Deliberation Scheduling for Time-Critical Scheduling in Stochastic Domains

Lloyd Greenwald, Thomas Dean

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-94-35
September 1994

Deliberation Scheduling for Time-Critical Scheduling in Stochastic Domains

Lloyd Greenwald* Thomas Dean*
lgg@cs.brown.edu tld@cs.brown.edu
Department of Computer Science
Brown University, Box 1910, Providence, RI 02912

Abstract

In this work we explore the use of deliberation scheduling for allocating computation among competing decision procedures in solving time-critical scheduling problems in stochastic domains. We present a two-level deliberation scheduling solution. At the higher level deliberation scheduling is used to partition infinite horizon scheduling problems temporally and sequence and allot computation time among the resulting problem instances. At the lower level deliberation scheduling is used to combine phases of a decision procedure for solving each problem instance. Our approach extends the work of Dean, Kaelbling, Kirman and Nicholson on planning under time constraints in stochastic domains to handle more complicated scheduling problems. We borrow from operations research in applying bottleneck-centered scheduling heuristics to improve initial policies and make use of Monte Carlo simulation for selectively constructing partial policies in large state spaces. Additionally, we employ a variant of Drummond's situated control rules to constrain the space of possible actions.

1 Introduction

In [10] we describe a scheduling and execution architecture and an iterative decision procedure for time-critical scheduling in stochastic domains. In this paper we develop the *deliberation scheduling* components of that approach. Deliberation scheduling is the process of allocating computation time among competing decision procedures to explicitly account for the costs and benefits of computational delays.

---

*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.

Scheduling in stochastic domains refers to the assignment of activities to resources over time such that certain constraints are met; despite the uncertainty caused by uncontrollable events. Time-critical activities have time-based constraints such as a deadline by when an activity must be performed. These domains can be modeled by stochastic processes with very large state and action spaces and uncertainty in predicting future states. A scheduling solution that is designed to respond in a timely manner to all possible states is called a *policy*.

Complementary work [9, 4] describe deliberation scheduling approaches to planning problems. However, there are fundamental differences between traditional planning domains and more combinatorial scheduling domains. When generating policies in planning domains, the large state space issue may be addressed by considering alternative actions (trajectories) within a restricted state space. The large action spaces inherent in scheduling problems makes intractable exhaustive search techniques such as policy iteration. All possible trajectories cannot be considered even within a restricted scheduling window. Instead, we employ Monte Carlo simulation on a stochastic model of the environment to construct partial policies for a subset of reachable states and provide default reflexes to handle the remaining states. We then apply bottleneck-centered scheduling heuristics [1] to improve initial policies. Additionally, we employ a variant of Drummond's [6] situated control rules to constrain the space of possible actions. Monte Carlo simulation addresses both the large action space and the large state space issues.

A scheduling system embedded in a dynamic environment may need to respond in a timely manner to sequences of dynamic events that cause changes to the state of the environment. Each new state may be considered a unique problem instance or a group of states may jointly be considered as a problem instance. Deliberation scheduling involves both partitioning the scheduling problem into problem instances and allocating computation time among decision procedures for the problem instances. In stochastic domains the scheduling system has at best probabilistic knowledge of future states. A stochastic model of the environment enables the system to predict future states and construct problem instances based on these predictions.

Deliberation scheduling requires some measure of the anticipated computational demands in solving problem instances in order to allocate processing time. Deliberation scheduling is aided 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* [3, 2] because they can output some result, of varying quality, at any time.

Our solution to scheduling in stochastic domains involves two levels of processing, each of which employs deliberation scheduling. At the lower level are the actual decision procedures for computing policies for particular problem instances. These decision procedures are composed of phases (anytime algorithms). Allocating computation across phases of a decision procedure is a deliberation scheduling problem often referred to as the compilation of anytime algorithms [15]. At the higher level, deliberation scheduling combines decision procedures for a predicted sequence of problem instances to generate a complete policy that optimizes some measure of global performance.

