An Analysis of Time-Dependent Planning

Thomas Dean* and **Mark Boddy**
Department of Computer Science
Brown University
Box 1910, Providence, RI 02912

Abstract

This paper presents a framework for exploring issues in *time-dependent planning*: planning in which the time available to respond to predicted events varies, and the decision making required to formulate effective responses is complex. Our analysis of time-dependent planning suggests an approach based on a class of algorithms that we call *anytime* algorithms. Anytime algorithms can be interrupted at any point during computation to return a result whose utility is a function of computation time. We explore methods for solving time-dependent planning problems based on the properties of anytime algorithms.

1 Introduction

Time-dependent planning is concerned with determining how best to respond to predicted events when the time available to make such determinations varies from situation to situation. In order to program a robot to react appropriately over a range of situations, we have to understand how to design effective algorithms for time-dependent planning. In this paper, we will be concerned primarily with understanding the properties of such algorithms, and providing a precise characterization of time-dependent planning.

The issues we are concerned with arise either because the number of events that the robot has to contend with varies, and, hence, the time allotted to deliberating about any one event varies, or the observations that allow us to predict events precede the events they herald by varying amounts of time. The range of planning problems in which such complications occur is quite broad. Almost any situation that involves tracking objects of differing velocities will involve time-dependent planning (*e.g.*, vehicle monitoring [Lesser and Corkill, 1983; Durfee, 1987], signal processing [Chung *et al.*, 1987], and juggling [Donner and Jameson, 1986]). Situations where a system has to dynamically reevaluate its options [Fox and Smith, 1985; Dean, 1987] or delay committing to specific options until critical information arrives [Fox and Kempf, 1985] generally can be cast as time-dependent planning problems.

To take a specific example, consider the problem faced by a stationary robot assigned the task of recognizing and intercepting or rerouting objects on a moving conveyor belt. Suppose that the robot's view of the conveyor is obscured at some point by a partition, and that someone on the other side of this partition places objects on the conveyor at irregular intervals. The robot's task requires that, between the time each object clears the partition and the time it reaches the end of the conveyor, it must classify the object and react appropriately. We assume that classification is computationally intensive, and that the longer the robot spends in analyzing an image, the more likely it is to make a correct classification. One can imagine a variety of reactions. The robot might simply have to push a button to direct each object into a bin intended for objects of a specific class; the time required for this sort of reaction is negligible. Alternatively, the robot might have to reach out and grasp certain objects and assemble them; the time required to react in this case will depend upon many factors. One can also imagine variations that exacerbate the time-dependent aspects of the problem. For instance, it might take more time to classify certain objects, the number of objects placed on the conveyor might vary throughout the day, or the conveyor might speed up or slow down according to production demands. The important thing to note is, if the robot is to make optimal use of its time, it should be prepared to make decisions in situations where there is very little time to decide as well as to take advantage of situations where there is more than average time to decide. This places certain constraints on the design of the algorithms for performing classification, determining assembly sequences, and handling other inferential tasks.

Traditional computer science concerns itself primarily with the complexity and correctness of algorithms. In most planning situations, however, there is no one correct answer, and having the right answer too late is tantamount to not having it at all. In dealing with potentially intractable problems, computer scientists are sometimes content with less than guaranteed solutions (*e.g.*, answers that are likely correct and guaranteed computed in polynomial time (Monte Carlo algorithms), answers that are guaranteed correct and likely computed in polynomial time (Las Vegas algorithms), answers that are optimal within some factor and computed in polynomial time (approximation algorithms). While we regard this small concession to reality as encouraging, it doesn't begin to address the problems in time-dependent planning. For many planning tasks, polynomial performance is not sufficient; we need algorithms that compute the best answers they can in the time they have available.

