Monte Carlo Simulation and Bottleneck-Centered Heuristics for Time-Critical Scheduling in Stochastic Domains

Lloyd Greenwald* Thomas Dean*
lgg@cs.brown.edu tld@cs.brown.edu
(401) 863-7662 (401) 863-7645
Department of Computer Science
Brown University, Box 1910, Providence, RI 02912

Abstract

In this work we extend the work of Dean, Kaelbling, Kirman and Nicholson on planning under time constraints in stochastic domains to handle more complicated scheduling problems. In scheduling problems the sources of complexity stem not only from large state spaces but from large action spaces as well. In these problems it is no longer tractable to compute optimal policies for restricted state spaces via policy iteration. We, instead, 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 this work we are interested in solving scheduling problems with time constraints in stochastic domains. Stochastic domains imply that there are events outside the system's control. Scheduling in this context refers to the assignment of activities to resources over time such that certain constraints are met. Time-critical activities have time-based constraints such as a deadline by when an activity must be performed. A scheduling solution attempts to adhere to time constraints despite the uncertainty caused by uncontrollable events. These problems can be modeled as stochastic processes with very large state and action spaces and uncertainty in predicting future states that result from controlled actions and/or uncontrollable events. Such problems cannot be handled by search techniques that require enumerating states in the state space or state trajectories in the corresponding phase space (the product of the state space and the set of times considered).

We develop a set of techniques that extend the work of Dean *et al.* [3] for solving time-critical planning problems. In that work a two phase iterative procedure is employed. The first phase determines a restricted subset of the state space on which to focus (called the *envelope*) and the second phase generates a *policy* mapping states to actions within this envelope. Focusing on a restricted subset of the state space allows for the tractable application of exhaustive search techniques such as policy iteration in domains in which the number of possible transitions from any state is limited. Both phases are designed as *anytime algorithms* [2] in that the quality of output of each

*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.

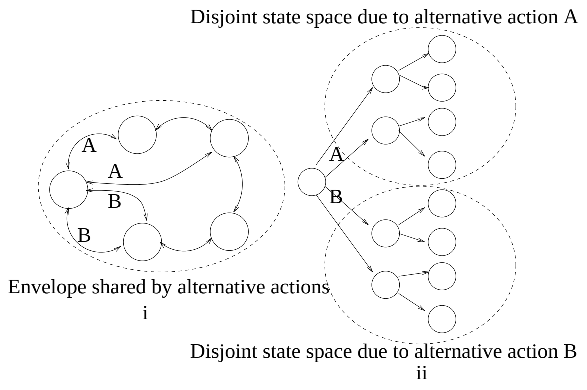
Disjoint state space due to alternative action A
A
A
B
B
A
B
Envelope shared by alternative actions
i
Disjoint state space due to alternative action B
ii

Figure 1: State spaces introduced by alternative actions in planning (i) and scheduling (ii) problems

phase is a function of the processing time allocated to it. *Deliberation scheduling* is employed to allocate on-line computation time between each phase and across iterations of both phases. This approach directly addresses uncertainty by modeling the environment as a stochastic automaton and constructing policies to account for alternative trajectories reachable from a given start state.

Consider the use of this approach in time-critical scheduling problems in stochastic domains. Specifically, the problem of scheduling planes and gates at a busy airport. Each arriving plane may be considered a job composed of two sequential operations. The first operation corresponds to unloading passengers at an arrival gate and the second operation corresponds to loading passengers at a departure gate. A stochastic process describes the arrival of planes at the airport and is affected by uncontrollable events such as weather. Stochastic processes also govern the processing requirements for operations at gates (modeling loading and unloading delays). Deadlines for the operations are determined by prespecified desired arrival and departure times. Other sources of uncertainty include gate closings (machine breakdowns). The optimization problem is to assign planes to gates at each time step to minimize some global measure of tardiness. In any given state there is, in general, one action for every possible assignment of planes to gates. And each action results in a distribution of next states as determined by the stochastic processes.