2 Scheduling and Execution

In this section we outline an architecture for on-line interaction of scheduling and execution. We also present a way to model the environmental dynamics as a stochastic process. These topics are treated in more detail in [9, 10].

We model the stochastic scheduling domain as a state space S made up of random variables. To represent the dependence of state on time we represent the state space as $S^{(t)} = \{X_1^{(t)}, X_2^{(t)}, \ldots, X_n^{(t)}\}$. In addition to the state space S we have an action space A. Each element $a \in A$ represents an assignment of activities to resources. Assignments have extent in time. We represent an assignment at a particular time step t as $a^{(t)}$ and the set of possible assignments at time step t to be $A^{(t)}$.

In general the effect of taking action $a^{(t)} \in A^{(t)}$ in state $s^{(t)} \in S^{(t)}$ is dependent upon the trajectory of states of the dynamical system from a base start state $H^{(t)} = \{s^{(0)}, s^{(1)}, \ldots, s^{(t)}\}$. $\Pr(s^{(t+1)}|a^{(t)}, H^{(t)})$ denotes the conditional probability of achieving state $s^{(t+1)}$ by taking action $a^{(t)}$ from the trajectory $H^{(t)}$. In some domains we can simplifies matters by making the *Markov assumption* which states that all the information in the trajectory is represented in state $s^{(t)}$ *i.e.,* $\Pr(s^{(t+1)}|a^{(t)}, H^{(t)}) = \Pr(s^{(t+1)}|a^{(t)}, s^{(t)})$.

In some domains a further simplification can be achieved by taking advantage of independence among state variables. In the case in which all state variables are independent, $\Pr(s^{(t+1)}|a^{(t)}, s^{(t)})$ can be computed as a product of the probabilities of the state variables. In the case in which all state variables are independent and correspond to uncontrollable events (*i.e.,* are independent of actions)

$$\Pr(s^{(t+1)}|a^{(t)}, s^{(t)}) = \Pr(X_1^{(t+1)}|X_1^{(t)}) \cdot \Pr(X_2^{(t+1)}|X_2^{(t)}) \cdots \Pr(X_n^{(t+1)}|X_n^{(t)})$$

Even state variables corresponding to controllable events may only be affected by a particular subset of actions. While we cannot expect complete independence of state variables we do expect scheduling domains to consist of a combination of independent state variables and small sets of dependent state variables, with a large number of state variables corresponding to uncontrollable events. This stochastic model is used to predict forthcoming states of the dynamic environment.

In deterministic environments a schedule corresponds to a single controlled trajectory of states. However, in stochastic environments a scheduling solution must account for uncontrollable events on-line. In traditional scheduling systems a fixed schedule generated off-line is modified by human operators to account for deviations from the expected trajectory. In time-constrained environments on-line interaction is not feasible; the system must prepare in advance to respond to alternative states at any time step.

Figure 1 captures the essential components of our embedded scheduling and execution architecture; $x(t)$ is the state of the system at time t, $u(t)$ is the action taken at time t by the composite scheduling 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 deliberative component Γ. Due to the combinatorics involved in deliberative processing there is typically a delay Δ between when a state triggers the deliberative process and when the resulting policy is available for execution. The remaining components of this figure are discussed shortly.

If we can use a stochastic model to predict the state of the environment for time t, $x(t)$, at some point in the past, $t' < t$, then Γ may begin constructing a policy for time t at time t' (in response to state $(x(t'))$. Deliberation scheduling deals with the problem of determining an appropriate t' at which to begin processing for time t.

