreted in two ways.

For any constant desired quality there may be a probability distribution over processing times for varying problem instances. Similarly, for any constant allocation of processing time there may be a distribution over observed output quality for varying problem instances. A typical distribution is depicted in Figure 1.iii. We can capture both these interpretations in a three-dimensional performance profile in which probability is the third dimension. Zilberstein refers to this representation as a *performance distribution profile*. For discrete processing times and output qualities we can use a tabular representation like Figures 1.iv where the values in the table represent the probability that that particular quality and processing time will occur jointly. The granularity of the table is a tradeoff between accuracy and space requirements. Each column must sum to one.

The actual interpretation of the quantity *output quality* depends upon the particular decision procedure. An objective metric must be chosen for this parameter. It captures the difference between an approximation and the optimal solution for a problem instance. In ideal situations performance profiles should be machine independent. However, in general a performance profile is specific to the decision procedure and its implementation environment. This implies that a performance profile in a uniprocessor environment may not immediately generalize to a multiprocessor environment. Furthermore, resource limitations may affect the decision procedure and therefore become implicitly represented in the performance profile.

Performance profiles are generally derived from experience. In certain limited cases they can be derived from a structural analysis of the corresponding decision procedure. In addition to graphical and tabular representations, performance profiles can be represented in closed form. This becomes especially important when we consider multidimensional profiles for which graphical representation becomes difficult to handle. A closed form can be derived from statistical experience through curve fitting, from standard classes of distributions or from other approximations. A closed form for an expected performance profile may be a piecewise linear or negative exponential function whereas a closed form for a performance distribution profile may have a normal distribution about the expected output quality.

The previous discussion has focused on two param-

eters of a decision procedure; namely, output quality and processing time. There are many more parameters that may be of interest in deriving performance profiles. Some parameters are implicit. It is assumed that a performance profile and its decision procedure are only applicable to a specific domain of problems. As mentioned above a performance profile may be implicitly tied to a specific computation resource. For example the quality of plans produced by a decision procedure that has access to unlimited storage may be superior to one constrained by limited storage, given equal processing time. Likewise, a uniprocessor algorithm may not generalize to a multiprocessor architecture. Thus, any resource limitations may be integral to the applicability of profile.

Other parameters are explicit. Some work has been done in differentiating the *objective value* of the output of a decision procedure with the *comprehensive value* stemming from its use in a particular situation [11]. An example of such a differentiation would be a successive approximation integration procedure. The objective value of such a procedure is measured in terms of numerical error, but the contribution to comprehensive value differs depending on whether the procedure is used for medical diagnosis or stock trading. We may explicitly account for the target use of the decision procedure by constructing different profiles for different uses. Even within the same domain the target time of use of the decision procedure output and the time window of applicability of the output may be explicit features of a performance profile.

The decision whether a parameter is implicit or explicit is not always straight-forward. For example, it may be necessary to construct separate profiles for different levels of resource limitations. Alternatively we can add a new dimension to our performance profile to account for different resource limitations. The decision on whether a parameter need be explicitly represented as an additional dimension, explicitly represented as a set of conditional performance profiles or implicitly assumed as part of the single profile for a decision procedure depends upon how the profiles will be used in deliberation scheduling.

2.2 Conditional Performance Profiles

In the previous section we alluded to *conditional performance profiles*. In this section we clarify their im-

portance. Conditional performance profiles are a way to represent the dependence of performance of a deliberation procedure on various input parameters. For example, resource limitations and the time window of applicability of decision procedure output are two specific input parameters. Conditional performance profiles are an important tool in anticipating computational demands in that they allow the planner to reason about the context in which a decision procedure is used.

In general, a conditional performance profile conditions a performance profile on variations in the input parameters. One such parameter is the quality of the problem instance. This is a particularly important variation in that the quality of a problem instance of one decision procedure in a sequence is often directly related to the output quality of prior decision procedures. Other input parameters such as resource limitations may have no relation to the context in which a decision procedure is used and, therefore, have little direct impact on deliberation scheduling. Nevertheless, if resource limitations can be varied and reasoned about they may affect the output quality and, therefore, affect the deliberation scheduling problem. In deliberation scheduling, reducing time allocated to any decision procedure may affect any decision procedure that uses the results of the first and all subsequent procedures.

With conditional performance profiles, instead of deriving a single profile for all possible input parameters, we may instead have separate profiles that partition the set of possible values of input parameters. As mentioned, one particular input parameter of interest is the quality of the problem instances. The performance profile depicted in Figure 1.i is input independent if it is applicable to all possible input parameters (*e.g.* all problem instances, regardless of quality). A corresponding conditional expected performance profile is depicted in Figure 2.i. In this figure the problem instances are partitioned into three discrete classes of input quality, each with its own expected performance profile. Figure 2.ii depicts a profile for a decision procedure in which, for any constant desired output quality, the processing time to achieve that quality decreases as input quality increases. In Figure 2.iii the input quality is given its own dimension in a conditional probability distribution profile of the joint probability of input quality, output quality and processing time. If the problem inputs may be partitioned based on multiple

Anticipating Computational Demands

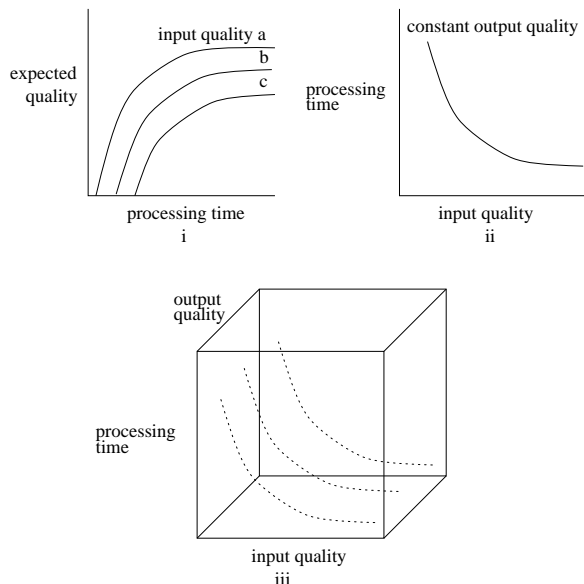


Figure 2: *Conditional Performance Profiles*

parameters (in addition to problem instance quality) we may either construct a new dimension for each parameter or combine the inputs into a single measure.