Planning is concerned with reasoning about whether to act and how. Scheduling is concerned with reasoning about

*This work was supported in part by the National Science Foundation under grant IRI-8612644 and by an IBM faculty development award.

when to act having already committed to act, and plays an important role in planning. In job-shop scheduling, if it is possible to suspend, and later resume, a job, then many otherwise difficult problems become trivial [Graham *et al.*, 1977; Bodin and Golden, 1981]. Such (preemptive) scheduling problems are somewhat rare in real job shops given that there is often significant overhead involved is suspending and resuming jobs (*e.g.*, traveling between workstations or changing tools), but they are considerably more common with regard to purely computational tasks (*e.g.*, suspending and resuming Unix processes). In many scheduling problems, each job has a fixed cost and requires a fixed amount of time to perform; spending any less than the full amount yields you nothing. In planning, we are interested in the computational tasks of deciding upon appropriate reactions. If the decision procedures for computing appropriate reactions are preemptible and provide better answers depending upon the time available to deliberate, then the time-dependent planning problem is considerably simplified. In the next section, we examine this claim in more detail.

2 Time-Dependent Planning

Temporal notation:

- a set of event types: $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \ldots\}$.
- a set of time points: $\mathcal{T} = \{t_1, t_2 \ldots\}$.
- a set of actual events: $\mathcal{A} = \{e_1, e_2 \ldots\}$; $\forall e \in \mathcal{A},\ (\mathsf{begin}(e) \in \mathcal{T}) \wedge (\mathsf{end}(e) \in \mathcal{T})$.
- a function type from $\mathcal{A}$ to $\mathcal{E}$; $\forall e \in \mathcal{A},\ \mathsf{type}(e) \in \mathcal{E}$.
- a function distance from $\mathcal{T} \times \mathcal{T}$ to the reals ($\Re$); $\forall t_1, t_2 \in \mathcal{T},\ \mathsf{distance}(t_1, t_2) \in \Re$.
- a precedence relation $\prec$ on $\mathcal{T}$ such that $\forall t_1, t_2 \in \mathcal{T},\ (\mathsf{distance}(t_1, t_2) > 0) \Rightarrow (t_1 \prec t_2)$.

An instance of a time-dependent planning problem consists of:

- a set of events, $\mathcal{C} = \{c_1, c_2, \ldots c_n\} \subseteq \mathcal{A}$, corresponding to conditions demanding a response from the robot.
- a set of events, $\mathcal{O} = \{o_1, o_2, \ldots o_m\} \subseteq \mathcal{A}$, corresponding to observations made by the robot.
- a set of reactions, $\mathcal{R} \subseteq \mathcal{E}$, corresponding to the types of actions that might be taken by the robot in reaction to a predicted event; ϕ indicates the null reaction.
- a function react from $\mathcal{R}$ to $\Re$, where $\mathsf{react}(\varepsilon)$ corresponds to the time required to carry out a reaction of type ε; $\mathsf{react}(\phi) = 0$.
- a function herald from $\mathcal{C}$ to $\mathcal{O}$, where $\mathsf{herald}(c)$ corresponds to the earliest event in $\mathcal{O}$ that would enable the robot to predict c.
- a function utility from $\mathcal{C} \times \mathcal{R}$ to $\Re$; $\forall c \in \mathcal{C},\ \mathsf{utility}(c, \phi) = 0$.
- a function response from $\mathcal{C}$ to $\Re$, where $\mathsf{response}(c)$ corresponds to the time between having the information available to predict c, and c itself. We will assume that a robot's reaction to c must be carried out prior to c, and, hence, we have $\mathsf{response}(c) = \mathsf{distance}(\mathsf{end}(\mathsf{herald}(c)), \mathsf{begin}(c))$.

We can divide the robot's time as follows:

prediction time: the time required to predict an event given the information available. For the robot to *predict* c, the robot must "know" $\mathsf{type}(c)$ and $\mathsf{begin}(c)$[1]. To simplify our analysis, we will assume that prediction time is fixed, and, hence, that it can be factored out of $\mathsf{response}(c)$.

deliberation time: the length of the maximum interval of time during which the robot must commit to a specific reaction if a reaction is to be carried at all. We note that during any given interval of time there may be many deliberative processes competing for the use of the same computational resources.

reaction time: the time required to react to a given event. Reactions differ in terms of how long they take to carry out. If we assume that reaction time is always negligible (*i.e.*, $\forall \varepsilon \in \mathcal{R},\ \mathsf{react}(\varepsilon) \approx 0$), then the deliberation time for an event c is just $\mathsf{response}(c)$.