Γ is an on-line process that performs three disparate functions. It uses the underlying environment model g to predict reachable states in upcoming time

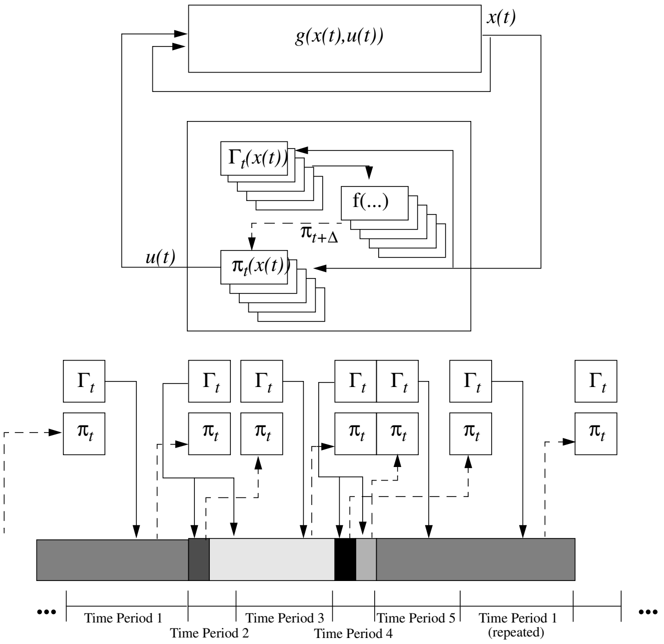

Figure 1: Embedded scheduling model and example

windows (problem instances); it executes domain specific decision procedures to construct policies for problem instances; and, it reasons about its use of time based on processing requirements, predictions and cost measures. We divide Γ conceptually into two components corresponding to two levels of processing. A set of decision procedures f that perform prediction of reachable states and construction of policies and a deliberation scheduling component Γ, that reasons about known properties of f and a given cost function and allocates computation time to f in order to provide the best possible performance. In Section 4.2 we present the decision procedures of f as composed of anytime phases that make use of deliberation scheduling.

The relationship between Γ, π and f is made explicit in Figure 1 and is summarized in the following procedure. At each time step t_j:

1. determine state $s_j = x(t_j)$ of environment g;
2. execute policy indicated by π_j to determine response to state s_j;
3. consult Γ to determine any new calls to f

4. if previous calls to f have terminated successfully since last time step, update π for those time steps by storing the constructed policies;

In Figure 1 we see Γ calling f during time period 1 to construct a policy for time period 2. We further note that just prior to time period 2, f updates π with its constructed policy. This policy is then in effect until the end of time period 2 at which point the policy for time period 3 takes effect. We show a new call to Γ at the beginning of each time period for simplicity. In general, Γ can initiate a new call to f in response to any state.

As described, Γ senses the state at each time step and performs deliberation scheduling to determine how to allocate on-line computation time to f to satisfy a given cost measure. For example, it may be optimal to solve the time period 4 problem beginning during time period 2 and then solve the time period 6 problem prior to the time period 5 problem. Deliberation scheduling can itself be a combinatorial problem. The implications of this for our scheduling solution is discussed in Section 4.

3 Deliberation Scheduling

In Section 4 we describe deliberation scheduling as it is employed in our solution to scheduling in stochastic domains. In this section we focus on some tools and features of deliberation scheduling in general, including the construction of conditional performance profiles and the compilation of anytime algorithms. Refer to [9, 15] for more detailed presentations.

Performance profiles are used to model the expected performance of different decision procedures for varying computation time allocations. These expectations are typically derived from past performance of the decision procedures applied to similar problem instances. A typical performance profile for an anytime algorithm is depicted in Figure 2.i, in which the expected quality of the decision procedure increases with diminishing returns (decreasing slope) as processing time increases. Compare this to a typical performance profile for a traditional algorithm depicted in Figure 2.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.

Conditional performance profiles are a way to represent the dependence of performance of a decision procedure on the quality of various input parameters in addition to processing time. For example, input quality may be