In order to address the fundamental differences between traditional planning domains and more combinatorial scheduling domains this work departs from [3] both in how we restrict the state space and how we generate policies within the restricted state space. In planning domains, performing policy generation in a restricted envelope of states addresses the large state space issue. Optimal policies may be generated by considering alternative actions (trajectories) within this restricted state space. However, in scheduling domains the combination of large action and state spaces imply that alternative actions from a given state can lead to disjoint state spaces. Therefore it is no longer meaningful to restrict attention to a specific set of states. When constructing policies, instead of focusing scheduling on an envelope of specific states, we restrict the space to a target time window. States of a scheduling problem are associated with specific time steps within the window. This allows searching for an optimal policy within the window even if the actual states accounted for in any iteration are completely different from the previous iteration. The difference between state spaces for planning and scheduling problems given alternative actions is depicted in Figure 1.

Even if we could salvage the notion of a restricted envelope of states for policy generation, the large action space inherent in scheduling problems makes intractable exhaustive search techniques such as policy iteration. Namely, because the large action space implies a large number of alternative transitions from any state. Given the large action space we cannot account for all possible trajectories 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 the most probable reachable states and provide default reflexes to handle the remaining states. We then borrow from Operations Research in applying bottleneck-centered scheduling heuristics [1] to improve initial policies. Additionally, we employ a variant of Drummond's [5] 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.

Deliberation scheduling, the process of allocating processor 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. 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 because they can output some result, of varying quality, at any time. The flexibility inherent in these decision procedures allows for a more flexible allocation of processing time to satisfy an overall performance criterion.

In this paper we focus on specific anytime decision procedures for policy generation and bottleneck detection. In [10] we explore the deliberation scheduling necessary to optimally allocate processing time between these decision procedures and across time windows.

2 Embedded Scheduling

In this section we outline our basic architecture for on-line interaction of scheduling and execution in stochastic domains and describe tools for modeling the stochastic domains. In [9, 11] this architecture is discussed in the context of planning. In Section 2.1 we describe this architecture in the context of scheduling. In Section 2.2 we model the stochastic domain.

2.1 Scheduling and Execution

In [9] we define *embedded planning* to be the problem of determining actions for an agent embedded in an uncertain environment with dynamics outside of the agent's control. The same description applies for scheduling in dynamic environments. A salient feature of these problems is that the system must react to changes in the environment in *real-time*, *i.e.,* the system must sense the state of the environment and act appropriately in a timely manner. There is rarely the luxury of waiting for a deliberative process 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 consequences for the agent.

An agent experiences the dynamics of the environment as a trajectory of states. For simplicity we assume a sequence of time steps with a state associated with each time step. We can then divide time into discrete time 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).

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

Figure 2: Model for embedded planning and control

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 alternative states at any time step. A complete policy may be constructed by combining partial policies that result from applying decision procedures to restricted subsets of the state space. Alternatively, a complete policy may be constructed through the combination of on-line decision procedures and off-line construction of *reflexes* that provide default responses in unlikely or unanticipated states. A policy may be used as part of a real-time control system that responds to changes in the environment.

Figure 2 captures the essential components of embedded planning (scheduling) 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 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. 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 [4, 8, 12].

We motivate a specific form of this general approach in [11]. This architecture 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. This architecture forms the backdrop for the scheduling techniques discussed in this paper. In future work we will describe how the framework of [9] can be used to focus deliberation in scheduling domains with particular regularity properties.

2.2 Scheduling in Stochastic Domains

We model the stochastic scheduling domain as a state space S made up of state variables $S = \{X_1, X_2, \ldots, X_n\}$. Each state variable is a random variable taking on a set of (discrete or continuous) values. For example state variable X_p may represent the location of plane p drawn from a discrete set of possible locations. One distinguished state variable is time. To represent the dependence of state on time we may represent the state space as $S^{(t)} = \{X_1^{(t)}, X_2^{(t)}, \ldots, X_n^{(t)}\}$ *e.g.,* $X_p^{(t)}$ may represent the location of plane p at time t. If each state variable takes on a discrete set of values then the size of the state space at any given time step $|S^{(t)}|$ is a product of the number of possible values for each variable for that time step. This quantity is exponential in the number of state variables.