Zilberstein points out that a performance distribution profile in which the probability distribution is widely spread out may indicate that the decision procedure performs differently under differing inputs. It may be useful in this case to gather statistics independently for different classes of input parameter values in order to obtain more specific performance profiles. For probability distributions with small variance, an expected performance profile compactly represents the decision procedure applied to all relevant input parameter values including all varying problem instance qualities. Alternatively, variance in a performance distribution profile may be used as an indication of the *sensitivity* of a decision procedure to variations in input parameters.

An important use of conditional profiles is in applications in which a regular sequence of classes of problem instances can be expected. This is discussed in Section 4.

2.3 Compilation of Anytime Algorithms

Deliberation scheduling is a special case of the more general problem of composing anytime algorithms with

conditional performance profiles to satisfy overall performance measures. Compiling anytime algorithms is the process of combining multiple algorithms with separate profiles into a single module with a new profile. For example we may compose anytime algorithms $g(x)$ and $h(y)$ into a new anytime algorithm $f(g(x), h(y))$ that combines the properties of both to optimize combined performance. A new profile describes the resulting behavior.

There are at least two aspects to consider in deliberation scheduling. One, how to determine the sequence of problem instances that need to be solved; and two, how to allocate processing time among decision procedures to solve those problem instances with satisfactory overall performance. These two problems are interrelated in that the allocation of time to solve earlier problem instances may help to determine the later problem instances (*i.e.* one measure of quality of a decision procedure may be its ability to improve the quality of subsequent problem instances). By anticipating the sequence of problem instances that will be realized we can anticipate the computational demands of decision procedures applied to these problem instances. We can then reason about the tradeoffs between allocating time among the procedures and the subsequent overall quality.

In some environments the sequence of problem instances may be predicted in advance of any deliberation. Alternately, the prediction of potential problem instances may be part of the decision procedure itself and reflected in its conditional performance profile. In [4, 1] the sequence of problem instances is indicated in advance by *precursor events*. In Sections 3 and 4 we discuss prediction by taking advantage of regularities in the domain.

Once we have a given set of problem instances or have narrowed the sequence of potential problem instances to classes of instances with similar conditional performance profiles, the deliberation scheduling problem can be considered to be a problem of compiling a sequence of anytime algorithms so that the overall performance is maximized. In general when performance profiles are represented in closed form the compilation problem involves solving differential equations; when represented in tabular format the compilation problem becomes a search problem. Compilation problems in which the problem instances are independent are con-

siderably easier than those in which the problem instances are dependent.

An important special case of compilation of anytime algorithms is when we can guarantee that the decision procedures (with their associated conditional performance profiles) will be executed in a strict sequential order. Similar are predictably periodic sets of problem instances or classes of problem instances.

3 Response Planning Approach to Embedded Planning

In this section we present an approach to solving time-critical decision-making problems by taking advantage of domain structure to make better use of the time available for processing difficult combinatorial tasks. Our approach uses predictable variability in anticipated computational demands to allocate on-line deliberation time and exploits what we call problem regularity and stochastic models of environmental dynamics to restrict attention to small subsets of the state space. This approach demonstrates how slow, high-level systems (*e.g.* for planning and scheduling) might interact with faster, more reactive systems (*e.g.* for real-time execution and monitoring) and enables us to generate timely solutions to difficult combinatorial planning and scheduling problems such as traffic control for multiple robot vehicles.

We make use of domain structure in two principal ways. First, we use a stochastic model of the domain and on-line knowledge of the state of the environment in order to predict future states. In this context a set of possible states in some target time period corresponds to a problem instance. The ability to predict future states allows us to begin processing in advance of the actual states. Furthermore, careful prediction allows us to restrict our planning to a small subset of the state space. We may consider the size of the state space corresponding to a predicted problem instance a measure of input quality. In terms of conditional performance profiles, restricting the state space to smaller subsets of states reduces the anticipated computational demand of the corresponding problem instance. Prediction alone is insufficient to make progress with these types of problems. Second, we take advantage of large variabilities in anticipated computational demand across time. By removing the restriction of equal processing

allocation for each problem instance, we gain flexibility in allocating processing time. Additionally, some forms of regularity allow us to meta-reason off-line about the allocation of on-line deliberation time.

3.1 Embedded Planning

We define *embedded planning* to be the problem of determining actions for an agent embedded in an uncertain environment with dynamics outside the agent's control. A salient feature of this problem is that the agent must react to changes in the environment in *real-time*, *i.e.* the agent must sense the state of the environment and act appropriately in a timely manner. There is rarely the luxury of waiting for a deliberative planner to construct the optimal action in response to environmental dynamics. There are hard deadlines imposed by the environment that, if violated, can lead to disastrous conclusions for the agent.

As mentioned earlier, we may consider each new state or set of possible states of the environment during a specified time period to be a problem instance. We define reactive response to be the ability to sense a change in the state of the environment and respond before the environment changes state again. *Response time* for reacting to a problem instance is defined as the minimum time between sensing a state and the necessity of response. The minimum time between state changes (granularity) is referred to as a *time step* and is domain dependent. We assume for these purposes that response time is a fixed constant. We can then divide time into discrete steps where a state is sensed at the beginning of each time step (the state may be the same as in the previous time step) and a response is required by the end of the time step (in some situations a null response may be satisfactory).

Only in ideal situations may we determine in advance a sequence of individual states (one per time step) to be used as problem instances for deliberation scheduling. In general, the best we may do in advance is to restrict each time step to a set of reachable states. Each of these sets may then either individually or in temporal or similarity based partitions be considered problem instances for deliberation scheduling. The choice of partitioning determines the form of conditional performance profiles required for deliberation scheduling.

A solution that provides a response at each time step is called a *policy*. A policy π is a mapping from states to

responses (actions). A *partial policy* provides responses to only a subset of possible states. A policy may be used as part of a real-time control system that responds to changes in the environment. In this model policies are constructed by applying decision procedures to problem instances that partition the phase space (product of state space and time steps). The focus of the response planning approach is to determine how to allocate on-line deliberation time to the decision procedures that jointly construct policies, such that particular performance criteria are satisfied. Additionally, response planning provides an interface between the combinatorial process of constructing the policy and the reactive execution of the policy.