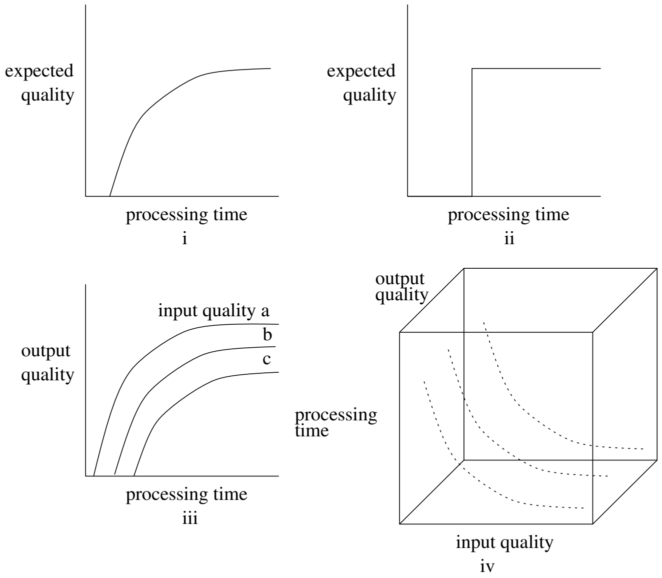

Figure 2: Performance Profiles

directly related to when computation of the decision procedure for that problem instance begins (due to prediction). Thus, computation start time may be an input parameter of a conditional performance profile. Other parameters include resource limitations and the target time window of the decision procedure. Figure 2.iii. depicts a conditional performance profile in which input quality takes on three distinct values. Figure 2.iv. depicts input quality as a separate dimension. Conditional performance profiles allow the system to reason about the context in which a decision procedure is used. For sequences of problem instances the input quality of a problem instance of one decision procedure in a sequence is often directly related to the output quality of prior decision procedures. In deliberation scheduling, reducing time allocated to any decision procedure may have non-local effects.

Deliberation scheduling is a special case of the general problem of *compiling* anytime algorithms [15] in which multiple algorithms with separate profiles are combined into a single module and profile. For example, we may compose anytime algorithms $g(x)$ and $h(y)$ into a new algorithm $f(g(x), h(y))$ that combines the properties of both to optimize combined performance.

The deliberation scheduling problem is 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. 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.

Zilberstein [15] shows that the general problem of compiling anytime algorithms is strongly NP-complete. Therefore, it is important to consider ways in which deliberation scheduling may be approximated, performed off-line or performed concurrently with policy generation.

4 Time Critical Scheduling

Our approach to time-critical scheduling in stochastic domains instantiates the components of the scheduling and execution architecture of Section 2. The high-level deliberation scheduling component Γ determines parameters for a sequence of calls to the low-level decision procedure f. These parameters include computation time allocated and the time window over which the decision procedure must compute a policy. In Section 4.1 we present methods for Γ to partition the state space temporally into problem instances associated with specific time windows. In Section 4.2 we present an algorithm for f that depends upon the partitioning and other allocation decisions made by Γ.

Determining parameters for constructing conditional performance profiles depends in part on the specifics of the deliberation scheduling task. For example, a scheduling horizon partitioned into dependent disjoint time windows (*i.e.,* the output from processing one window (policy) may be used as input for the next) requires conditional performance profiles that incorporate the expected output quality of the previous time window as an input. Other possible input parameters include the expected ratio of activities to available resources in the target time window as determined by previous output and the stochastic model of the domain; the expected difficulty of target time window; accumulated delay from previous decision procedures; number of initial constraints on processing within window (see Section 4.2); and allocated processing time.

We assume that conditional performance profiles can be compiled from past performance. In domains like air traffic control extensive data exists from which to construct profiles. However, some problems require gathering new data. In domains like robot navigation it may be necessary to construct profiles from relatively few trials. In this case much care must be taken in interpolating profiles from the data. In some domains a stochastic model of the environment may be used to simulate the system and gather data off-line. This approach is taken in [4]. Another option is to dynamically alter (or learn) profiles as the scheduling system is executing.

Deliberation scheduling with conditional performance profiles is a combinatorial problem. Performing deliberation scheduling on-line as part of component Γ may introduce unacceptable delays. In [9] we describe a way to take advantage of domain regularity to perform deliberation scheduling off-line. However, in some problems it is not possible to perform deliberation scheduling off-line. In particular, in the approach of [3] in which *precursor events* indicate sequences of problem instances only at run-time, deliberation scheduling must be performed on-line. In these cases the on-line cost of deliberation scheduling must be considered. See [5] for a relevant discussion.