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 in the scheduling domain. 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 the gate assignment problem an action consists of an assignment of planes to gates at some time step. At any given time step (or state corresponding to that time step) there is, in general, one action for every possible assignment of planes to gates. If we take the problem to have g gates and p planes then the number of actions in any state is $|A^{(t)}| = O(\frac{p!}{(p-g)!})$ which is roughly $O(p^g)$.

In order to predict the effect of actions taken in particular states (at particular time steps) we need to model the joint probabilities of states and actions. 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.,* $\Pr(s^{(t+1)}|a^{(t)}, s^{(t)}) = \Pr(s^{(t+1)}|a^{(t)}, H^{(t)})$.

Even with the simplification of the Markov assumption we are still dealing with the necessity of manipulating very large transition matrices. The transition matrix for the corresponding Markov decision process is a mapping from $S^{(t)} \times A^{(t)} \times S^{(t+1)}$ to $[0, 1]$ at each time step t. Dean *et al.* [3] were able to explicitly represent restricted envelopes of the state space as automata (in order to reduce the size of the state space) and perform exhaustive search on these restricted automata in order to generate optimal policies. The ability to do this requires a limited number of transitions per state (small action space) in the automata. As demonstrated above the action space for scheduling problems is large and as discussed in Section 1 the corresponding ability to construct restricted envelopes is diminished. In Section 3.1 we employ Monte Carlo simulation combined with a greedy local scheduling technique to deal with these issues.

In some scheduling domains we can simplify the corresponding transition matrices by taking advantage of independence among state variables. Consider the case in which all state variables are independent. In this case we have

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

which can be much more compactly represented and efficiently computed. Consider the case in which all state variables are independent and correspond to uncontrollable events (*i.e.,* are independent of actions as well). In this case we have

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

Even state variables corresponding to controllable events may only be affected by a particular subset of actions and have no change due to others. In some cases the transition matrix may be *homogeneous i.e.,* independent of the actual time step and/or be represented by a *stationary* distribution *i.e.,* independent of any previous values of the state variable.

Consider a stochastic process describing the arrival of planes at an airport. This process may be independent of *any* action taken at the airport or state of the airport itself. If we consider each plane to have a separate state variable describing its arrival then the probability that plane p will arrive at location l at time t can be denoted $\Pr(X_p^{(t)} = l)$ independent of any action or any other state variables.

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. The extent to which these expectations apply determines how efficiently Monte Carlo simulation can be applied to a particular domain.

Monte Carlo simulation involves flipping an m-sided coin once for each independent subset of state variables (where m is the number of discrete values the set can take) biased by the corresponding distributions. This approach is presented in Section 3.1. Other simulation approaches may be substituted depending upon the nature of the scheduling domain.

We make this discussion concrete by describing the state and action spaces for the gate assignment problem mentioned in Section 1. In this formulation consider a state to codify the status of each job (plane) and machine (gate) at a particular point in time. The state variables include the location and status of each plane and the availability of each gate. A plane's status indicates how far along it is in completing its activity (*i.e.,* the remaining operations and adjusted expected processing times) and the values of its stochastic parameters (*e.g.,* arrival time). A gate's status includes whether it is open or closed and, if occupied, what plane is currently assigned to it. In a full implementation, plane state variables could be created on demand or used to represent 'virtual' planes; also additional variables would be added for factors that might influence flight time. For simplicity assume that planes always land before their scheduled arrival times, so that delays can be attributed to there being too few gates or those gates poorly assigned.

In a sense, each state itself corresponds to a scheduling problem. The action taken in a state may either be part of a temporal trajectory of optimal actions or a greedy schedule. In any given state only a subset of the action space is available (*e.g.,* a plane that has not landed cannot be assigned to any gate). The available actions are constrained by job parameters such as release dates.