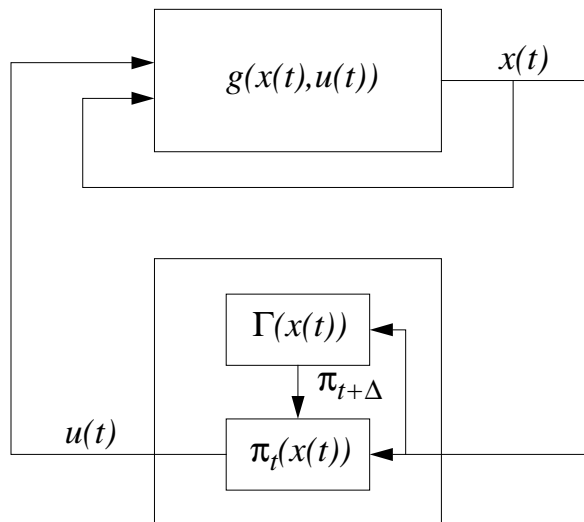


Figure 3: Model for embedded planning and control

Figure 3 captures the essential properties of embedded planning and control systems; $x(t)$ is the state of the system (including the embedded agent) at time t , $u(t)$ is the action taken at time t by the composite planning and control system, and the function $g(x(t), u(t))$ determines the dynamics of the system. In this work we assume that we are dealing with a discrete time system. The action is determined by a policy that can be executed by the real-time control system. The policy π_t has a temporal index to indicate that it may change under the control of the planning component Γ . Due to the combinatorics involved in planning there is typically a delay Δ between when a state triggers the planning process and when the resulting policy is avail-

able for execution. We distinguish between state and control in keeping with standard practice in the control literature. For a description of the motivation and use of this formalism for modeling dynamical systems see [5, 9, 13].

3.2 Response Planning

Response planning is an approach to embedded planning that makes specific assumptions about the underlying environment and computational resources. In [10] we outline the general motivations and options for designing composite planning and control systems.

The response planning approach combines the reactivity of off-line construction of *universal plans* [19] (complete policies that precompute actions for all world states) with the adaptability of on-line decision-making. Time-critical decision-making is accomplished despite the constraints of limited storage and computational resources in the face of large state spaces. We do so by noting that predictive knowledge of the underlying domain may be used to anticipate the reachable states in a problem instance before actually observing the environment at the corresponding time step(s). This knowledge is then used to compute partial policies for such restricted subsets in advance. Complete policies are produced by combining the partial policies with default responses for remaining states. Problem variability allows us to provide strategies in which difficult and uncertain problem instances consume more processing time than less computationally intensive problem instances. Domain-dependent regularity allows for off-line precomputing of deliberation scheduling (processing time allocation) strategies.

As defined in Section 3.1 the primary components of an embedded planning solution are the reactive component π and the deliberative component Γ . π must be able to execute a policy constrained by hard deadlines. Without loss of generality we consider the reactive process (π) to be comprised of a *reactive process table* of responses indexed by world states. In general π can be realized by any computation that is initiated in response to a state and completed before the arrival of the next state.

Restricting π (we abuse the notation by calling both the process and it's table π) to be a table simplifies the interaction between the deliberative process (Γ) and the reactive process (π). In particular, the deliberative

process interacts on-line with the reactive process by populating the reactive process table with responses for upcoming time steps. To represent changes in the policy over time we index the reaction table by time steps, (*i.e.* π_j for time step t_j) and require a complete policy for each time step (though not necessarily unique).

One motivating assumption for the construction of π is that in typical embedded planning problems the state space is very large. In particular, the number of states that make up environment g is generally exponential in the number of state variables in the embedded planning problem. Given limited resources (storage and computation) large state spaces eliminate the possibility of computing and storing responses for every possible state in the environment. Therefore, we require that each π_j be bounded in size. In particular we bound the size k of π_j to be of low-order polynomial in the number of state variables. Define π_j^k to be a reactive process table of size k .

By introducing a bound on the size of each π_j we limit the possible policies that Γ can construct. In particular, for time steps in which greater than k states are reachable, Γ must determine the optimal set of states for which to plan, given some optimality criterion. Alternatively, we could redefine π_j^k as a table indexed by equivalence classes of states. In some embedded planning problems such as partially observable processes in which the actual state is difficult to measure with certainty, indexing by equivalence classes of similar states (similar in terms of policy) could be a more natural solution. On the other hand, doing so would sacrifice fine-grained control over the environment. In Figure 4 we depict π_j as a combination of both. A bounded number of responses triggered by equivalence classes of states and a bounded number of responses triggered by states. Equivalence classes may be used either to capture partial observability or to encompass states that are not included in a given partial policy π_j . In the latter case we can consider the responses triggered by equivalence classes to be default responses to be applied if none of the state-based responses are appropriate. Thus, the deliberation scheduler can reason about which states to include in π_j given the expected quality of executing default responses.

Bounding the size of π_j makes it even more important to provide accurate on-line prediction to reduce the size of the reachable state space. Since a con-

$\pi_j(x(t))$	
Trigger	Response
$[x(t)]_{s_1}$	u_{s_1}
$[x(t)]_{s_2}$	u_{s_2}
...	...
$[x(t)]_{s_k}$	u_{s_k}
$s_1(t)$	u_1
$s_2(t)$	u_2
...	...
$s_k(t)$	u_k

Figure 4: Tabular, bounded size representation of π_j^k partitioned into similarity-based equivalence class indexed responses and state-indexed responses; where s_k is a known state equal to $x(t)$ and $[x(t)]_{s_k}$ is the class of states determined by $x(t)$ and represented by s_k .

ditional performance profile for a decision procedure takes into account both the limitations on the resources and the quality of the input, if we fix the resource limitations we are left with the quality of input as our most important degree of freedom in deliberation scheduling. In particular, the quality of input (*i.e.* the quality of the particular problem instance achieved for the time period) is dependent upon how far in advance of the time period the decision procedure is called and the ability to restrict the size of the reachable state space via prediction.

The response planning approach requires certain properties of the problem domain to hold. A first requirement is that computing high-quality responses for some states requires multiple time steps. If this requirement does not hold then prediction and deliberation in advance are unnecessary. Purely reactive approaches apply. In addition there must be states for which responses can be computed quickly (*i.e.* fractions of time steps) or for which responses, once computed, are relevant for many time steps. Otherwise, delay caused by multiple step processing would grow unbounded.

We must take limited resources into account when designing Γ . In general, Γ does not have the luxury of computing optimal responses for all reachable

Anticipating Computational Demands

states for all future time steps. Γ is an on-line process that performs three separate functions. It performs domain specific processing to compute the responses to be taken for sensed states (*i.e.* constructs policies); it uses the underlying environment model g to predict reachable states in upcoming time periods; and, it performs deliberation scheduling by reasoning about its use of time based on conditional performance profiles, predictions and performance measures.

The ability to predict future reachable states depends upon Γ 's knowledge of g . If g is deterministic then Γ can predict the exact state for any time step given some start state. If g is stochastic then the best Γ can do is predict a distribution over states for a given time step.