For each $c \in \mathcal{C}$, we assume that the robot has some decision procedure for inferring how to react to c. Each decision procedure, when allocated some amount of time, returns some $\varepsilon \in \mathcal{R}$ corresponding to its best guess about how to react[2]. We define γ from $\mathcal{C} \times \Re^+$ to $\mathcal{R}$ so that for each $c \in \mathcal{C}$ and positive real number δ, $\gamma(c, \delta) = \varepsilon$ where ε is the robot's best guess about how to react to c having spent δ in deliberation. In describing the robot's ability to cope with time-dependent planning problems, we will be interested in the composite function $\mathsf{utility}(c, \gamma(c, \delta))$[3]. Throughout this paper, we will assume that there is exactly one processor dedicated to deliberation.

In the following, we consider several classes of time-dependent planning problems corresponding to different restrictions on the robot's ability to compute reactions to predicted events. There are two issues we have to consider. The first concerns how much work the robot has to do in order to determine how best to allocate the time available for deliberation. We'll refer to this as the *deliberation scheduling* problem. Apart from deciding what to think about when, the rest of the robot's intellectual capacities are fixed. The time required for deliberation scheduling will not be factored into the overall time allowed for deliberation. For the techniques we are concerned with, we will demonstrate that deliberation scheduling is simple, and, hence, if the number of predicted events is relatively small, the time required for deliberation scheduling can be considered negligible. The second issue is concerned with how

[1] A more realistic model might require only that the robot "knows" ε and t such that ε is a supertype of $\mathsf{type}(c)$ and $t \prec \mathsf{begin}(c)$.

[2] If the decision procedure takes a fixed amount of time to formulate an answer, we can assume that it outputs ϕ until that fixed time is up.

[3] In our analysis, we assume that there is a wide range of utilities associated with the various reactions, but this stipulation is not critical. The utility of the outcome can be constant; it is sufficient for our analysis to apply that the *expected* utility have a sufficiently wide range.

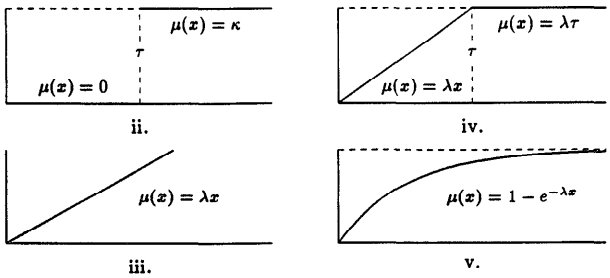

Figure 1: Performance profiles for decision procedures

well the robot can hope to do, assuming that it has made the best decision about how to allocate its deliberation time. In order to talk about making the best decision, we have to introduce some method of comparing decisions.

If we assume that the robot can only perform one action at a time, then a *solution* to a time-dependent planning problem consists of a mapping $\psi : \mathcal{C} \rightarrow \mathcal{A}$ and a single-processor schedule for the set, $\{\psi(c) \mid c \in \mathcal{C}\}$, subject to

- $\forall c \in \mathcal{C}$, $\mathsf{type}(\psi(c)) \in \mathcal{R}$ (appropriately typed)
- $\forall c \in \mathcal{C}$, $\mathsf{end}(\psi(c)) \prec \mathsf{begin}(c)$ (appropriately timed)
- $\forall c' \neq c''$, $(\mathsf{end}(\psi(c')) \prec \mathsf{begin}(\psi(c''))) \vee (\mathsf{end}(\psi(c'')) \prec \mathsf{begin}(\psi(c')))$ (no overlap)

We define a cost function on solutions:

$$\mathsf{cost}(\psi) = -\sum_{c \in \mathcal{C}} \mathsf{utility}(c, \mathsf{type}(\psi(c))).$$

If Ψ is the set of all solutions, then we are interested in that $\psi \in \Psi$ that minimizes cost.

