Probabilistic Temporal Planning with Uncertain Durations

Mausam and **Daniel S. Weld**
Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
{mausam,weld}@cs.washington.edu

Content areas: Planning, Markov Decision Processes

Abstract

Few temporal planners handle both concurrency and uncertain durations, but these features commonly co-occur in real-world domains. In this paper, we discuss the challenges caused by concurrent, durative actions whose durations are uncertain. We present five implemented algorithms, including $\Delta\text{DUR}_{\text{prun}}$, a planner guaranteed to find the optimal policy. An empirical comparison reveals that $\Delta\text{DUR}_{\text{exp}}$, our fastest planner, obtains orders of magnitude speed-up compared to $\Delta\text{DUR}_{\text{prun}}$— with little loss in solution quality. Importantly, our algorithms can handle probabilistic effects in addition to stochastic durations, and they are effective even when duration distributions are multi-modal.

1. Introduction

Recent progress in temporal planning raises hopes that this technology may soon apply to a wide range of real-world problems, but several problems remain. In particular, *concurrent actions* with *stochastic durations* characterize many real-world domains. While these issues have independently received some attention, only three systems have addressed both concurrency and duration uncertainty in concert. These seminal planners handle extremely expressive languages but have several limitations.

Tempastic's Generate & Test approach (Younes & Simmons 2004a) sacrifices completeness and optimality guarantees. Generalized Semi MDP's (GSMDPs) phase-type approximations (Younes & Simmons 2004b) may contain many more state variables than the original problem, and scalability has not been demonstrated. Neither Prottle (Little, Aberdeen, & Thiebaux 2005) nor Tempastic account for varied distributions of possible durations, and Prottle plans for a finite horizon — and thus for an acyclic state space.

In this paper, we present five novel planning algorithms. In some respects our planners handle a simpler action representation, but our algorithms are complete, scale to problems with millions of states, exploit the structure of each action's duration distribution, and achieve optimal or close to optimal solutions:

- $\Delta\text{DUR}_{\text{prun}}$ formulates the problem as a concurrent MDP (Mausam & Weld 2004) in a modified interwoven epoch search space, solves the problem with pruned Real Time Dynamic Programming (RTDP) achieving an optimal solution.
- $\Delta\text{DUR}_{\text{samp}}$ substitutes the faster sampled RTDP for pruned RTDP.
- $\Delta\text{DUR}_{\text{hyb}}$ first solves a simpler problem (where actions overlap less tightly) to find a satisficing solution, then uses a process like $\Delta\text{DUR}_{\text{samp}}$ to improve the most important parts of the policy.
- $\Delta\text{DUR}_{\text{arch}}$ approximates a multi-modal duration distribution by considering only one representative length for each mode and re-plans to protect against unexpected events.
- Our fastest planner, $\Delta\text{DUR}_{\text{exp}}$ takes this idea to an extreme by approximating a stochastic duration with its expected length and using a deterministic-duration planner for planning and re-planning.

These planners handle concurrent actions, multi-modal stochastic durations, and probabilistic effects. In addition, this paper makes two additional contributions:

1. We conduct an empirical evaluation that illuminates the planning speed *vs.* solution quality (make-span) trade-off of our planners. $\Delta\text{DUR}_{\text{exp}}$ plans orders of magnitude faster than $\Delta\text{DUR}_{\text{prun}}$, and yet its policies are rarely more than 35% longer.
2. We prove several theorems characterizing the challenges posed by various features of durative planning domain languages. The results provide insights into *deterministic* temporal planners as well as planners handling stochastic durations.

2. Background

Following (Bonet & Geffner 2003), we define a *Markov decision process* as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0 \rangle$ in which $\mathcal{S}$ is a finite set of discrete states, and $\mathcal{A}$ is a finite set of actions. An applicability function, $Ap : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, denotes the set of actions that can be applied in a given state ($\mathcal{P}$ represents the power set). $\mathcal{P}r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function. We write $\mathcal{P}r(s'|s, a)$ to denote the probability of arriving at state s' after executing action a in state s. $\mathcal{C} : \mathcal{S} \times \mathcal{A} \rightarrow \Re^+$ is the cost model, $\mathcal{G} \subseteq \mathcal{S}$ is the set of absorbing goal states, and s_0 is the start state.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

We assume full observability, and we seek to find an optimal, stationary policy — *i.e.*, a function π: $\mathcal{S} \rightarrow \mathcal{A}$ which minimizes the expected cost (over an indefinite horizon) incurred to reach a goal state. A *cost function* J: $\mathcal{S} \rightarrow \Re$, mapping states to the expected cost of reaching a goal state, defines a policy as follows:

$$\pi_J(s) = \underset{a \in Ap(s)}{\operatorname{argmin}} \left\{ \mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) J(s') \right\}$$

The *optimal* policy can be derived from the value function, J^*: $\mathcal{S} \rightarrow \Re$, which satisfies the following pair of *Bellman equations*:

$$J^*(s) = 0, \text{ if } s \in \mathcal{G} \text{ else} \tag{1}$$
$$J^*(s) = \min_{a \in Ap(s)} \left\{ \mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) J^*(s') \right\}$$

Policy Construction Algorithms: *Value Iteration* is a dynamic programming approach in which the optimal value function is calculated as the limit of a series of approximations. If $J_n(s)$ is the value of state s in iteration n, then $J_{n+1}(s)$ is calculated by a *Bellman backup* as: $J_{n+1}(s) = \min_{a \in Ap(s)} \left\{ \mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) J_n(s') \right\}$.