We divide Γ conceptually into two components, f and Γ . The on-line decision procedure, f , performs prediction of reachable states and construction of policies. f may be called with varying input parameters including the target time window for the policy and the time allocation for deliberation for policy construction. The deliberation scheduler, that maintains the name Γ , reasons with the aid of the conditional performance profile associated with f to determine the start time step and processing time to allocate to f in order to provide the best possible performance according to a given performance criterion.

It is important to note that, although it may be possible to compute stationary distributions of reachable states off-line and construct a single table π^k , there are advantages to performing prediction on-line. In particular, a stationary distribution is only applicable asymptotically. In the short-run it is beneficial to look at states that have recently occurred to prune the possible states reachable in the near future and calculate distributions over those states for upcoming time steps. In a sense, on-line processing allows us to compute *on demand* rather than process all possible states. If g can be modeled as a Markov chain then we need only look at the most recently occurred state in predicting future distributions. This can be further simplified by using dependencies among state variables to provide short-term predictions [3].

f is an on-line process that, given a trajectory of previously visited states, and a target time window uses the known model of g to predict and prioritize reachable states. It may then calculate responses for the

states so as to maximize the expected quality of the resulting policy. One particular formalization for f is as follows. Given a state trajectory $H = s_0, s_1, \dots, s_i$ for times t_0, t_1, \dots, t_i , a projected time t_n ($i \leq n$) and a *window size* m , f computes a series of reactive process tables $\pi_n^k, \pi_{n+1}^k, \dots, \pi_{n+m-1}^k$ such that each respective table π_j^k contains a policy with state-based responses for k states reachable at time t_j . In addition, (for completeness) the policy may include class-based default responses as previously described. Given our resource limitations we must also restrict m to be low order polynomial in the size of the state space, just as we did for k . As a special case, if the underlying source g is a Markov chain then we need merely provide f with the most recent time and state (t_i, s_i) rather than the entire trajectory.

Decision procedure performance may be optimized by designing deliberation scheduling (Γ) such that the predicted set of reachable states is minimized for the target time window (thus, maximizing input quality). Short-term dependencies may be taken advantage of by designing Γ as a reactive process that schedules deliberation in response to states of the environment g . Specifically, at each time step Γ can either commence a new call to f with a set of parameters H, t_n, m (that may cause an existing call to be exited) or do nothing. When completed, Γ populates $\pi_n^k, \dots, \pi_{n+m-1}^k$ with the results of the call to $f(H, t_n, m)$. Given bounded resources we must limit the number of calls to f that are active at any given time. Since processing profiles are based on exclusive use of a uniprocessor, multiple calls to f at a time require a multiprocessor system. As mentioned previously, conditional performance profiles for uniprocessor systems may not generalize to multiprocessor use.

The above description of Γ as a reactive process overlooks some important points. First, the deliberation scheduling optimization problem that Γ must solve, namely determining the optimal call to f for a given time step, is combinatorial in nature. In other words, it may take multiple time steps to compute. Second, for the same reasons that π cannot be a universal plan, Γ cannot store actions (calls to f) computed off-line that are applicable to every state. In order to be effective Γ must be both reactive, to interrupt f during processing in response to actual state trajectories, and deliberative, to reason about f 's processing requirements given the underlying process. As before, this dilemma can

be solved two ways. Either limit the number of states that can be reacted to or perform deliberation in advance based on prediction. But, if we choose the latter approach to Γ we would still need to schedule calls to Γ which schedules calls to f , ad infinitum. We avoid the infinite regress in this case by the expedient of compiling Γ off-line into a reactive *strategy table*.

Γ is designed as a strategy table such that for each time step, based on the observed state, Γ defines a strategy to either call f with particular parameters (for one or a sequence of calls to f) and wait for the results (possibly terminating an existing call) or do nothing. Again when constructing a bounded on-line table we must decide between indexing by actual states and default computations or indexing by equivalence classes of states. The same arguments apply here as did for the construction of π , namely equivalence classes are a natural way to express some problems but they suffer from a loss of fine-grained control over the environment. We choose to construct Γ in general similarly to π as a bounded table partly indexed by environmental states and partly indexed by equivalence classes of states.

We have alluded to a particular domain structure we call *regularity*. This structure occurs naturally in problems such as traffic control for multiple robot vehicles. In such problems we can partition the state space into a sequence of equivalence classes of states corresponding to contiguous time periods. In general, this regularity is disrupted only by catastrophic (very low probability) events. Figure 5 demonstrates what we mean by a regular sequence of equivalence classes. As long as regularity is maintained the process will proceed in a fixed, possibly repeating sequence of equivalence classes of states. Catastrophic events force the process into a unique state that must be handled separately. The actual state trajectory may vary widely from one repetition to the next but, the sequence of classes is, for the most part, deterministic. Another way to look at regularity is as a property of inherent local dependencies among state variables. In this case the state at time t is only dependent on a subset of the state variables at times prior to t .

Regularity is applicable in the design of strategy tables in that it allows for the off-line construction of *rigid* strategies. A rigid strategy is computed by performing deliberation scheduling off-line using (temporally-

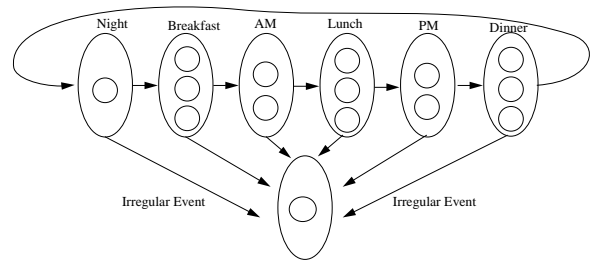


Figure 5: *Regularity demonstrated by deterministic sequence of equivalence classes; catastrophic events disrupt regularity.*

based) equivalence classes of states as the input to f rather than actual states. In other words, the conditional performance profiles for f have as input parameters the target class and the start class rather than specific states. Problems in the same equivalence class have similar anticipated computational demands. The argument here is that class-based partitioning of the problem provides the deliberative scheduler with as much information as state-based partitioning, given the additional structure of regularity. This is not true in all domains but is a nice property of some domains that we can exploit through rigid strategies.

Additionally, we provide a mechanism within Γ for state-based triggering of strategies. In domains in which regularity may be occasionally violated we can use state-based triggering to provide default strategies that attempt to regain regularity. This is similar to the principal of match-up scheduling used in operations research. Figure 6 depicts Γ as a strategy table with a rigid strategy indexed by time step (temporal equivalence class) and a bounded number of default state-based strategies. In Section 4 we further discuss the implications of regularity for deliberation scheduling. In other work we are considering indexing among a bounded number of rigid strategies as well. In non-regular domains we may provide for state-based deliberation scheduling in which a bounded number of states trigger strategies directly.