Now, we introduce five classes characterizing the robot's capability to compute solutions as a function of the time available for deliberation. The first class includes the other four; it is mentioned only because later we will consider problems that do not fit within this class. Figure 1 illustrates performance profiles for instances of each class (ii) through (v). In the following, let $\mu(x, y) = \mathsf{utility}(x, \gamma(x, y))$.

i. monotonic improvement:

$$\forall c \in \mathcal{C},\ \forall \delta, \epsilon \in \Re^+,\ \mu(c, \delta) \leq \mu(c, \delta + \epsilon).$$

ii. one-shot improvement:

$$\forall c \in \mathcal{C},\ \exists \tau, \kappa \in \Re^+,\ \mu(c, \delta) = \begin{cases} 0 & \text{for } 0 \leq \delta < \tau, \\ \kappa & \text{for } \tau \leq \delta < \infty. \end{cases}$$

iii. linear improvement and unbounded utility:

$$\forall c \in \mathcal{C},\ \exists \lambda \in \Re^+,\ \mu(c, \delta) = \lambda * \delta.$$

iv. piecewise linear improvement and bounded utility:

$$\forall c \in \mathcal{C},\ \exists \tau, \lambda \in \Re^+,\ \mu(c, \delta) = \begin{cases} \lambda * \delta, & 0 \leq \delta < \tau, \\ \lambda * \tau, & \tau \leq \delta < \infty. \end{cases}$$

v. diminishing returns:

$\forall c \in \mathcal{C}$, $\exists f$, $\mu(c, t) = f(t)$ such that f is monotonic increasing, continuous, and piecewise differentiable, and $\forall x, y \in \Re^+$ such that $f'(x)$ and $f'(y)$ exist, $x < y \Rightarrow (f'(y) \leq f'(x))$.

For each time t, the robot has some set of events, $\mathcal{L}_t$, that it needs to formulate reactions for. If $\forall c \in \mathcal{C}\ \exists o \in \mathcal{O}, o = \mathsf{herald}(c)$, then $\mathcal{L}_t = \{c \mid \mathsf{end}(\mathsf{herald}(c)) \preceq t \prec \mathsf{begin}(c)\}$[4]. In deliberation scheduling, the robot has to determine how best to budget its time among the events in $\mathcal{L}_t$. The object is to generate a solution ψ that minimizes cost given the deliberative capabilities of the robot. Every time that a new event is predicted the robot will have to reformulate its strategy for allocating the available deliberation time. The problem of constructing an optimal strategy for class (ii) capabilities is solvable in pseudo-polynomial time [Graham *et al.*, 1977]. Classes (iii) and (iv) are special cases of class (v). In the following, we describe a polynomial-time algorithm for constructing optimal strategies for class (v) capabilities. Classes (iii) and (iv) can be handled using simpler algorithms, which we discuss as well. We assume that $\forall \varepsilon \in \mathcal{R}$ $\mathsf{react}(\varepsilon) = 0$. Let $f_c(x) = \mu(c, x)$.

iii. Let λ_c be the slope of the linear function f_c. An optimal strategy is to deliberate on that c in $\mathcal{L}_t$ such that λ_c is maximal. At any point, the processor will be working on the event c such that the change in $f_c(\delta)$ with respect to δ is maximized over $\mathcal{L}_t$; working on any other event will result in a higher cost.

iv. We can use the fact that for any $\delta \geq \tau_c$, $\mu(c, \delta)$ is a constant function, and that for $\delta < \tau_c$, $\mu(c, \delta)$ is linear. This allows us to define a simple analytic, rather than iterative, algorithm for deliberation scheduling, which lack of space precludes our including in this paper.