Value iteration and similar algorithms (*e.g.* Policy Iteration) tend to be quite slow since they search the entire state space. *Reachability analysis* is a technique, exploited by RTDP (Barto, Bradtke, & Singh 1995) and other methods, that speeds up this search by restricting it to the part of state space reachable from the initial state s_0.

Conceptually, RTDP is a lazy version of value iteration in which the states are updated in proportion to the frequency with which they are visited by the repeated execution of the greedy policy[1]. An RTDP *trial* is a path starting from s_0, following the greedy policy and updating the values of the states visited using Bellman backups; the trial ends when a goal is reached or the number of updates exceeds a threshold. RTDP repeats these trials until convergence. Note that common states are updated frequently, while RTDP wastes no time on states that are unreachable, given the current policy. RTDP's strength is its ability to quickly produce a relatively good policy. An optimization, called Labeled RTDP (LRTDP), speeds termination and guarantees convergence to the optimal value function (under mild assumptions) (Bonet & Geffner 2003).

Probabilistic Planning with Concurrent Actions: Concurrent MDPs allow concurrent action executions by treating a *set* of actions, called an *action combination* (A), as a single executable abstract action creating an MDP in this augmented action space (Mausam & Weld 2004). The model defines the set of applicable combinations in a state and the probability transition function of executing an action combination.[2] Additionally, the cost model for such a CoMDP returns the cost (typically the time or a weighted sum of time and resource components) of concurrently executing several actions in a state.

Since a CoMDP is an MDP in an abstract action space, MDP algorithms like RTDP can solve a CoMDP. But a CoMDP has a possibly exponential number of joint actions. Pruning and sampling are two techniques that speed up the naive RTDP. Pruned RTDP eliminates some provably suboptimal action combinations from a Bellman backup. Sampled RTDP speeds policy construction substantially — with negligible loss in policy quality — by stochastically backing up a biased subset of possible abstract actions in each Bellman backup.

One may speed planning even further by *hybridizing* two algorithms, one slow but optimal and the other fast but suboptimal; the resulting policy is guaranteed to be within an error-bound (Mausam & Weld 2005). Hybridization runs the anytime optimal algorithm long enough to generate a policy which is good on the common states but stops well before it converges in every state. Then, to ensure that even for the rarely explored states the policy is guaranteed to reach a goal with probability 1, it substitutes the sub-optimal policy on states from thereon, returning this *hybrid* policy. It evaluates this hybrid policy and compares this evaluated cost with the current cost of start state in the optimal algorithm. If the two values are close enough we can prove that the current hybrid solution is close to optimal, and the planning terminates. Otherwise, the optimal algorithm is resumed for some more time, thus repeating the process.

Probabilistic Planning with Deterministic Durations: In the original formulation, a CoMDP does not model explicit action durations; instead it embeds that information in the action cost model. By using an augmented ("*interwoven*") state space, similar to the representation used by several deterministic temporal planners (Bacchus & Ady 2001; Haslum & Geffner 2001; Do & Kambhampati 2001), one can use CoMDPs to model explicit action durations and allow an action to start while other actions are executing (Mausam & Weld 2005). Unfortunately, this work on concurrent durative actions assumes deterministic durations, which is often unrealistic.

3. Challenges for Temporal Planning

This section presents theoretical results, which apply to planning with both deterministic and uncertain discrete durations, regardless of whether effects are deterministic or stochastic. Actions of uncertain duration are modeled by associating a distribution (possibly conditioned on the outcome of stochastic effects) over execution times. We focus on problems whose objective is to achieve a goal state while minimizing total expected time (*make-span*), but our results extend to cost functions that combine make-span and resource usage. This raises the question of *when* a goal counts as achieved. We require that all executing actions terminate before the goal is considered achieved.

[1]A greedy policy is one that chooses the action with the best Q-value defined as $Q_{n+1}(s,a) = \mathcal{C}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) J_n(s')$.

[2]While the original formulation is conservative, banning combinations which violate mutual exclusion rules similar to those in classical planning (Blum & Furst 1995), the model and algorithms are general — working with any semantics that defines these two quantities consistently.

```
:action a
   :duration 4
   :condition (over all P) (at end Q)
   :effect (at end Goal)
:action b
   :duration 2
   :effect (at start Q) (at end (not P))
```

Figure 1: A domain to illustrate that an expressive action model may require arbitrary decision epochs for a solution. In this example, b needs to start at 3 units after a's execution to reach *Goal*.

We start by asking the question "Is there a restricted set of time points such that optimality is preserved even if actions are started only at these points?"

Definition *Any time point when a new action is allowed to start execution is called a* decision epoch. *A time point is a* pivot *if it is either 0 or a time when an executing action might terminate. A* happening *is either 0 or a time when an action actually terminates.*

Intuitively, a happening is a point where an action termination actually "happens." When execution crosses a pivot (a possible happening), information is gained by the executive (*i.e.*, did or didn't the action terminate) which may "change the direction" of future action choices. Clearly, if action durations are deterministic, then the set of pivots is the same as the set of happenings.

Theorem 1 *Restricting decision epochs to pivots causes incompleteness (i.e., a problem may be incorrectly deemed unsolvable).*

Proof: Consider the deterministic temporal planning domain in Figure 1 that uses $\text{PDDL}_{2.1}$ notation (Fox & Long 2003). If the initial state is P=true and Q=false, then the only way to reach *Goal* is to start a at time 0, and b at time 3. Clearly, no action could terminate at 3, still it is a necessary decision epoch. □

Intuitively, two $\text{PDDL}_{2.1}$ actions may require a certain relative alignment within them to achieve the goal. This alignment may force one action to start somewhere (possibly at a non-pivot point) in the midst of the other's execution, thus requiring many decision epochs to be considered.