One formalization for Γ is to construct one strategy table per temporal equivalence class. For an environmental model with period d , Γ is constructed off-line into d strategy tables, $\Gamma_0, \dots, \Gamma_{d-1}$, one for each temporal equivalence class. Each strategy table contains h specific strategies indexed on actual states and one

Trigger	$\Gamma_t(x(t))$ Strategy
$[x(t)]_t$	$(f(\sigma), \dots)$
$s_1(t)$	$(f(\sigma_1), \dots)$
$s_2(t)$	$(f(\sigma_2), \dots)$
...	...
$s_h(t)$	$(f(\sigma_h), \dots)$

Figure 6: Tabular, bounded size representation of Γ_j partitioned into one temporal-based equivalence class corresponding to the domain regularity and state-indexed strategies to account for non-regularity.

default strategy to take for this equivalence class if no specific strategies apply.

Strategies limited to a single call to f per time step will not generally lead to optimality, especially when the processing time for some states is considerably less than a single time step. Thus, we design Γ to reason about a sequence of calls to f at each time step. If preemption of decision procedures is desirable, the strategy table Γ may contain a sequence of *partial* calls that can be swapped out and resumed later. This requires small additional overhead in processing time and storage but does not violate any assumptions of our model. In this case the profile for a preempted job must correspond to the profile when initially started (no intermediate knowledge may be used).

We make one further restriction concerning the interaction of Γ , f and π given bounded resources. We have described Γ as populating π with policies generated by f , with one policy π_j per time step t_j . Although we have bounded the number of policies a single call to f can produce (by the time window m , which is polynomial in the number of state variables), we have not bounded the cumulative number of policies that can be produced by a sequence of calls to f and stored by the system before being executed. To do this we define a *time horizon* l that determines how far in the future for which f can plan. In particular for any current time step t_i , window start time t_n and window size m , $n + m - i - 1 \leq l$. If we restrict l to be polynomial in the number of state variables then the total number of policies stored at any time is also polynomial in the number of state variables.

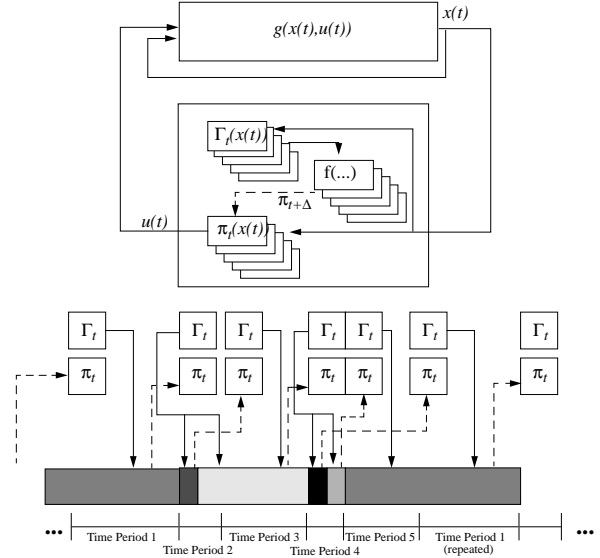


Figure 7: Response Planning model and example

Given the components of our response planning solution to embedded planning summarized in Figure 7 we have two algorithms to construct. The off-line construction of Γ taking f as input (deliberation scheduling) and the on-line execution of response planning. We can sketch the on-line algorithm that governs response planning in general as follows. At each time step t_j we perform the following functions:

1. determine state s_j of environment g ;
2. determine temporal equivalence class $\text{equiv}(s_j)$ for state s_j ;
3. determine similarity-based equivalence class $\text{similar}(s_j)$ for state s_j ;
4. index into reaction table π_j to find response corresponding to state s_j ;
5. if response found execute, else execute default response corresponding to $\text{similar}(s_j)$;
6. index s_j into strategy table for time period t_j ($\Gamma_{\text{equiv}(t_j)}$) to find parameters for new (partial) call to f (or sequence of calls, subject to horizon limitation l) or no op;
7. if strategy found execute it, else execute rigid strategy corresponding to $\text{equiv}(t_j)$;
8. if previous call to f has terminated successfully update π for that time window by storing the m

- constructed policy tables on-line;
9. if executing new strategy requires previous call to f to be interrupted before completion then update π with most recent policy produced by anytime algorithm for f ; and
 10. discard any out-of-date policies.

Note that we make the assumption that the state, temporal equivalence class and similarity-based equivalence class can all be sensed immediately from the environment.

New calls to f include parameters such as target time window and processing time allocated. This procedure is exemplified at the bottom of Figure 7. In this figure we show a new call to Γ at the beginning of each time period for simplicity. In general, Γ can instigate a new call to f in response to any state. In the figure we see Γ calling f during first time period to construct policies for second time period. We further note that just prior to the second time period, f updates π with its constructed policy. This policy is then in effect until the end of the second time period at which point the policy for the third time period takes effect.

4 Exploiting Regularity in Anticipating Computational Demands

In this section we return to the notion of regularity and explore it in the context of deliberation scheduling and compilation of anytime algorithms. In doing so we tie together the particular approach to solving time-critical decision-making problems proposed by response planning with the general idea of anticipating computational demands. In Section 4.1 we compare the deliberation scheduling task of response planning to that introduced by Boddy and Dean [4, 1]. In Section 4.2 we show that the notion of compilation of anytime algorithms introduced by Zilberstein [22] encompasses the particular notions of deliberation scheduling used in response planning.

In response planning we introduce the notion of regularity so that deliberation scheduling may be performed off-line to construct rigid strategies. In particular, we assume that our environment g proceeds in a fixed, possibly recurrent pattern of temporally equivalent classes of states. Although at any given time step (or time

period) the class of states that can occur is deterministic, the actual state experienced from a given class is stochastically determined by the environment g . Given that there may be a very large (exponential in the number of state variables) number of states in any equivalence class, this regularity does not trivialize the response planning problem.

We use this deterministic knowledge of the sequence of equivalence classes in deliberation scheduling by reasoning about conditional performance profiles in which the input parameters are start class and target class rather than start state and target state. Inherent in this usage is the assumption that the performance of our decision procedure for a given allocation of processing time is determined by these two classes (as well as the window size). In [10] we use an air traffic control problem to motivate these assumptions. The problem becomes more involved if the responses constructed by f can alter the sequence of classes visited. We allow for this possibility in response planning by extending a rigid strategy to include some state-based strategies. More generally, we could extend Γ to include conditional or stochastic strategies but, that is beyond the scope of this treatment.