v. We define a sequence $c_1, \ldots c_k$ such that, for $1 \leq i \leq k$, $c_i \in \mathcal{L}_t$ and $\mathsf{begin}(c_i) \prec \mathsf{begin}(c_{i-1})$. Let $\Lambda_j = \{c_i \mid 1 \leq i \leq j\}$, and initially set $\mathsf{alloc}(c) = 0$ for all $c \in \mathcal{L}_t$. The algorithm in Figure 2 shows how to compute a strategy for scheduling the deliberation processor from t until the last event in $\mathcal{L}_t$. The strategy assumes time slicing with a fixed smallest allocation of processing time Δ. This strategy remains in effect as long as no new events are predicted. For any $\epsilon > 0$, there is some $\Delta > 0$ such that the above strategy is optimal within ϵ. A more complicated approach that involves solving a system of simultaneous equations enables us to generate an optimal strategy analytically without time slicing.

Most existing planning systems have class (ii) capabilities. Such systems require a fixed amount of time to compute the same cost solution no matter how much time is available. If there is more time available, the system deliberates about something else or does nothing; if there

[4] In a more realistic model, membership in the set $\mathcal{L}_t$ might change in response to many factors. For instance, an observation $o \in \mathcal{O}$ might serve to eliminate some previously predicted $c \in \mathcal{L}_t$. While this is not allowed in the current model, it can be added without affecting our analysis.

```
for i = 1 to k
    start ← (if i = k, then t, else begin(c_{i+1}))
    stop  ← begin(c_i)
    for j = 0 to ⌊distance(start,stop)/Δ⌋
        choose c ∈ Λ_i s.t. ∀e ∈ Λ_i, (c ≠ e) ⇒ (f'_c(alloc(c)) ≥ f'_e(alloc(e)))
        deliberate on c from (start+jΔ) to min((start+(j+1)Δ),stop)
        alloc(c) ← alloc(c) + min((start+(j+1)Δ),stop) − (start+jΔ)
```

Figure 2: Deliberation scheduling algorithm for class v.

is less time available, the system carries out some default reaction. Planning problems for which such capabilities appear satisfactory are somewhat rare. Consider a distribution function describing the time available to respond to c given all situations in which c is predicted to occur. If the variance is small, the robot's decision procedures are optimized for the mean, and for any situation in which c is predicted to occur it is unlikely that there will be other events to contend with, then class (ii) capabilities will perform well. Class (ii) capabilities will also do well if the time required to determine a reaction that is within ϵ of optimal is small compared to the mean. Classes (iii) through (v) will outperform class (ii) capabilities in a wide range of natural planning problems in which the variance is significant, and the potential gain from increased deliberation and the number and importance of other events to contend with at any given moment varies substantially.

The above analysis can be extended in a myriad of ways. In the following sections, we will consider just a few issues that we think particularly interesting. Before leaving this section, it seems worthwhile to reiterate some of the assumptions that have been made, and comment on their relevance.

1. prediction time can be factored out of response time,
2. predicted events come to pass at the expected time,
3. there exist separate decision procedures for each event type,
4. the cost of preemption is negligible,
5. cost is the inverse sum of the individual utilities,
6. there exist decision procedures of the sort described in (iii–v),
7. utility is a monotonically increasing function of deliberation time, and
8. reaction time is negligible.

We claim without further argument that the first four can be relaxed to suit many realistic planning problems. Assumption 5 is tantamount to assuming that coordinating responses buys you nothing. Relaxing assumption 5 is problematic; if all responses are complexly interdependent, then the kinds of decision procedures and the methods whereby they communicate become quite complicated. The requisite analysis is similarly complicated and is relegated to a companion paper currently in preparation. Throughout the rest of this paper, we will continue to assume that responses are independent. Section 3 considers the existence of decision procedures that satisfy the requirements underlying assumption 6. Section 4 considers issues in relaxing assumptions 7 and 8.