Temporal planners may be classified as having one of two architectures: constraint-posting approaches in which the times of action execution are gradually constrained during planning (*e.g.*, Zeno and LPG (Penberthy & Weld 1994; Gerevini & Serina 2002)) and extended state-space methods (*e.g.*, TP4 and SAPA (Haslum & Geffner 2001; Do & Kambhampati 2001)). Theorem 1 holds for both architectures but has strong computational implications for state-space planners because limiting attention to a subset of decision epochs can speed these planners. (The theorem also shows that planners like SAPA and Prottle are incomplete.) Fortunately, an assumption restricts the set of decision epochs considerably.

Definition *Actions are* TGP-style *if their preconditions must be true throughout execution, their effects are guaranteed to be true only after termination, and they may not execute concurrently if they clobber each other's preconditions or effects (Smith & Weld 1999).*

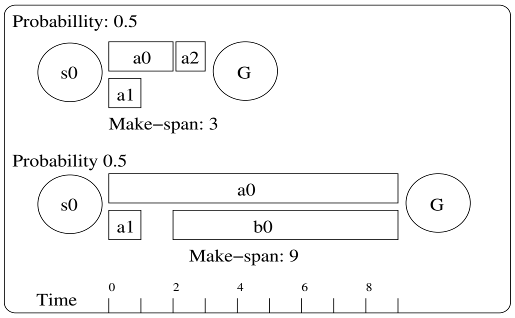


Figure 2: (Adapted from (Mausam & Weld 2006)) Pivot decision epochs are necessary for optimal planning in face of non-monotonic continuation. In this domain, *Goal* can be achieved by $\langle\{a_0, a_1\}; a_2\rangle$ or $\langle b_0\rangle$; a_0 has duration 2 or 7; and b_0 is mutex with a_1. The optimal policy starts a_0 and then, if a_0 does *not* finish at time 2, it starts b_0 (otherwise it starts a_1).

Theorem 2 *If all actions are TGP-style, then the set of decision epochs may be restricted to pivots without sacrificing completeness or optimality.*

Proof Sketch: By contradiction. Suppose that no optimal policy satisfies the theorem; then there must exist a path through the optimal policy in which one must start an action, a, at time t even though there is no action which could have terminated at t. Since the planner hasn't gained any information at t, a case analysis (which requires actions to be TGP-style) shows that one could have started a earlier in the execution path without increasing the make-span. □

When planning with uncertain durations there may be a huge number of times when actions *might* terminate; it is useful to further constrain the range of decision epochs.

Definition *An action has* independent duration *if there is no correlation between its probabilistic effects and its duration. An action has* monotonic continuation *if the expected time until action termination is nonincreasing during execution.*

Actions without probabilistic effects have independent duration. Actions with monotonic continuations are common, *e.g.* those with uniform, exponential, Gaussian, and many other duration distributions. However, actions with bimodal or multi-modal distributions don't have monotonic continuations (see Section 6).

Conjecture 3 *If all actions are TGP-style, have independent duration and monotonic continuation, then the set of decision epochs may be restricted to happenings without sacrificing completeness or optimality.*

To date we have only proven the conjecture for the case in which all durations are uniformly distributed and the maximum concurrency in the domain is 2. If an action's continuation is nonmonotonic then failure to terminate can increase the expected time remaining and cause another sub-plan to be preferred (see Figure 2). Similarly, if an action's duration isn't independent then failure to terminate changes the probability of its eventual effects and this may prompt new actions to be started.

4. Optimal Planning with Uncertain Durations

As explained in the discussion of Theorem 1, temporal planners may be classified as using constraint-posting or extended state-space methods. While the constraint approach is promising, few (if any) *probabilistic* planners have been implemented using this architecture; one exception is BURIDAN (Kushmerick, Hanks, & Weld 1995), which performed poorly. In contrast, the MDP community has proven the state-space approach successful. Since powerful temporal planners also use the state-space approach, we adopt it. We assume TGP-style actions, independent durations, and monotonic continuations, but Section 6 relaxes the latter, extending our algorithms to handle multi-modal duration distributions. Recall that we use a discrete temporal model and aim to minimize the time required to reach a goal.

Since optimal planning decisions may require reasoning about the distribution of an action's remaining execution time, we modify the *interwoven epoch search space* (Mausam & Weld 2005) for our purposes: augmenting the world-state space $\mathcal{S}$ with the set of actions currently executing and the time passed since they were started. Formally, let the interwoven state[3] $s \in \mathcal{S}_{\underline{-}}$ be an ordered pair $\langle X, Y\rangle$ where $X \in \mathcal{S}$ and $Y = \{(a,\delta) | a \in \mathcal{A}, 0 < \delta \leq \Delta_M(a)\}$. Here, X represents the values of the state variables (*i.e.* X is a state in the original state space) and Y denotes the set of ongoing actions "a" and the time passed[4] since their start "δ". $\Delta_M(a)$ denotes the maximum time within which action a will complete. Thus the overall interwoven-epoch search space is $\mathcal{S}_{\underline{-}} = \mathcal{S} \times \bigotimes_{a\in\mathcal{A}} \left(\{a\} \times Z_{\Delta_M(a)}\right)$, where $Z_{\Delta_M(a)}$ represents the set $\{0, 1, \ldots, \Delta_M(a)-1\}$ and $\bigotimes$ denotes the Cartesian product over multiple sets.

To allow the possibility of simply waiting for the next decision-epoch, that is, not executing any action at the current epochs, we augment the set $\mathcal{A}$ with a *no-op* action. We allow no-op to be applicable in all states where some action is executing. The no-op has a variable duration equal to the time until the next decision epoch.