4.1 Deliberation Scheduling

Deliberation scheduling is a resource allocation problem in which the resource is processing time and it is allocated among competing decision procedures to be applied to varying problem instances. An embedded agent can anticipate computational demands by determining the problem instances for a forthcoming time period and accessing profiles indicating the expected performance for varying processing allocations for varying decision procedures applied to similar problem instances in the past.

In response planning the competing decision procedures involve computing policies for target time periods (problem instances). Regularity guarantees that the target time periods are known deterministically in a fixed sequence. Deliberation scheduling applied to response planning involves determining the start class, processing time allocation and time window for each target time period in order to maximize overall performance. In response planning, overall performance is determined by the expected performance of the policies executed by π .

Anticipating Computational Demands

The assumption of regularity simplifies the problem of determining the problem instances from which to base deliberation scheduling. Other researchers have used different techniques. In [4, 1] the sequence of problem instances is indicated by precursor events. A precursor event indicates to the deliberation scheduler a sequence of future events (problem instances) that need to be solved. However, determining the sequence of problem instances for deliberation scheduling based on precursor events requires that deliberation scheduling be performed on-line. By exploiting deterministic regularity response planning can move the deliberation scheduling problem off-line.

Once the sequence of problem instances is determined, the deliberation scheduling problem is to allocate processing time and, possibly, other parameters to the competing decision procedures. One important difference between response planning and the deliberation scheduling of Boddy and Dean [4, 1] is that our problem requires that the deliberation scheduler determine both time allocation and start time for processing a decision procedure for a particular problem instance. The treatment by Boddy and Dean [4, 1] does not require this because they assume that decision procedures are independent and only the relative processing allocation is important, not the start time. In the following we show that we can convert a class of response planning problems with start class dependence into start-class independent deliberation scheduling problems. Although this result is interesting it is specific to the properties of start-class dependent profiles and cannot be generalized to other conditional performance profiles.

Consider a deliberation scheduling problem for response planning that consists of a set of conditional performance profiles, one per equivalence class (corresponding to a fixed time window¹). The dimensions of these performance profiles include start-class, processing time allocated and output quality. A typical start time independent performance profile is given in Figure 8.i and a corresponding set of start-time dependent conditional performance profiles are given in Figure 8.ii. Figure 8.iii shows that this dependent set of profiles demonstrates a situation in which there is benefit to starting later when there is more informa-

¹We have argued that we require different profiles for differing size time windows so, we assume here that the size of the time window is fixed.

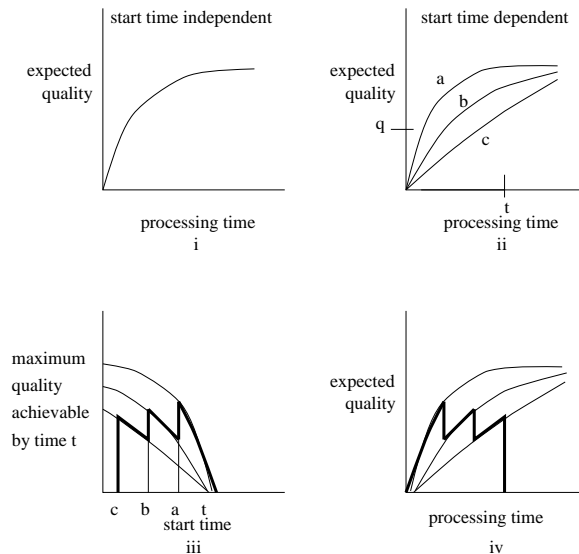


Figure 8: *Converting from start-time dependent conditional performance profiles to processing time versus quality performance profile.*

tion available (more accurate prediction). In particular, if we start processing before time b , performance is determined by the time c conditional performance profile which takes longer to achieve quality q than either the time b or time a conditional performance profiles. The variability in profiles based on start time could be related to the increased uncertainty in predicting reachable states further into the future.

In Figure 8.iv we see the resulting start-time independent profile. In a sense the profile is not start-time independent at all but, rather, the start-time dependence has been incorporated into the profile itself (and relies on the fact that processing takes place in time interval immediately prior to target time class). This profile shows that in the deliberation scheduling problems of interest to response planning, allocating additional processing time does not always improve quality because it requires starting processing sooner when less accurate prediction information is available.

While this transformation displays the relationship between the deliberation scheduling problems of response planning and those of Boddy and Dean it does not allow us to use the algorithms presented by Boddy and Dean because the resulting profiles do not display the required monotone increasing property. This

clearly indicates that start-time dependent deliberation scheduling is in general more difficult than start-time independent deliberation scheduling.

4.2 Deliberation Scheduling as a Compilation of Anytime Algorithms

The deliberation scheduling problems of response planning and Boddy and Dean are both special cases of the general problem of compilation of anytime algorithms. We have already shown that deliberation scheduling with start-time independence is a special case of deliberation scheduling with start-time dependence. In this section we show that start-time dependence is a special case of input quality dependence.

The point to note in interpreting a response planning deliberation scheduling problem as a problem of compiling anytime algorithms is that conditioning profiles on target equivalence classes is equivalent to conditioning profiles on input quality. The input quality is a direct function of the target time period and start-time.

Zilberstein defines a conditional performance profile as corresponding to a decision procedure that may have several different performance profiles, each characterizing its performance when operating in a different environment. In response planning each problem instance corresponds to a different temporal equivalence class. The ability to restrict the set of reachable states within a class by prediction is equivalent to altering the input quality. Prior actions can affect the set of reachable states by forcing the state trajectory off the fixed sequence of equivalence classes defined by problem regularity.

Once we have a given set of problem instances or have narrowed the sequence of potential problem instances to classes of instances with similar conditional performance profiles, the deliberation scheduling problem can be considered to be a problem of compiling a sequence of anytime algorithms so that the overall performance is maximized. Algorithms for which problem instances are independent are easier to compile than dependent problem instances. In the deliberation scheduling problem of response planning we have a fixed sequence of problem instances for which to allocate processing time. In general, compilation of anytime algorithms deals with any partial order of problem instances.