4.1 Allocating Processor Time Across Infinite Scheduling Horizons

In constructing policies that change over time the high-level deliberation scheduler Γ must determine how to partition the infinite problem temporally into a series of fixed horizon problem instances (time windows). Although it is possible to split the problem in terms of a variable other than time (*e.g.,* groups of similar states), our scheduling and execution model favors time-based partitioning of the state space.

Figure 3 depicts the difference between partitioning the state space into disjoint or overlapping time windows. With disjoint independent time windows a distinct policy π is generated for each window. For dependent windows the arrows indicate the dependence of one window on the output of another. For example, the decision procedure generating the policy for time window B may use information about the policies generated for other time windows such as window A. Time windows need not be of fixed size.

The deliberation scheduler Γ must determine both the partitioning of the problem and the allocation of processing to decision procedures for the

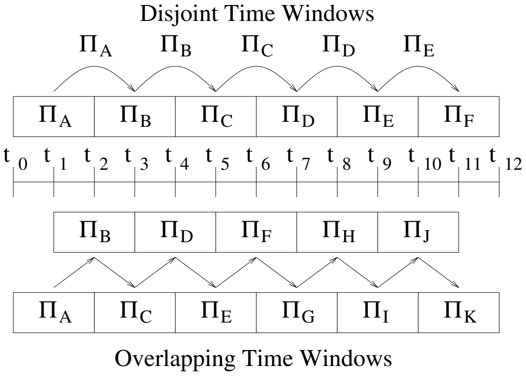

Figure 3: Partitioning infinite scheduling horizon into problem instances

resulting problem instances. For independent windows Γ may allocate processor time in any order. For dependent windows either processing time must be allocated in an order determined by the dependencies or Γ must make assumptions about the expected output quality of decision procedures.

As depicted in Figure 3, with overlapping time windows the same state (or time step) may be part of two or more windows. Decision procedures for overlapping time windows may take advantage of the output of decision procedures for other time windows. For two windows that overlap the decision procedure for the later window may take advantage of policies and constraints already constructed for states within its window. Conditional performance profiles for the high-level deliberation scheduling problem may include as parameters: policy and constraint quality from other time windows, processor allocation duration and location, and target time window duration and location.

In the special case of recurrent dynamics we can reduce the infinite scheduling horizon to a finite horizon. Deliberation schedules constructed off-line for that horizon may be repeated as long as the dynamics recur. The problem is more or less difficult depending on how far in advance of the target time window (problem instance) allocation of processing time for the decision procedure is allowed to begin and how soon it can be repeated for the same problem instance in a different period. If we let the size of the recurrent period be P then the deadline (beginning of the target time window) for decision procedure j in period p is $pP + d_j$. The earliest processing allocation time allowable for a decision procedure for a particular time window j is given by r_j. The sets j, d_j, and r_j define a deliberation scheduling problem. Three levels of difficulty are:

1. No decision procedure (for a particular time window) is allocated processor time before the beginning of the execution period that the target time window is a part of: $r_j = pP$. This guarantees that a deliberation schedule for one period can be repeated across all periods.
2. No decision procedure for a time window can be allocated processing time until the decision procedure for the same time window in the previous period has completed: $r_j = (p-1)P + d_j, p > 0$. To guarantee that a deliberation schedule constructed this way can be repeated we constrain the total schedule to take no more then P time and we constrain times to be allocated in pairs: $\sigma(t_i) = j \rightarrow \sigma(t_i + P) = j$.
3. Same as previous case except we choose a number of periods $m > 1$ and require $\sigma(t_i) = j \rightarrow \sigma(t_i + mP) = j$.

Case 2 makes use of slack time in the prior period to improve solutions. Case 3 is optimal for deterministic processing times assuming that a steady-state interval m can be found. Once we know m the problem is reduced to Case 2. If processing time is nondeterministic then we can at best get expected repeatability.