The Transition Function: Uncertain durations require significant changes to the probability transition function ($\mathcal{P}r_{\underline{-}}$) for the interwoven space from the definitions of (Mausam & Weld 2005). Since our assumptions justify Conjecture 3, we need only consider happenings when choosing decision epochs. (Section 6 relaxes this assumption.)

The computation of transition function is described in Algorithm 1. Although the next decision epoch is determined by a happening, we still need to *consider* all pivots for the next state calculations as all these are potential happenings. This makes the algorithm computationally intensive because there may be many pivots and many action combinations could end at each one. In our implementation, we cache the transition functions so that we do not have to recompute the information for any state.

[3]We use the subscript $\underline{-}$ to denote the *interwoven* state space ($\mathcal{S}_{\underline{-}}$), value function ($J_{\underline{-}}$), *etc.*

[4]Our representation is similar to that of (Mausam & Weld 2005), but there is a significant difference: in the previous work, the states contained the times *remaining* for executing actions. But in case of uncertain durations, time remaining is a *distribution*, so we instead represent the time *elapsed* for each action — a strictly more expressive approach.

Algorithm 1 ComputeTransitionFunc(s=$\langle X, Y\rangle$,A)

```
1: $\mathcal{Y} \leftarrow Y \cup \{(a, 0)\}\ \forall a \in A$
2: $mintime \leftarrow \min_{(a,\delta)\in\mathcal{Y}}$ minimum remaining time for $a$
3: $maxtime \leftarrow \min_{(a,\delta)\in\mathcal{Y}}$ maximum remaining time for $a$
4: for all $t \in [mintime, maxtime]$ do
5:     $A_t \leftarrow$ set of actions that could possibly terminate at $t$
6:     for all non- empty subsets $Asub_t \subseteq A_t$ do
7:         $p_c \leftarrow$ prob. that exactly $Asub_t$ terminates at $t$.
8:         $\mathcal{W} \leftarrow \{(X_t, p_w) \mid X_t$ is a world state; $p_w$ is the probability that $Asub_t$ terminates yielding $X_t\}$.
9:         for all $(X_t, p_w) \in \mathcal{W}$ do
10:            $Y_t \leftarrow Y_t \cup \{(a, t + \delta)\}\ \forall(a, \delta) \in \mathcal{Y}, a \notin Asub_t$
11:            insert $(\langle X_t, Y_t\rangle, p_w \times p_c)$ in $output$
12: return $output$
```

Start and Goal States: The start state is $\langle s_0, \emptyset\rangle$ and the new set of goal states is $\mathcal{G}_{\underline{-}} = \{\langle X, \emptyset\rangle | X \in \mathcal{G}\}$.

Thus we have modeled our problem as a CoMDP in the interwoven state space. We have redefined the start and goal states, and the probability transition function. Now we can use the techniques of CoMDPs to solve our problem. In particular, we can use our Bellman equations as below.

Bellman Equations: Define $\delta_{el}(s, s')$ as the time elapsed between two interwoven states s and s'. The set of equations for the solution of our problem can be written as:

$$J^*_{\underline{-}}(s) = 0, \text{ if } s \in \mathcal{G}_{\underline{-}} \text{ else} \tag{2}$$

$$J^*_{\underline{-}}(s) = \min_{A\in Ap_{\underline{-}}(s)} \sum_{s'\in\mathcal{S}_{\underline{-}}} \left\{\delta_{el}(s, s') + \mathcal{P}r_{\underline{-}}(s'|s, A) J^*_{\underline{-}}(s')\right\}$$

The main bottleneck in solving this problem, besides the size of the interwoven state space, is the high branching factor.

Policy Construction: RTDP & Hybridization Since we have modeled our problem as a CoMDP in interwoven space, we may use pruned RTDP (ΔDUR$_{\text{prun}}$) and sampled RTDP (ΔDUR$_{\text{samp}}$) for policy construction. Furthermore, only small adaptations are necessary to incrementally compute the (admissible) *maximum concurrency* (MC) and (more informed, but inadmissible) *average concurrency* (AC) heuristics (Mausam & Weld 2005).

Likewise, we can further speed planning by hybridizing (ΔDUR$_{\text{hyb}}$) an anytime optimal algorithm with a suboptimal (but faster) algorithm to produce a near-optimal policy in significantly less time (Mausam & Weld 2005; McMahan, Likhachev, & Gordon 2005). We use RTDP in the interwoven space as the optimal algorithm. For a fast solution we solve a simpler problem that allows for a new set of actions to be started only when all other executing actions finish (*i.e.*, essentially solving the CoMDP in the original state space). The novel twist stems from the fact that uncertain durations require computing the *cost* of an action combination as the mean of the max of the possible duration outcomes.[5]

5. Expected-Duration Planner

When modeled as a CoMDP in the full-blown interwoven space, stochastic durations cause a cancerous growth in the branching factor. In general, if n actions are started each with m possible durations and each having r probabilistic effects, then there are $(m-1)[(r+1)^n - r^n - 1] + r^n$ potential successors. Thus, the branching factor is multiplicative in the duration uncertainty and exponential in the concurrency.