One interesting point is that in response planning the input quality is not the output of a previous algorithm but rather a combination of the results of deliberation scheduling and the prediction component of f . This indirect dependence of input quality on deliberation scheduling is a subtlety that must be dealt with carefully. The anticipated computational demands of a decision procedure is based on the expectation that the results of a prior decision procedure will not cause the system to stray from the fixed regular sequence of classes. In response planning the algorithm that provides the expected input for a subsequent algorithm can actually be executed after that algorithm has finished deliberating as long as the result is ready before the first algorithm needs to execute the results of its deliberation. The first algorithm must use some expectation of the quality of the second algorithm to do its deliberation. If the second algorithm does not live up to that quality then it is too late and the deliberation provided by the first algorithm is severely diminished. This discussion points out the fact that response planning has three stages, deliberation scheduling, actual deliberation for policies and execution of policies. The execution of policies has a strict order used by the deliberation scheduling but the deliberation itself does not have a strict order and relies upon the a priori knowledge of the expectations of the other deliberations to provide accurate policies.

5 Response Planning Example: Traffic Planning and Control of Multiple Robot Vehicles

Consider the problem of employing multiple robot vehicles for delivering meals and pharmaceuticals and disposing medical waste in a large hospital. Robot vehicles are ideal for this task because of their low air contamination and ability to safely handle hazardous materials. In an artificial setting, such as a factory floor, tasks can be engineered so that path planning and routing problems are solved off-line and then implemented by a centralized controller. In the worst case there may exist delays and uncertainty that are handled in standardized ways. However, in a natural environment such as a hospital, tasks cannot be predicted with certainty ahead of time. Therefore, in order to provide effective use of resources, traffic planning and control must dynamically adjust to changing

Anticipating Computational Demands

situations.

In this problem we assume a finite set of vehicles, a finite set of locations (source and/or destination), and a fixed network of limited-width undirected paths connecting the locations. The network serves as a pathway for the vehicles. The pathway may be augmented with infrared beacons to aid vehicle navigation. Additionally, vehicles can be controlled and tracked either through the network itself or through other devices such as radio ethernet or infrared transceivers. At any given time a subset of the vehicles are active on the network with specific destinations and, possibly, deadlines. The traffic planning and control problem is to schedule routes for these vehicles such that they reach their destinations with minimal tardiness. In order to do so the planner must avoid congestion, bottlenecks and contention within the limited-width network for currently active vehicles while taking into account the anticipated activation of inactive vehicles in the near future. It may be necessary to assume single-lane pathways in some problems (such as those of automatic guided vehicles), thus exacerbating the contention problems.

We can assume for simplicity that the traffic planner need only anticipate congestion caused by vehicles. The problem of avoiding obstacles and people can be handled by an on-board reflexive control system or may be engineered away by providing special conduits (such as in-ceiling crawl-space) for vehicle movement. This is consistent with traditional separation into central controller and on-board vehicle controls. Furthermore, we can assume that assignments of delivery and disposal tasks to specific vehicles are determined by an external source (or sources) and not under the control of the traffic planner. This includes obeying capacity constraints at the source and destination locations.

A combined planning and control system may be employed to mediate traffic congestion. The controller issues traffic control commands to the active vehicles in order to satisfy the routes determined by the planner. Prior to entering an intersection or approaching a source or destination location a vehicle is given a traffic control routing action indicating which of the alternative pathways to follow.

Formally, the following parameters describe the planning and control problem.

- a network including

- vertices, N , corresponding to internal routing points,
- locations, L , corresponding to source and destination points, and
- edges, E , connecting vertices and locations;
- a set of vehicles, V ; each vehicle $v \in V$ has
 - current location, $loc_v \in (N \cup L \cup E)$, ($loc_v \in L$ indicates that v is currently inactive),
 - current destination $dest_v \in L$,
 - optional current deadline $dead_v \in \mathbf{T}$ (\mathbf{T} is the set of all time points, possibly infinite), and
 - current routing action, $act_v \in \mathbf{Z}$, indicating the pathway to follow at the next vertex or location; and
- a stochastic process g modeling environmental dynamics and future vehicle task assignments (described below).

The state of the system at any time step is the set of parameters for all vehicles. At each time step the controller issues a new vector of routing actions to the vehicles to satisfy routes determined by the planner. The subsequent state of the system is governed by a stochastic process g whose transitions model control actions and external, uncontrollable events (such as inactive vehicle assignment, active vehicle breakdown, velocity uncertainty, delay, etc.,).

The performance of the planner is measured by an objective function on the movement of vehicles. For delivery tasks with deadlines this may be to minimize tardiness whereas for other tasks we may try to minimize total, maximum or average completion times across all tasks. Tardiness and completion times are affected by congestion of vehicles in limited-width paths as well as uncontrollable delays modeled by the stochastic process.

Constructing routes to minimize the above objective functions is a difficult combinatorial problem that is made increasingly difficult by uncertainty in future parameters. For single-lane pathway problems we can consider the routing problem to be analogous to a space of $(N \cup L \cup E) \times \mathbf{T}$ points and the task to commit (assign) at most one vehicle to each point in this space, for some given horizon of time steps into the future. However, the dynamics of the problem require that any routing solution be modified regularly. Possibly

as often as every time step. The planning algorithm determines a routing assignment based on a given set of problem parameters and conveys this information to the controller in the form of a policy. A policy describes a vector of routing actions for any given system state. A partial policy includes a vector of routing actions for some subset of system states and can be augmented by default control vectors (*e.g.* stop all vehicles) for the remaining states. Partial policies are necessary in large state spaces given limited on-line storage and computation resources.

In this paper we are not primarily concerned with the actual planning algorithm to solve the optimization problem. We are more interested in the ability to characterize planning algorithms by their processing time and quality measures given particular input parameters (*e.g.* planning window). The actual planning solution itself may be heuristic or exact.

Executing default control vectors is very costly and is to be avoided as often as possible. We focus on the task of scheduling the on-line deliberation consumed by the planning algorithm in order to minimize cost (maximize expected quality of policies).

The multiple robot vehicle problem of delivering meals and pharmaceuticals and disposing medical waste in a large hospital satisfies the regularity limitations that we require for our response planning approach. In particular, the pattern of delivery activity throughout the hospital varies widely throughout the day. Meal delivery vehicles are abundant during meal-times and rarely active during other periods. Pharmaceutical delivery and waste disposal vehicles have some scheduled deliveries and some deliveries that are made on demand. The density of these on-demand deliveries varies throughout the day and week. Although the actual delivery tasks are not known ahead of time, the stochastic model g can be used to determine periods of relatively high activity versus periods of relatively low activity. The amount of time necessary to plan traffic control for periods of low activity is significantly less than those for high activity.