3 Anytime Algorithms

In the previous section, we described a class of time-dependent planning problems characterized by there being a variety of reactions to predicted events and a range of response times occurring in practice. In all of the scenarios that we looked at, the robot made use of decision procedures having the property that the utility of the reactions suggested by these procedures monotonically increase over time. Our analysis indicates that being forced to rely upon procedures with a fixed one-time improvement can lead to poor performance. Furthermore, our analysis suggests a class of algorithms that could significantly improve performance. The most important characteristics of these algorithms are that (i) they lend themselves to preemptive scheduling techniques (*i.e.*, they can be suspended and resumed with negligible overhead), (ii) they can be terminated at any time and will return some answer, and (iii) the answers returned improve in some well-behaved manner as a function of time. It is the last two of these two characteristics that really distinguishes the algorithms we are interested in from more traditional algorithms, and. in recognition of this, we christen them *anytime* algorithms.

There are large classes of algorithms that satisfy the characteristics described above. The study of methods for iterative approximation is a large and active area in numerical analysis [Tompkins and Wilson, 1969; Hageman and Young, 1981]. Algorithms for heuristic search, in particular those employing variable lookahead and fast evaluation functions, can easily be cast as anytime algorithms [Pearl, 1985]. Symbolic processing in general can be viewed as the manipulation of finite sets (of bindings, constraints, entities, etc.). The behavior of iterated functions over finite sets is the subject of the study of discrete iterations [Robert, 1986]. The analysis of discrete iterations is closely tied to work on connectionist models [Hinton and Sejnowski, 1983], the study of cellular automata [Farmer *et al.*, 1984], and the design of VLSI systems [Mead and Conway, 1980]. Our preliminary literature search uncovered a large body of research on the properties of anytime algorithms and their application to control problems.

```
plan(T) ←
  setof(E,predict(T,E,_),Events),
  allocate(Events,[]),plan(t̂).

act(T) ←
  setof(E,predict(T,E,T),Events),
  execute(Events,[]),act(t̂).

allocate([],Processes) ←
  schedule(Processes).

allocate([Event|L],Processes) ←
  decision_process(Event,P),
  allocate(L,[P|Processes]).

execute([],Reactions) ←
  delegate(Reactions).

execute([Event|L],Reactions) ←
  decide(Event,R), execute(L,[R|Reactions]).

start ←
  concurrently(plan(t̂), act(t̂)).

predict(T,Event,T+Δ) ←
  herald(Precursors,Event,Δ),
  holds(Precursors,T).
```

Figure 3: Simple interpreter for time-dependent planning

4 A Framework for Time-Dependent Planning

In this section, we describe a simplified framework for employing decision procedures implemented as anytime algorithms. We use a specification language based loosely on PROLOG to capture both the inferential and the procedural aspects of our approach. Let $\hat{t}$ refer to the current time as indicated by the robot's clock. Figure 3 provides a listing for a simple PROLOG program[5] that implements a time-dependent planner.

We assume that `plan` and `act` run concurrently on separate processors, and require some small amount of time for each invocation neglecting recursive calls. The procedure `decision_process(E,P)` returns in `P` a process set up to run an anytime algorithm uniquely associated with the event `E`. Such a process is terminated when the begin point of its associated event passes. The procedure `schedule` implements deliberation scheduling, using a strategy such as those described in Section 2 to allocate processor time to existing processes. The procedure `predict(T,E,T+Δ)` takes a time `T` and returns in `E` an event predicted to occur at time `T+Δ`. In the interpreter of Figure 3, an event has to be repeatedly predicted in order to continue deliberating on an appropriate reaction for that event. The procedures `herald` and `holds` might be implemented using a temporal database of the sort described in [Dean, 1987]. Execution is handled in a manner similar to deliberation: the procedure `decide(E,R)` looks up the process associated with the event `E` and then uses that process to return in `R` the best guess for a reaction given the time spent in deliberation, and the procedure `delegate` turns over the execution of these reactions to some independent processor. In the remainder of this section, we relax two of the assumptions imposed in Section 2, and see how the simple planner in Figure 3 is complicated as a result.