3 Time-Critical Scheduling with Anytime Algorithms

In this section we describe a technique for time-critical scheduling in stochastic domains. As mentioned in Section 2 the basic architecture we employ involves on-line interaction between scheduling and execution. In that architecture policies are explicitly given temporal indexes to indicate that they may change over time. An integral part of a scheduling solution in stochastic domains is the mechanism to determine how to adjust policies over time. Therefore, there are two levels of processing to consider in designing the on-line scheduling component Γ. At the lower level are the actual decision procedures for computing policies that solve the scheduling problem for a particular target time period. Since time is a state variable this target time period defines a restricted state space for which to generate partial policies. At the higher level, meta-level deliberation scheduling combines these decision procedures for differing time periods to generate a complete policy that optimizes some measure of global performance.

The deliberation scheduler determines parameters with which to call the anytime algorithms that make up our decision procedures. These parameters include computation time allocated and the target time window over which the decision procedure must compute a policy. There is considerable flexibility in determining how to divide the state space temporally in order to dynamically adjust policies over time. A deliberation scheduler may allocate time to decision procedures that compute policies independently for disjoint time windows (*i.e.,* each decision procedure constructs a policy from scratch for a given set of time steps). This approach is taken in [9, 11]. Alternatively, the deliberation scheduler may employ a sliding time window in which the target time windows of each decision procedure overlap and make use of policies computed by prior decision procedures. At the higher level, computation time is allocated among the decision procedures for disjoint or overlapping time windows. In [10] we discuss in detail these alternative deliberation scheduling approaches. In this section we present low-level anytime decision procedures that are consistent with either approach.

As a starting point we assume that the deliberation scheduler calls a decision procedure with a specific target time window and resource allocation. Additionally, we allow for other input parameters such as initial partial policy. These issues will be discussed shortly.

One unique aspect of our technique is that the low-level decision procedures are themselves made up of phases, each of which is itself an anytime algorithm. We apply deliberation scheduling a second time to allocate time between the different phases of processing at this level. In this section we discuss separately each phase of processing and discuss their interaction in more detail under the context of deliberation scheduling and compilation of anytime algorithms in [10].

Our general approach to policy generation consists of the following two phases:

1. Monte Carlo Simulation and Constrained Dispatch Scheduling

2. Bottleneck Detection and Constraint Propagation

The first phase defines a policy within a given time window using greedy scheduling rules and constraints on the action space in conjunction with Monte Carlo simulation. 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. Each of these phases is implemented as an anytime algorithm whose output quality varies as a function of its input quality and allocated processing time. Deliberation scheduling is employed to allocate computation to each phase separately to optimize their combined performance. The phases are combined in an iterative series of sequences.

3.1 Monte Carlo Simulation and Constrained Dispatch Scheduling

In this section we design the first phase of our two-phase iterative anytime decision procedure for policy generation. We assume that a high-level deliberation scheduler has determined a particular target time window for which to generate a policy and allocated a fixed amount of processing time. Processing time is allocated under the expectation that our combined two-phase anytime algorithm will make the best use of this time to optimize the policy for the given target time window. In allocating processing time the high-level deliberation scheduler uses expectations of output quality for the decision procedure as a function of processing time in similar situations in the past. A second-level deliberation scheduler determines how much time to allocate between each phase (and any subcomponents of the phase) and across iterations of both phases.

As discussed in Section 2.2 one fundamental characteristic of scheduling domains is a very large action space. In a sense, each state corresponds to a local scheduling problem in which an action must be chosen from a large set of available actions (of course, these local scheduling problems are not independent). Not all actions are available in every state. In any given state only a subset of the action space is available (*e.g.,* a plane that has not landed cannot be assigned to any gate). The available actions are additionally constrained by job parameters such as release dates.

Many of the heuristic job shop scheduling procedures described in the literature are based on "priority dispatching" rules; greedy rules for choosing the operation to be scheduled next for each machine. These include first-come first-served, latest release date first, earliest deadline first, etc., In our context a dispatch scheduler chooses from among the available (unconstrained) actions in a given state. Note that the dispatch scheduler need not enumerate all possible actions in that state. In practice a dispatch scheduler determines the individual assignments of activities to resources. Combined, these assignments comprise a single action. But, the dispatch scheduler need not search in the action space itself to determine the action chosen. If preemption of activities is allowed the dispatch scheduler may choose assignments for all unconstrained machines. However, if preemption is not feasible (as in the case of planes assigned to gates) then the dispatch scheduler only determines assignments for resources which are free or become free in that state.