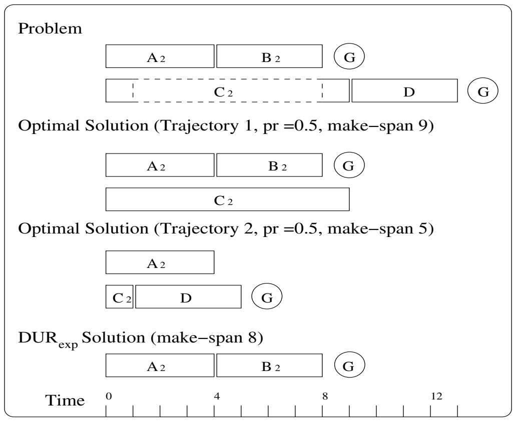
To manage this computational tumor we must curb the branching factor. One method is to ignore duration distributions. We can assign each action a *constant* duration equal to the mean of its distribution, then apply a deterministic-duration planner such as that of (Mausam & Weld 2005). However, when executing the deterministic-duration policy in a setting where durations are actually stochastic, an action will likely terminate at a time *different* than its mean, expected duration. The ΔDUR$_{\text{exp}}$ planner addresses this problem by augmenting the deterministic-duration policy created to account for these unexpected outcomes. The procedure is easiest to understand in its *online* version (Algorithm 2): wait until the unexpected happens, pause execution, and re-plan. If the original estimate of an action's duration is implausible, we compute a revised deterministic estimate in terms of $\mathcal{E}_a(min, max)$ — the expected value of a's duration distribution restricted between times min and max. Recall that Δ_M denotes the max duration of an action. Thus, $\mathcal{E}_a(0, \Delta_M(a))$ will compute the expected duration of a.

Algorithm 2 Online ΔDUR$_{\text{exp}}$

```
1:  build a deterministic-duration policy from the start state $s_0$
2:  **repeat**
3:      execute action combination specified by policy
4:      **wait for interrupt**
5:          **case:** action $a$ terminated as expected {//do nothing}
6:          **case:** action $a$ terminates early
7:              extend policy from current state
8:          **case:** action $a$ didn't terminate as expected
9:              extend policy from current state revising
                $a$'s duration as follows:
10:                 $\delta \leftarrow$ time elapsed since $a$ started executing
11:                 $nextexp \leftarrow \lceil \mathcal{E}_a(0, \Delta_M(a)) \rceil$
12:                 **while** $nextexp < \delta$ **do**
13:                     $nextexp \leftarrow \lceil \mathcal{E}_a(nextexp, \Delta_M(a)) \rceil$
14:                 **endwhile**
15:             $a$'s revised duration $\leftarrow nextexp - \delta$
16:     **endwait**
17: **until** goal is reached
```

This algorithm also has an *offline version* in which re-planning for all contingencies is done ahead of time and we used this version in the experiments for fairness. Although the offline algorithm plans for all possible action durations, it is still much faster than the other algorithms. The reason is that each of the planning problems solved is now significantly smaller (less branching factor, smaller reachable state space), and all the previous computation can be succinctly stored in the form of the $\langle$interwoven state, value$\rangle$ pairs and thus reused.

[5]For example, suppose two actions both with uniform duration distributions between 1 to 3 are started concurrently. The probabilities that both actions will finish by time 1, 2 and 3 are 1/9, 3/9, and 5/9 respectively. Thus the expected duration of completion of the combination is 1×1/9 + 2×3/9 + 3×5/9 = 2.44.



Problem
A2
B2
G
C2
D
G
Optimal Solution (Trajectory 1, pr =0.5, make−span 9)
A2
B2
G
C2
Optimal Solution (Trajectory 2, pr =0.5, make−span 5)
A2
C2
D
G
DURexp Solution (make−span 8)
A2
B2
G
Time
0
4
8
12

Figure 3: An example of a domain where the ΔDUR$_{\text{exp}}$ algorithm does not compute an optimal solution.

Unfortunately, our ΔDUR$_{\text{exp}}$ algorithm is not guaranteed to produce an optimal policy. How bad are the policies generated by the expected-duration planner? The experiments show that ΔDUR$_{\text{exp}}$ typically generates policies which are extremely close to optimal. Even the worst-case pathological domain we are able to construct leads to an expected make-span which is 50% longer than optimal (in the limit). The example is illustrated below.

Example: We consider a domain which has actions $A_{2:n}, B_{2:n}, C_{2:n}$ and D. Each A_i and B_i takes time 2^i. Each C_i has a probabilistic duration: with probability 0.5, C_i takes 1 unit of time, and with the remaining probability, it takes $2^{i+1}+1$ time. Thus, the expected duration of C_i is $2^i + 1$. D takes 4 units. In sub-problem SP_i, the goal may be reached by executing A_i followed by B_i. Alternatively, the goal may be reached by first executing C_i and then recursively solving the sub-problem SP_{i-1}. In this domain, the ΔDUR$_{\text{exp}}$ algorithm will always compute $\langle A_i; B_i\rangle$ as the best solution. However, the optimal policy starts both $\{A_i, C_i\}$. If C_i terminates at 1, the policy executes the solution for SP_{i-1}; otherwise, it waits until A_i terminates and then executes B_i. Figure 3 illustrates the sub-problem SP_2 in which the optimal policy has an expected make-span of 7 (*vs.* ΔDUR$_{\text{exp}}$'s make-span of 8). In general, the expected make-span of the optimal policy on SP_n is $\frac{1}{3}[2^{n+2} + 2^{4-n}] + 2^{2-n} + 2$. Thus, $\lim_{n\to\infty} \frac{exp}{opt} = \frac{3}{2}$.$\square$

6. Multi-modal Duration Distributions

The planners of the previous two sections benefited by considering the small set of happenings instead of pivots, an approach licensed by Conjecture 3. Unfortunately, this simplification is not warranted in the case of actions with multi-modal duration distributions, which can be common in complex domains where all factors can't be modeled explicitly. For example, the amount of time for a Mars rover to transmit data might have a bimodal distribution — normally it would take little time, but if a dust storm were in progress (unmodeled) it could take much longer. To handle these cases we model durations to be specified with a mixture of Gaussians parameterized with a set of triples: $\langle amplitude, mean, variance \rangle$.