For example, consider a typical day. The night-time and early morning periods are characterized by low activity levels of some deterministically scheduled pharmaceutical and waste disposal deliveries and some deliveries due to emergency procedures. The traffic control problem during this period is relatively simple.

However the influx of deliveries necessary for the breakfast rush makes the traffic control problem much more difficult. Given the combinatorial nature of the route planning problem, deliberation time for this period is significant. We can take advantage of this situation by beginning to plan for the breakfast rush period during the night. While this involves considering many more reachable states than if we waited until morning, the additional time available for computation gained makes up for the increased computational difficulty. Furthermore, the pattern of peak and non-peak hours is fairly regular throughout the week. While the night-time and breakfast periods vary greatly in both problem complexity and problem parameters, the breakfast period on Tuesday is of the same difficulty level as the breakfast period Wednesday even if the actual combination of vehicles and delivery tasks varies. This regularity can be exploited in allocating on-line deliberation time between the night-time and breakfast tasks.

6 Discussion

We have focused on the specific issues involved in the construction of the response planning approach to embedded planning and the role of anticipating computational demands in providing time-critical responsiveness. There are many other issues and extensions of this work not included in this treatment. We briefly touch upon some here. In [10] we touch upon some others.

The response planning approach takes a very restricted view of possible strategies for Γ by enforcing domain regularity. As we have mentioned previously there are more flexible options available for the construction of Γ . In particular we may better handle opportunism and irregular events by composing Γ as a conditional or stochastic strategy. Certain events may trigger different strategy branches.

Given that processing time is dependent upon the actual problem instance, the time allocated to a decision procedure may not produce the desired quality. Zilberstein [22] introduces the concept of on-line monitoring to deal with these situations. Additionally, the concept of dynamically adjustable performance profiles and profiles conditioned on intervening states that pass during deliberation extend the dynamic properties of response planning.

It is interesting to think about multiprocess extensions of our two process, deliberative and reactive, architecture for response planning. In particular we can envision an architecture of multiple deliberative processes and multiple reactive tables. The multiple reactive tables may correspond to multiple agents, each table controlling the behavior of a unique agent. Multiple planners may compete for the control of each reactive table.

Finally the complexity of deliberation scheduling/compilation of anytime algorithms can be characterized for different classes of performance profiles. Zilberstein [22] and Boddy and Dean [4, 1] provide some results along these lines.

7 Related Work

Simon and Kadane [21] consider the case of allocating time to search in which deliberation time is quantized into chunks of fixed, though not necessarily constant, size. Etzioni [7] considers a similar model in which the deliberative chunks correspond to different problem solving methods. Russell and Wefald [18] describe a general approach in which deliberation is considered as a sequence of *inference steps* leading to the performance of an action. They explicitly compute the expected value of different sequences of inference steps using a technique similar to Howard's method for computing the expected value of information [12].

There are a number of approaches similar to the approach of using anytime decision procedures described in this paper. Anytime algorithms and various deliberation scheduling problems are described in [4, 1] and Chapter 8 of [5]. The advantages of the anytime algorithm approach include an ability to make use of any amount of time available, robust behavior in the presence of unexpected interruption, and simplifying the problem of optimal or near-optimal deliberation scheduling. Horvitz [11] uses what he calls flexible computations to allocate computational resources at run time. Lesser [16] uses a similar notion for solving time-critical problems. Liu *et al.* [20] uses the term *imprecise computation* in the context of real-time operating systems to refer to a computation that has both a necessary and an optional component. Zilberstein [22] discusses how anytime algorithms can be composed to perform more complicated computations.

Additionally, there have been a variety of planning systems that are related to the problem. Georgeff's *procedural reasoning system* [8] was designed for on-line use in evolving situations, but it simply executes user-supplied procedures rather than constructing plans of action on its own. Systems for synthesizing plans in stochastic domains, such as those by Drummond and Bresina [6], and Kushmerick, Hanks and Weld [14] do not directly address the problem of generating plans given time and quality constraints. Lansky [15] has developed planning systems for deterministic domains that exploit structural properties of the state space to expedite planning. Ow *et al.* [17] describe some initial efforts at building systems that modify plans incrementally. Neither Lansky nor Ow *et al.*'s systems deal with uncertainty and Lansky's system cannot handle concurrent planning and execution.

Acknowledgements

This work was supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601 and by the Air Force and the Advanced Research Projects Agency of the Department of Defense under Contract No. F30602-91-C-0041.

References

- [1] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings IJCAI 11*, pages 979–984. IJCAI, 1989.
- [2] Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*. Morgan-Kaufmann, Los Altos, California, 1989.
- [3] Thomas Dean. Decision-theoretic planning and markov decision processes. In preparation, 1994.
- [4] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-88*, pages 49–54. AAAI, 1988.
- [5] Thomas Dean and Michael Wellman. *Planning and Control*. Morgan Kaufmann, San Mateo, California, 1991.
- [6] Mark Drummond and John Bresina. Anytime synthetic projection: Maximizing the probability of

- goal satisfaction. In *Proceedings AAAI-90*, pages 138–144. AAAI, 1990.
- [7] Oren Etzioni. Tractable decision-analytic control. In Brachman et al. [2], pages 114–125.
- [8] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings AAAI-87*, pages 677–682. AAAI, 1987.
- [9] M. Gopal. *Modern Control System Theory*. Halsted Press, New York, 1985.
- [10] Lloyd Greenwald and Thomas Dean. Solving time-critical decision-making problems with predictable computational demands. In *Second International Conference on AI Planning Systems*, 1994.
- [11] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, 1987.
- [12] Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.
- [13] R. E. Kalman, P. L. Falb, and M. A. Arbib. *Topics in Mathematical System Theory*. McGraw-Hill, New York, 1969.
- [14] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. Unpublished Manuscript, 1993.
- [15] Amy L. Lansky. Localized event-based reasoning for multiagent domains. *Computational Intelligence*, 4(4), 1988.
- [16] Victor R. Lesser, Jasmina Pavlin, and Edmund Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, 1988.
- [17] P. S. Ow, S. F. Smith, and A. Thiriez. Reactive plan revision. In *Proceedings AAAI-88*. AAAI, 1988.
- [18] Stuart J. Russell and Eric H. Wefald. Principles of metareasoning. In Brachman et al. [2], pages 400–411.
- [19] Marcel J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings IJCAI 10*, pages 1039–1046. IJCAI, 1987.
- [20] 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*, pages 12–19. IEEE, 1989.
- [21] Herbert A. Simon and Joseph B. Kadane. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.
- [22] Shlomo Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, University of California at Berkeley, 1993.