We augment the dispatch scheduler by borrowing from the work of Drummond [5] on situated control rules. For any state, a set of constraints limits the assignments possible for the dispatch scheduler. These limitations on assignments implicitly disable a subset of the action space. In Section 3.2 we discuss how these constraints on dispatch scheduling are derived. Constrained dispatch scheduling solves the problem of scheduling in the face of a large action space. Note that it is not necessary to pick a single dispatch scheduling rule that applies to all time windows. Rather our deliberation scheduling procedure may model varying dispatch rules on varying problem instances and use this information to determine the appropriate rule for the current state and/or window.

Merely performing constrained dispatch scheduling to derive locally optimal actions along a nominal trajectory of states is not enough to adequately address the policy generation problem. The combined effect of executing local scheduling solutions in each state does not, in general, lead to an optimal global schedule. However, methods for computing optimal global schedules by exhaustive search are not tractable given the large state and action spaces of scheduling problems.

In our compromise solution we use Monte Carlo simulation to turn local scheduling decisions into policies executable under uncertainty. We then iterate on these initial policies by constraining bottleneck assignments in order to move toward a more globally optimal solution. The quality of the solution varies depending upon the amount of computation resources allocated by the high-level deliberation scheduler. In the rest of this section we discuss the process of generating policies from constrained dispatch scheduling and Monte Carlo simulation and discuss bottleneck detection and constraint propagation in Section 3.2.

Because of the large state space it is not tractable to generate a policy by computing actions via dispatch scheduling for each reachable state in any time window. We use Monte Carlo simulation to explore the state space and reduce the probability that an unplanned-for state (one in which only reflexes are available) will be encountered in practice. Monte Carlo simulation addresses both the large action space and the large state space issues in generating policies.

As discussed in Section 2.2 each state is composed of a set of independent subsets of state variables. Many of these subsets consist of a single state variable whose transition matrix depends only on its current state and the action chosen by the dispatch scheduler. Other subsets consist of a single state variable corresponding to an uncontrollable event whose transition is independent of the action itself. The remaining subsets' transition matrices may be efficiently represented and computed if the number of state variables in each is small. Monte Carlo simulation involves flipping an m-sided coin once for each independent subset of state variables (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.

The phase one policy generation anytime decision procedure has two incarnations. First, it may be called without any prior policy or set of constraints on the action space. In this case it must derive an initial policy. This corresponds to an initial call to a decision procedure that computes a policy for a time window from scratch, independent of the policies used in other time windows.

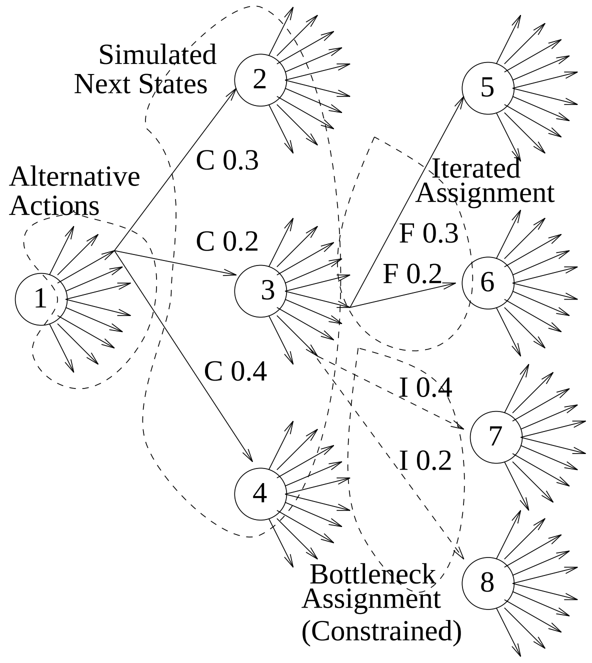

Figure 3: Scheduling algorithm applied to two step time window

Second, it may be called with a given policy and set of constraints. In this case it must improve on the given policy by iterating on trajectories stemming from any new constraints on assignments. This corresponds to an initial call in the case of overlapping time windows or to subsequent calls on the same time window in either case.