If $\exists\tau\ \forall\varepsilon \in \mathcal{R}$, $\mathsf{react}(\varepsilon) = \tau$ (*i.e.*, all reactions take exactly the same time), the task of determining when to stop deliberating and start acting is trivial. In Figure 3, we assume that $\tau = 0$. In the following, we consider three simple cases in which $\mathsf{react}(\varepsilon)$ is allowed to vary; let $\hat{\varepsilon}_c$ be the robot's current best guess for how to react to c. For the first case, suppose that $\mathsf{utility}(\varepsilon, c)$ is completely independent of $\mathsf{react}(\varepsilon)$, and that we have an accurate estimate of $\mathsf{react}(\varepsilon)$ for all $\varepsilon \in \mathcal{R}$. In this case, deliberation scheduling is performed as it was described in Section 2 with the exception that, if $\mathsf{react}(\hat{\varepsilon}_c)$ changes significantly during deliberation, then the deadline for reacting to c will have changed requiring the scheduler to determine a new strategy for $\mathcal{L}_t$. For a particular $c \in \mathcal{C}$, the deadline for reacting to c can be obtained by subtracting $\mathsf{react}(\hat{\varepsilon}_c)$ from $\mathsf{begin}(c)$. As a second case, suppose that $\mathsf{utility}(\varepsilon, c)$ is completely determined by $\mathsf{react}(\varepsilon)$ (*e.g.*, $\mathsf{utility}(\varepsilon, c) = \lambda\mathsf{react}(\varepsilon)$ with constant λ). In this case, if $\mathsf{utility}(\varepsilon, c)$ is well behaved (*e.g.*, functions of class (v) as described in Section 2), then deliberation scheduling and deciding when to act can be handled as in the first case. Finally, if $\mathsf{alloc}(c)$ corresponds to the amount of time allocated to deliberating about c, and $-\mathsf{utility}(\varepsilon, c) = \mathsf{react}(\varepsilon) + \mathsf{alloc}(c)$ (*i.e.*, all that matters is minimizing the sum of deliberation time and reaction time), the problem is still easy. Since $-\frac{d\mathsf{utility}}{d\mathsf{alloc}} = 1 + \frac{d\mathsf{react}}{d\mathsf{alloc}}$, we stop deliberating when $\frac{d\mathsf{react}}{d\mathsf{alloc}} \geq -1$. Similar analyses yield solutions to a number of additional special cases. It should be noted, however, that relaxing the assumption that reaction time is negligible can make time-dependent planning arbitrarily complex.

Another assumption we might relax is that utility increases monotonically with deliberation time. Utility may decrease over time when reaction time is non-zero and interacts with deliberation time, as discussed above. Utility may also decrease as the world changes over time rendering the information obtained from previous observations obsolete. Given that the general problem of scheduling sensing is far too complex to be addressed here, we pose a simpler problem: suppose that our decision procedures depend critically upon the information available when they are first started.

The basic scenario is as follows. An event c is predicted to occur at time t. At some point, $\mathsf{commit}(c)$, between t and $\mathsf{begin}(c)$, a set of sensor readings, $\mathsf{data}(c)$, is collected—assume that the time required for collection is negligible. In the time between $\mathsf{commit}(c)$ and $\mathsf{begin}(c)$, the robot has to formulate an appropriate reaction to c based upon extrapolations from $\mathsf{data}(c)$. The accuracy of the extrapolations performed during deliberation de-

[5]The construct `setof(X,P,R)` repeatedly invokes the procedure `P` in which `X` appears unbound. The resulting bindings are returned as a list in `R`.

pend upon distance(commit(c), begin(c)). If the extrapolations are perfect, then the expected utility of the computed reaction will depend solely upon deliberation time. Given that the extrapolations are not perfect, the problem of deliberation scheduling—in particular, the problem of determining commit(c)—is somewhat more complicated than the problems we considered in Section 2. By representing utility as a function f of deliberation time and distance(commit(c), begin(c)), we can make use of the partial derivatives of f to implement a variant of the scheduling algorithm described for class (v) functions.