CoMDP Formulation: Although we cannot restrict decision epochs to happenings, we need not consider *all* pivots; they are required only for actions with multi-modal distributions. In fact, it suffices to consider pivots in *regions* of the distribution where the expected-time-to-completion increases. We need consider only happenings in other cases.

Two changes are required to the transition function of Algorithm 1. In line 3, the *maxnext* computation now involves time until the next pivot in the increasing remaining time region for all actions with multi-modal distributions (thus forcing us to take a decision at those points, even when no action terminates). Another change (in line 6) allows a *non-empty* subset $Asub_t$ for $t = maxnext$. That is, next state is computed even without any action termination. By making these changes in the transition function we reformulate our problem as a CoMDP in the interwoven space and thus solve, using our previous methods of pruned/sampled RTDP, hybrid algorithm or expected-duration algorithm.

Archetypal-Duration Planner: We also develop a multi-modal variation of the expected-duration planner, called $\Delta\text{DUR}_{\text{arch}}$. Instead of assigning an action a deterministic duration equal to the expected value, this planner assigns it a probabilistic duration with various outcomes being the means of the different modes in the distribution and the probabilities being the probability mass in each mode. This enhancement reflects our intuitive understanding for multi-modal distributions and the experiments confirm that $\Delta\text{DUR}_{\text{arch}}$ produces solutions having shorter make-spans than those of $\Delta\text{DUR}_{\text{exp}}$.

7. Experiments

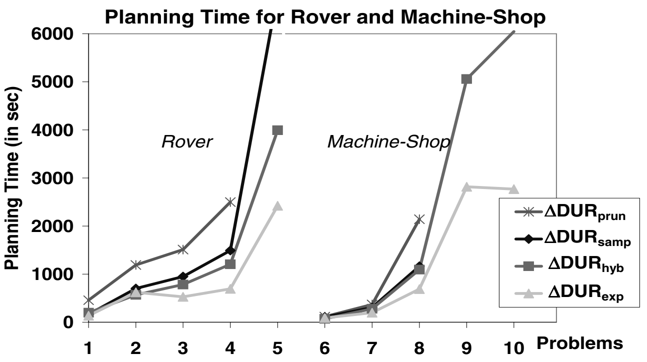


Figure 4: Planning time comparisons: Variation along algorithms when all initialized by the average concurrency (AC) heuristic; $\Delta\text{DUR}_{\text{exp}}$ performs the best.

In this section we compare the computation time and solution quality (make-span) of our five planners for domains with and without multi-modal duration distributions. We also tested on the effectiveness of the maximum- (*MC*) and average-concurrency (*AC*) heuristics.

Experimental Setup: We test our algorithms on problems in three domains, in each case generating problem instances by varying the number of objects, amount of parallelism, action durations, and distances to the goal. The first domain is a probabilistic, durative variant of NASA Rover domain from the 2002 AIPS Planning Competition, the second is a probabilistic temporal version of the Machine-Shop domain, and the third is an artificial domain, which lets us independently control the domain size and degree of parallelism. Our largest problem had 4 million world states of which 65536 were reachable. Our algorithms explored up to 1,000,000 distinct states in the interwoven state space during planning. The domains contained as many as 18 actions, and some actions had as many as 13 possible durations.

Our implementation is in C++ and built on top of GPT's Labeled RTDP (Bonet & Geffner 2003). The heuristics (maximum- and average-concurrency) are calculated on demand as states are visited. We ran experiments on a 2.8 GHz Pentium 4 with 2 GB memory running Linux.

Comparing Running Times: We compare all algorithms with and without heuristics and find that the heuristics significantly speed up the computation on all problems; indeed, some problems are too large to be solved without heuristics. Comparing them amongst themselves we find that AC beats MC — regardless of the planning algorithm; this isn't surprising since AC sacrifices admissibility.

In Figure 4 we report the running times of various algorithms on the Rover and Machine-Shop domains when all durations are unimodal. We report the results for algorithms initialized with the AC heuristic. $\Delta\text{DUR}_{\text{exp}}$ out-performs the other planners by substantial margins. As this algorithm is solving a comparatively simpler problem, fewer states are expanded and thus the approximation scales better than others — solving, for example, two Machine-Shop problems, which were too large for most other planners. In most cases hybridization speeds planning by significant amounts. It performs better than $\Delta\text{DUR}_{\text{exp}}$ only for the artificial domain.

Comparing Solution Quality: We measure quality by simulating the generated policy across multiple trials. We report the ratio of average expected make-span and the optimal expected make-span for domains with all unimodal distributions in Figure 5. We find that the make-spans of the inadmissible heuristic AC are at par with those of the admissible heuristic MC. The hybrid algorithm is approximate with a user-defined bound. In our experiments, we set the bound to 5% and find that the make-spans returned by the algorithm are quite close to the optimal and do not always differ by 5%. $\Delta\text{DUR}_{\text{exp}}$ has no quality guarantees, still the solutions returned on the problems we tested upon are not much worse than other algorithms. Thus, we believe that this approximation will be quite useful in scaling to larger problems without losing solution quality.