We summarize the entire phase one policy generation anytime decision procedure.

Given:

- a target time window
- an initial state or distribution over states at the onset of the time window
- an allocation of processing time
- (optional) an initial policy
- (optional) a set of constraints on certain states within the target time window

Compute:

```
While allocated time remaining:
    For each state on list (either with new constraints or initial set
    of states or added by Monte Carlo simulation):
       - determine action based on applying constrained dispatch
         scheduler to state
       - perform Monte Carlo simulation x times to derive at most x
         next-states (where x is determined by the second-level
         deliberation scheduler)
       - add states to list as long as they are within time window
    Perform phase two bottleneck detection and constraint propagation
      (for an amount of time determined by the second-level
      deliberation scheduler)
```

Figure 3 depicts the mechanics of our two phase algorithm. From initial state 1 the dispatch scheduler chooses action C from among the unconstrained actions. Monte Carlo simulation is then executed to achieve three alternative next states reachable from state 1 through action C with varying probabilities. These states are then added to the list and the process is repeated. From state 3 action I is chosen and, through simulation, leads to states 7 and 8. After phase one has completed, bottleneck detection is invoked on the resulting policy and determines that action I in state 3 contains a bottleneck assignment. The action space for state 3 is constrained against action I (and other actions derived from the bottleneck assignment). The two phases are then repeated with the new constraint added. This time the dispatch scheduler chooses action F to take from state 3 and, subsequently, states 5 and 6 are reached through simulation. The resulting policy is an improvement (in expected value) on the previous policy.

Note that in the above discussions we describe Monte Carlo simulation as being part of policy generation. This is true if we consider the decision procedure as performing dispatch scheduling in advance on the most probable states as determined by Monte Carlo simulation. However, this need not be the only interpretation of the technique. We can also think of the dispatch scheduler combined with constraints on states as implicitly defining a policy. Monte Carlo simulation is then a means to examine the most probable trajectories that may be achieved in executing this policy, for use in bottleneck detection.

3.2 Bottleneck Detection and Constraint Propagation

In Section 3.1 we discuss how constraints are used to restrict the action space during local dispatch scheduling. In this section we discuss how to derive constraints that will iteratively lead to improved global policies. Specific mechanisms for determining which constraints to add are domain specific. We present one representative example method based on work in Operations research. Other techniques may also be effectively employed to provide constraints.

The two phase anytime algorithm for time-critical scheduling that we sketch above is most effective if constraints are only added when improved policies are expected. Constraint propagation procedures that demonstrate this property are called *monotonic*. Monotonic constraint propagation guarantees that a locally optimal schedule will be found. However, it does not guarantee that a globally optimal schedule will be achieved. In order to guarantee global optimality without explicit exhaustive search over the action space we must be able to retract as well as add constraints. This extension to our two phase approach is not discussed any further here. Note that in domains without preemption it is reasonable to constrain all assignments in a given state.

The technique we describe for propagating constraints is based on the bottleneck-centered heuristics [1, 15] of Operations Research. Bottleneck-centered heuristics are 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. Bottleneck-centered heuristics have demonstrated their effectiveness in deployed scheduling solutions [15]. We employ bottleneck-centered heuristics through two main steps. First, find a critical interaction (bottleneck) among activities; namely one which contributes directly to a high expected cost policy. In practice [15] 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. This step corresponds to a monotonic constraint propagation method.

In order to apply the bottleneck-centered heuristic in our context we must describe ways to implement both steps. There are at least two alternatives for detecting bottlenecks. We can either look for trajectory-based bottlenecks or global bottlenecks. In the former case, we analyze the current policy for the trajectory with the highest expected cost (if the optimization problem is to minimize cost). This is considered a bottleneck trajectory. We then analyze each scheduling action along the trajectory to determine which contributed most to the bottleneck. In the latter case of global bottlenecks, both steps are combined into a single step. First we define a local measure for the additional expected cost accrued in a given state. We then penalize each scheduling assignment active in that state with the local cost. The assignments with the highest total expected cost are considered bottleneck assignments. These assignments are only constrained if, in doing so, a policy of smaller expected cost may be achieved. This phase is implemented as an anytime decision procedure in that in either case we repeat the procedure for one or more bottlenecks and/or assignments depending upon how much time is allocated to this phase of processing.