The above discussion focuses on off-line construction of deliberation schedules for Γ to execute on-line. However, in some domains it may be advantageous to reallocate processor time on-line in response to system dynamics. This corresponds to the *recurrent deliberation* model of [4].

4.2 Allocating Processor Time Among Anytime Phases

Our time-critical scheduling solution is completed by instantiating the prediction and decision procedure component f of the scheduling and execution architecture of Section 2. f performs the policy generation algorithm central to our solution. Our policy generation algorithm is a two-phase iterative algorithm that employs deliberation scheduling methods to allocate computation time across phases and iterations. These phases are briefly outlined here and described in detail in [10].

The first phase makes use of Monte Carlo simulation and constrained dispatch scheduling to define a policy within a given time window. The second phase analyzes the results of the first phase in order to detect areas for improvement (bottlenecks) within the policy and adds additional constraints on the action space in order to mitigate the bottlenecks and improve subsequent policies. The number of times the phases are iterated and the amount of time allocated to each iteration is determined by a deliberation scheduling procedure that attempts to optimize the policy generated given a fixed computation time allocation. The fixed computation time allocation is determined by Γ based on performance profiles provided by f.

The phase one dispatch scheduler determines local assignments of activities to resources for a given state using greedy rules such as first-come first-served, latest release date first or earliest deadline first. We augment the dispatch scheduler by borrowing from the work of Drummond [6]. For any state, a set of constraints limits the possible assignments. These limitations implicitly disable a subset of the action space. Constrained dispatch scheduling solves the problem of scheduling in the face of a large action space. We use Monte Carlo simulation to turn local scheduling decisions into policies executable under uncertainty. Monte Carlo simulation involves flipping an m-sided coin once for each independent subset of state variables within a state (where m is the number of discrete values the set can take) biased by the corresponding distributions. The result of this set of coin tosses is a next state and a probability of reaching that state.

In phase two we constrain bottleneck assignments from the phase one policy in order to move toward a more globally optimal solution. Our technique for propagating constraints is based on bottleneck-centered heuristics [1, 13], a form of opportunistic scheduling in which critical points of resource contention are located and used to identify and remove potential for negative interactions between activities. First, find a critical interaction (bottleneck) among activities; namely one which contributes directly to a high expected cost policy. In practice [13] critical interactions occur during time intervals in which there is a high demand/supply ratio for resources. Second, identify an assignment of activities to resources that, if constrained, will both reduce the bottleneck and guarantee that an improved policy exists that may be found through dispatch scheduling.

The deliberation scheduling task for this algorithm is equivalent to the *precursor deliberation* of [4]. Whether the phases should be iterated once or many times is determined by whether or not one phase of the algorithm "informs" the other phase. Bottleneck detection always "informs" dispatch scheduling. The deliberation scheduling problem is to determine the number of iterations and the allocation of processing time for each phase of each iteration. For k iterations we have $t_{\mathrm{TOT}} = t_{P1_1} + t_{P2_1} + \cdots + t_{P1_k} + t_{P2_k}$ [4]. Deliberation scheduling requires conditional performance profiles for dispatch scheduling that model policy improvement as a function of constraint quality, previous policy quality and processor allocation; and conditional performance profiles for bottleneck detection that model constraint quality with similar parameters.

The form of deliberation scheduling for f depends upon the method of deliberation scheduling of Γ. With disjoint, independent time windows the input quality for the initial iteration of dispatch scheduling uses the baseline of the conditional performance profile. However, if available, constraints and policies that are propagated across time windows are reflected in the input quality.

There is no single "best" dispatch scheduling rule to apply in phase one of our algorithm over all problems. The deliberation scheduling procedure of f may model alternative dispatch rules on varying problem instances and use this information to determine the appropriate rule for each iteration of each call to f. In this case deliberation scheduling must reason about profiles for each rule and choose the appropriate rule as well as the processor allocation for each iteration of processing.

If conditional performance profiles remain static for a given execution of the algorithm then deliberation scheduling for f may be performed off-line. In particular, we may use the conditional performance profiles to compile a combined anytime algorithm and associated profile that determines the optimal allocation of processing time among phases and the appropriate dispatch scheduling rules to employ. This profile may then be used by Γ to determine how to partition the problem temporally and allocate processing across windows to achieve some level of global performance.