Multi-modal Domains: We develop multi-modal variants of our domains; *e.g.*, in the Machine-Shop domain, time for

Algos	Average Quality of Make-Span		
	Rover	Machine-Shop	Artificial
$\Delta\mathrm{DUR}_{\mathrm{samp}}$	1.001	1.000	1.001
$\Delta\mathrm{DUR}_{\mathrm{hyb}}$	1.022	1.011	1.019
$\Delta\mathrm{DUR}_{\mathrm{exp}}$	1.008	1.015	1.046

Figure 5: All three planners produce near-optimal policies as shown by this table of ratios to the optimal make-span.

fetching paint was bimodal (if in stock, paint can be fetched fast, else it needs to be ordered). There was an alternative but costly paint action that doesn't require fetching of paint. Solutions produced by $\Delta\mathrm{DUR}_{\mathrm{samp}}$ made use of pivots as decision epochs by starting the costly paint action in case the fetch action didn't terminate within the first mode of the bimodal distribution (*i.e.* paint was out of stock).

The running time comparisons are shown in Figure 6(a) on a log-scale. We find that $\Delta\mathrm{DUR}_{\mathrm{exp}}$ performs super-fast and $\Delta\mathrm{DUR}_{\mathrm{arch}}$ is not far behind. However, the make-span comparisons in Figure 6(b) clearly illustrate the approximations made by these methods in order to achieve planning time. $\Delta\mathrm{DUR}_{\mathrm{arch}}$ seems to have a good balance of planning time and solution quality.

8. Related Work

Tempastic (Younes & Simmons 2004a) uses a rich formalism (*e.g.* continuous time, exogenous events, and expressive goal language) to generate concurrent plans with stochastic durative actions. Tempastic uses a completely non-probabilistic planner to generate a plan which is treated as a candidate policy and repaired as failure points are identified. This method does not guarantee completeness or proximity to the optimal. Moreover, no attention was paid towards heuristics or search control making the implementation impractical.

GSMDPs (Younes & Simmons 2004b) extend continuous-time MDPs and semi-Markov MDPs, modeling asynchronous events and processes. Younes and Simmons solve GSMDPs by approximation with a standard MDP using phase-type distributions. The approach is elegant, but its scalability to realistic problems is yet to be demonstrated. In particular, the approximate, discrete MDP model can require many states yet still behave very differently than the continuous original.

Prottle (Little, Aberdeen, & Thiebaux 2005) also solves problems with an expressive action language: effects can occur in the middle of action execution and dependent durations are supported. Prottle uses an RTDP-type search guided by heuristics computed from a probabilistic planning graph; however, it plans for a finite horizon — and thus for an acyclic state space. It is difficult to compare Prottle with our approach because Prottle optimizes a different objective function (probability of reaching a goal), outputs a finite-length conditional plan as opposed to a cyclic plan or policy, and is not guaranteed to reach the goal.

We know of no other planners that address both concurrency and uncertain durations in a probabilistic context but there has been past research in planning with uncertain durations where each action is associated with an unweighted set of durations, *e.g.*, CIRCA (Musliner, Murphy, & Shin 1991).

Aberdeen *et al.* (2004) use domain-specific heuristics to plan with concurrent but non-durative actions. Little *et al.* (2006) extend the classical Graphplan to propose an alternative solution to a concurrent MDP. Many researchers have studied planning with stochastic, durative actions in *absence* of concurrency. For example, Foss and Onder (2005) use *simple temporal networks* to generate plans in which the objective function has no time component. Boyan and Littman (2000) propose *Time-dependent MDPs* to model problems with (non-concurrent) actions having time-dependent, stochastic durations; their solution generates piece-wise linear value functions. NASA researchers have developed techniques for generating non-concurrent plans with uncertain continuous durations using a greedy algorithm which incrementally adds branches to a straight-line plan (Bresina *et al.* 2002; Dearden *et al.* 2003). IxTeT is a temporal planner that uses constraint based reasoning within partial order planning (Laborie & Ghallab 1995). It embeds temporal properties of actions as constraints and does not optimize make-span.

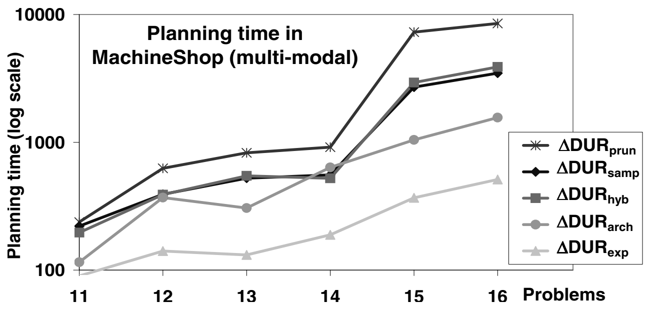
9. Conclusions

Although concurrent actions and uncertain durations characterize many real-world domains, few planners can handle both challenges in concert. This paper considers state-space based solutions which are popular both in deterministic temporal planning and in probabilistic planning. We find that this architecture must cope with 1) large state spaces, 2) high branching factors, and 3) an explosion in number of decision epochs. We bound the space of decision epochs in terms of *pivots* (times when actions may potentially terminate) and conjecture further restrictions, thus making the problem tractable. We model our problem as a concurrent MDP in an augmented space and are able to solve it optimally using our $\Delta\mathrm{DUR}_{\mathrm{prun}}$ algorithm; $\Delta\mathrm{DUR}_{\mathrm{samp}}$ and $\Delta\mathrm{DUR}_{\mathrm{hyb}}$ are much faster and also produce close-to-optimal solutions.

Of our five algorithms $\Delta\mathrm{DUR}_{\mathrm{exp}}$ is the fastest — it solves multiple planning problems, each with low branching factor and smaller reachable state space. These small problems approximate an action's stochastic duration with its expected value. Although the offline $\Delta\mathrm{DUR}_{\mathrm{exp}}$ must consider action terminations at each of its pivots (*i.e.*, when the expectation is violated), for each planning subproblem all durations are treated deterministic; this shrinks the problem size enormously, quickly producing near-optimal solutions for many problems.