We describe the latter method in slightly more detail. The focus of this method may be considered a form of *value determination*. In this process an immediate value is computed for each state based on a local measure of expected increase in tardiness. We then compute the value of each assignment by calculating the value of the state in which the assignment is made and the values of subsequent states in which the assignment remains active. In some cases we may wish to include a factor γ so that states further into the future may have a diminished contribution to the value of an assignment.

In scheduling domains in which a tree-structured stochastic matrix is expected, value determination for all assignments may be computed during a breadth-first search of the policy generated during phase one. The contribution to the value of each assignment from each state depends upon the probability that that state will be reached (as determined by Monte Carlo simulation) and the diminished cost of that state based on the factor γ. The assignments with the highest expected cost are then constrained only if, in doing so, a policy of less expected cost may be achieved. We may test for this condition by simulating the effect of the constraint and retracting it immediately if an improved policy is not achieved.

In general, we cannot guarantee fairness with this approach. Although we guarantee that constraint propagation leads to policy improvement, we cannot guarantee that the schedule will treat all activities fairly. Additional mechanisms must be provided to analyze the tradeoffs between global optimization and local fairness. Particularly in the face of infinite scheduling horizons in which a bottleneck assignment can delay assigning a particular plane indefinitely.

4 Related Work

In this work we present a heuristic solution to scheduling in stochastic domains subject to time constraints. We use a hierarchy of anytime decision procedures guided by deliberation scheduling. Deliberation scheduling and compilation of anytime algorithms for this work are discussed in [10]. In [9] we discuss a general architecture for reasoning in stochastic domains in which the domain is assumed to obey certain regularity constraints. The techniques that we have explored here are complementary to that work and their integration will be discussed in future work.

This work extends the work of Dean *et al.* [3] to handle more combinatorial scheduling problems. We have borrowed from the Operations Research literature on bottleneck-centered heuristics developed by Adams *et al.* [1] and analyzed by Muscettola [15]. We augment this work by borrowing from the work of Drummond [5]. The approach described in this paper borrows from and extends the anytime projection approach of Drummond and Bresina [6].

There have been a variety of planning systems that are related to the problem. Georgeff's *procedural reasoning system* [7] 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 [13] do not directly address the problem of execution in dynamic situations or generating plans given time and quality constraints. Lansky [14] has developed planning systems for deterministic domains that exploit structural properties of the state space to expedite planning. Smith *et al.* [16] 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] Dean, Thomas and Boddy, Mark, An Analysis of Time-Dependent Planning, *Proceedings AAAI-88, St. Paul, Minnesota*, AAAI, 1988, 49–54.

[3] Dean, Thomas, Kaelbling, Leslie, Kirman, Jak, and Nicholson, Ann, Planning With Deadlines in Stochastic Domains, *Proceedings AAAI-93, Washington, D.C.*, AAAI, 1993, 574–579.

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

[5] 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).

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

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

[8] Gopal, M., *Modern Control System Theory*, (Halsted Press, New York, 1985).

[9] Greenwald, Lloyd and Dean, Thomas, Anticipating Computational Demands when Solving Time-Critical Decision-Making Problems, *submitted to the Workshop on the Algorithmic Foundations of Robotics*, 1994.

[10] Greenwald, Lloyd and Dean, Thomas, Deliberation Scheduling for Time-Critical Scheduling in Stochastic Domains, *submitted to UAI-94*, 1994.

[11] Greenwald, Lloyd and Dean, Thomas, Solving Time-Critical Decision-Making Problems with Predictable Computational Demands, *submitted to the Second International Conference on AI Planning Systems*, 1994.

[12] Kalman, R. E., Falb, P. L., and Arbib, M. A., *Topics in Mathematical System Theory*, (McGraw-Hill, New York, 1969).

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

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

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

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