5 Related Work

In [9] we describe a way to take advantage of domain regularity to perform deliberation scheduling off-line and execute Γ as a fixed deliberation strategy.

In this solution we assume that while we only have stochastic knowledge of the actual state trajectory, we have deterministic knowledge of the sequence of equivalence classes of problem instances. The work of Dean *et al.* [4] introduces a general approach to planning in stochastic domains and several models of deliberation scheduling.

We extend the work of [4] to handle more combinatorial scheduling problems and borrow from the bottleneck-centered heuristics developed by Adams *et al.* [1] and analyzed by Muscettola [13]. The approach described in this paper borrows from and extends the anytime projection approach of Drummond and Bresina [7, 6].

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 [7], and Kushmerick, Hanks and Weld [11] do not directly address the problem of execution in dynamic situations or generating plans given time and quality constraints. Lansky [12] has developed planning systems for deterministic domains that exploit structural properties of the state space to expedite planning. Smith *et al.* [14] describe some initial efforts at building systems that modify plans incrementally. Neither Lansky or Smith *et al.*'s systems deal with uncertainty and Lansky's system cannot handle concurrent planning and execution.

References

[1] Adams, J., Balas, E., and Zawack, D., The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science*, **34**(3) (1988) 391–401.

[2] Boddy, Mark and Dean, Thomas, Solving Time-Dependent Planning Problems, *Proceedings IJCAI 11, Detroit, Michigan*, IJCAII, 1989, 979–984.

[3] Dean, Thomas and Boddy, Mark, An Analysis of Time-Dependent Planning, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 49–54.

[4] Dean, Thomas, Kaelbling, Leslie, Kirman, Jak, and Nicholson, Ann, Deliberation Scheduling for Time-Critical Sequential Decision Making,

Proceedings of the Ninth Workshop on Uncertainty in AI, Washington, D.C., 1993, 309–316.

[5] Dean, Thomas and Wellman, Michael, *Planning and Control*, (Morgan Kaufmann, San Mateo, California, 1991).

[6] Drummond, Mark, Situated Control Rules, Brachman, Ronald J., Levesque, Hector J., and Reiter, Raymond, (Eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, (Morgan-Kaufmann, Los Altos, California, 1989).

[7] Drummond, Mark and Bresina, John, Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction, *Proceedings AAAI-90, Boston, Massachusetts*, AAAI, 1990, 138–144.

[8] Georgeff, Michael P. and Lansky, Amy L., Reactive Reasoning and Planning, *Proceedings AAAI-87, Seattle, Washington*, AAAI, 1987, 677–682.

[9] Greenwald, Lloyd and Dean, Thomas, Anticipating Computational Demands when Solving Time-Critical Decision-Making Problems, *The First Workshop on the Algorithmic Foundations of Robotics*, A. K. Peters, Boston, MA, 1994.

[10] Greenwald, Lloyd and Dean, Thomas, Monte Carlo Simulation and Bottleneck-Centered Heuristics for Time-Critical Scheduling in Stochastic Domains, *ARPI Planning Initiative Workshop*, 1994.

[11] Kushmerick, Nicholas, Hanks, Steve, and Weld, Daniel, An Algorithm for Probabilistic Planning, Unpublished Manuscript, 1993.

[12] Lansky, Amy L., Localized Event-Based Reasoning for Multiagent Domains, *Computational Intelligence*, **4**(4) (1988).

[13] Muscettola, Nicola, An Experimental Analysis of Bottleneck-Centered Opportunistic Scheduling, *Proceedings of the Second European Workshop on Planning, Vadstena, Sweden*, 1993.

[14] Ow, P. S., Smith, S. F., and Thiriez, A., Reactive Plan Revision, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988.

[15] Zilberstein, Shlomo, *Operational Rationality through Compilation of Anytime Algorithms*, PhD thesis, University of California at Berkeley, 1993.