We also show how to extend each planner to domains in which actions have multi-modal duration distributions. We create one planner, $\Delta\mathrm{DUR}_{\mathrm{arch}}$, specifically for these problems. $\Delta\mathrm{DUR}_{\mathrm{arch}}$ creates policies whose make-span is better than those of $\Delta\mathrm{DUR}_{\mathrm{exp}}$, yet runs faster than $\Delta\mathrm{DUR}_{\mathrm{prun}}$, $\Delta\mathrm{DUR}_{\mathrm{samp}}$ or $\Delta\mathrm{DUR}_{\mathrm{hyb}}$ — an interesting point on the planning time *vs.* solution-quality trade-off.

Overall, our paper takes an important step in exposing the issues related to concurrent actions with duration uncertainty. Indeed, our theorems provide insights into deterministic temporal planners as well as those handling stochastic durations.



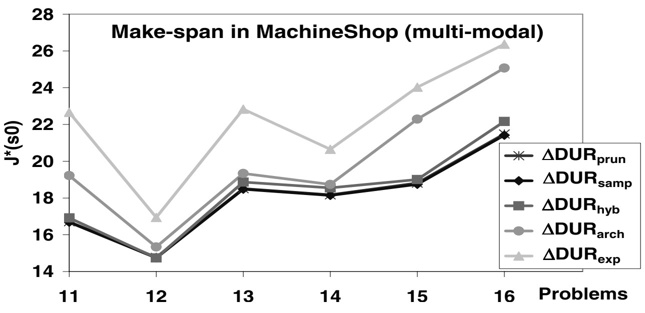


Figure 6: Comparisons in the Machine-Shop domain with multi-modal distributions. (a) Computation Time comparisons: $\Delta\mathrm{DUR}_{\mathrm{exp}}$ and $\Delta\mathrm{DUR}_{\mathrm{arch}}$ perform much better than other algos. (b) Make-spans returned by different algos: Solutions returned by $\Delta\mathrm{DUR}_{\mathrm{samp}}$ are almost optimal. Overall $\Delta\mathrm{DUR}_{\mathrm{arch}}$ finds a good balance between running time and solution quality.

In the future we hope to develop algorithms to handle more expressive action models similar to $\mathrm{PDDL}_{2.1}$ and Prottle's language. Combining constraint-posting methods with a probabilistic context may be the key to solve such problems. We also wish to extend our algorithms to handle continuous duration distributions.

Acknowledgments

We thank Blai Bonet for providing the source code of GPT. We thank Sumit Sanghai for theorem proving skills and advice. We also thank Jiun-Hung Chen, Nilesh Dalvi, Maria Fox, Jeremy Frank, Subbarao Kambhampati, and Håkan Younes for helpful suggestions. Raphael Hoffman, Daniel Lowd, Tina Loucks, Alicen Smith, Deepak Verma and the anonymous reviewers gave useful comments on prior drafts. This work was supported by NSF grant IIS-0307906, ONR grants N00014-02-1-0932, N00014-06-1-0147 and the WRF / TJ Cable Professorship.

References

Aberdeen, D.; Thiebaux, S.; and Zhang, L. 2004. Decision-theoretic military operations planning. In *ICAPS'04*.

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. In *IJCAI'01*, 417–424.

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *IJCAI'95*, 1636–1642. Morgan Kaufmann.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS'03*, 12–21.

Boyan, J. A., and Littman, M. L. 2000. Exact solutions to time-dependent MDPs. In *NIPS'00*, 1026.

Bresina, J.; Dearden, R.; Meuleau, N.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty : A challenge for AI. In *UAI'02*.

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D. E.; and Washington, R. 2003. Incremental Contingency Planning. In *ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In *ECP'01*.

Foss, J., and Onder, N. 2005. Generating temporally contingent plans. In *IJCAI'05 Workshop on Planning and Learning in Apriori Unknown or Dynamic Domains*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR Special Issue on 3rd International Planning Competition* 20:61–124.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs with action graphs. In *AIPS'02*, 281.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *ECP'01*.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1-2):239–286.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *IJCAI'95*, 1643.

Little, I., and Thiebaux, S. 2006. Concurrent probabilistic planning in the graphplan framework. In *ICAPS'06*.

Little, I.; Aberdeen, D.; and Thiebaux, S. 2005. Prottle: A probabilistic temporal planner. In *AAAI'05*.

Mausam, and Weld, D. 2004. Solving concurrent Markov decision processes. In *AAAI'04*.

Mausam, and Weld, D. 2005. Concurrent probabilistic temporal planning. In *ICAPS'05*, 120–129.

Mausam, and Weld, D. 2006. Challenges for temporal planning with uncertain durations. In *ICAPS'06*.

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML'05*.

Musliner, D.; Murphy, D.; and Shin, K. 1991. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74:83–127.

Penberthy, J., and Weld, D. 1994. Temporal planning with continuous change. In *AAAI'94*, 1010.

Smith, D., and Weld, D. 1999. Temporal graphplan with mutual exclusion reasoning. In *IJCAI'99*, 326–333. Stockholm, Sweden: San Francisco, CA: Morgan Kaufmann.

Younes, H. L. S., and Simmons, R. G. 2004a. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS'04*, 325.

Younes, H. L. S., and Simmons, R. G. 2004b. Solving generalized semi-markov decision processes using continuous phase-type distributions. In *AAAI'04*, 742.