5 Conclusion

In this paper, we define the problem of time-dependent planning, and argue that a wide variety of planning problems satisfy our definition. Our formulation of the problem suggests a solution in terms of *anytime algorithms*: algorithms that can be interrupted and resumed with little overhead, that can provide increasingly good answers over a range of response times, and that, therefore, provide solutions over a range of response times[6]. We demonstrate that under certain assumptions (listed at the end of Section 2), anytime algorithms can be coordinated in an effective strategy for handling time-dependent planning problems.

Our preliminary literature search indicates that the class of decision procedures that can be implemented as anytime algorithms is quite large. We are currently working on a framework that relies on a library of generic anytime algorithms: general-purpose algorithms that can be composed to build specialized decision procedures. This framework generates bounded-depth decision trees that serve to combine the results of several generic anytime algorithms using simple operators. One outstanding problem involves generating appropriate performance profiles for such composite anytime algorithms. If this problem can be satisfactorily resolved, we believe that our framework will provide a practical approach to building high-performance planning systems for time-dependent applications.

References

[Bodin and Golden, 1981] L. Bodin and B. Golden. Classification in vehicle routing and scheduling. *Networks*, 11:97–108, 1981.

[Chung *et al.*, 1987] Jen-Yao Chung, Jane W.S. Liu, and Kwei-Jay Lin. Scheduling periodic jobs using imprecise results. Technical Report UIUCDCS-R-87-1307, University of Illinois at Urbana-Champaign Department of Computer Science, 1987.

[Dean, 1987] Thomas Dean. Intractability and time-dependent planning. In Michael P. Georgeff and Amy L. Lansky, editors, *The 1986 Workshop on Reasoning About Actions and Plans*, pages 245–266. Morgan-Kaufman, 1987.

[Donner and Jameson, 1986] Marc D. Donner and David H. Jameson. A real-time juggling robot. IBM Research Report RC 12111 (54549), IBM, 1986.

[Durfee, 1987] Edmund H. Durfee. A unified approach to dynamic coordination: Planning actions and interactions in a distributed problem solving network. Technical Report 87-84, University of Massachusetts at Amherst Department of Computer and Information Science, 1987.

[Farmer *et al.*, 1984] D. Farmer, T. Toffoli, and S. Wolfram. *Cellular Automata.* North Holland, 1984.

[Fox and Kempf, 1985] B.R. Fox and K.G. Kempf. Opportunistic scheduling for robotics assembly. In *IEEE International Conference on Robotics and Automation*, pages 880–889, 1985.

[Fox and Smith, 1985] M.S. Fox and S. Smith. Isis: A knowledge-based system for factory scheduling. *Expert Systems*, 1:25–49, 1985.

[Graham *et al.*, 1977] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Proceedings Discrete Optimization*, 1977.

[Hageman and Young, 1981] L.A. Hageman and D.M. Young. *Applied Iterative Methods.* Academic Press, 1981.

[Hinton and Sejnowski, 1983] G.E. Hinton and T.J. Sejnowski. Optimal perceptual inference. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 448–453, 1983.

[Horvitz, 1987] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 AAAI Workshop on Uncertainty in Artificial Intelligence*, 1987.

[Lesser and Corkill, 1983] V.R. Lesser and D.D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4:15–33, 1983.

[Mead and Conway, 1980] C.A. Mead and M.A. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[Pearl, 1985] Judea Pearl. *Heuristics.* Addison-Wesley, 1985.

[Robert, 1986] F. Robert. *Discrete Iterations: A Metric Study.* Springer-Verlag, 1986.

[Tompkins and Wilson, 1969] C.B. Tompkins and W.L. Wilson. *Elementary Numerical Analysis.* Prentice-Hall, 1969.

[6] Horvitz [Horvitz, 1987] discusses the problem of reasoning under resource constraints in a more general framework. In particular, he makes a more extensive use of probability and utility theory than we have space for here. His conclusions concerning the algorithms and architectures appropriate for effective reasoning under resource constraints are entirely compatible with the discussion in this paper.