Concurrent Probabilistic Temporal Planning : Initial Results

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

Content areas: Markov Decision Processes, Planning

Abstract

Probabilistic planning problems are often modeled as Markov decision problems (MDPs), which assume that a single action is executed per decision epoch and that actions take unit time. However, in the real world it is common to execute several actions in parallel, and the durations of these actions may differ. This paper presents our ongoing work on incorporating concurrent, durative actions in probabilistic planning. In particular we describe *Concurrent MDPs*, MDPs which allow multiple instantaneous actions to be executed simultaneously, and present two algorithms which perform orders of magnitude faster than naive approaches. We then add explicit action durations into our model and encode them as concurrent MDPs in an augmented state space. We present a novel heuristic and prove it admissible. Initial experimental results demonstrate the promise of our approach, showing speedup due to both the heuristic and our sampled RTDP algorithm.

1. Introduction

Recent progress achieved by planning researchers has yielded new algorithms which relax, individually, many of the classical assumptions. However, in order to apply automated planning to many real-world domains we must eliminate larger groups of the assumptions in concert. For example, (Bresina *et al.* 2002) notes that optimal control for a NASA Mars rover requires reasoning about uncertain, concurrent, durative actions and a mixture of discrete and metric fluents. While today's planners can handle large problems with *deterministic* concurrent durative actions, and semi-MDPs provide a clear framework for durative actions in the face of uncertainty, few researchers have considered concurrent, uncertain actions — the focus of this paper.

For example, a Mars rover has the goal of gathering data from different locations with various instruments (color and infrared cameras, microscopic imager, Mossbauer spectrometers *etc.*) and transmitting this data back to Earth. Concurrent actions are essential since instruments can be turned on, warmed up and calibrated while the rover is moving, using other instruments or transmitting data. Similarly, uncertainty must be explicitly confronted as the rover's movement, arm control and other actions cannot be accurately predicted.

We adopt the framework of *Markov decision processes* (MDPs) and extend it to allow multiple actions per decision epoch. In the traditional case of a single action per decision epoch, state-space heuristic search and dynamic programming have proven quite effective . However, allowing multiple concurrent actions at a time point will inflict an exponential blowup on all of these techniques.

In this paper we summarise our methods (published in (Mausam & Weld 2004)) to efficiently solve concurrent MDPs with actions of unit length. The focus of this paper is our initial results in extending these methods to problems with durative actions, *i.e.* concurrent probabilistic temporal planning - in short, *CPTP*. Specifically, we extend the technique of *real-time dynamic programming* (RTDP) (Barto, Bradtke, & Singh 1995; Bonet & Geffner 2003) to solve our problems. We model a CPTP problem as a concurrent MDP in an augmented state space and propose lower bounds on value functions which can be used as admissible heuristics in the heuristic search using RTDP.

2. Background

Following (Bonet & Geffner 2003), we define a *Markov decision process* as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0, \gamma \rangle$[1] in which

- $\mathcal{S}$ is a finite set of discrete states.
- $\mathcal{A}$ 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.
- $\mathcal{C} : \mathcal{A} \rightarrow \Re^+$ is the cost model[2].
- $\mathcal{G} \subseteq \mathcal{S}$ is a set of absorbing goal states.
- s_0 is a start state.
- $\gamma \in [0, 1]$ is the discount factor. If $\gamma = 1$ our problem is known as the *stochastic shortest path problem*.

[1] We believe our elimination and sampling techniques can be easily extended to handle MDPs with rewards or non-absorbing goal states, and we plan to test this empirically.

[2] Indeed, most of our techniques allow costs to be conditioned on states as well as actions.

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

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

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

$$J^*(s) = 0, \text{ if } s \in \mathcal{G} \text{ else}$$

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

Various algorithms have been developed to solve MDPs. Please refer to (Bertsekas 1995) for a discussion on Value iteration and Policy iteration. These tend to be quite slow due to complete state space search. *Reachability Analysis* is a technique employed to speed up this search. In this, the search is restricted to the part of state space reachable from the initial state s_0. Two algorithms exploiting this are LAO* (Hansen & Zilberstein 2001) and our focus: RTDP (Barto, Bradtke, & Singh 1995).

RTDP, conceptually, is a lazy version of value iteration in which the states get updated in proportion to the frequency with which they are visited by the repeated executions of the greedy policy. Specifically, RTDP is an anytime algorithm that simulates the greedy policy along a single trace execution, and updates the values of the states it visits using Bellman backups. An RTDP *trial* is a path starting from s_0 and ending 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; however, complete convergence (at every state) is slow because less likely (but potentially important) states get updated infrequently. Furthermore, RTDP is not guaranteed to terminate. *Labeled RTDP* fixes these problems with a clever labeling scheme that focusses attention on states where the value function has not yet converged (Bonet & Geffner 2003). Labeled RTDP is guaranteed to terminate, and is guaranteed to converge to the optimal value function (for states reachable using the optimal policy) if the initial value function is admissible.

3. Concurrent Markov Decision Processes

Extending traditional MDPs to *concurrent MDPs*, *i.e.* allowing multiple parallel actions, each of unit duration, requires several changes. Clearly, certain actions can't be executed in parallel; so we adopt the classical planning notion of mutual exclusion (Blum & Furst 1995) and apply it to a *factored* action representation: *probabilistic STRIPS* (Boutilier, Dean, & Hanks 1999). Two actions are *mutex* (may not be executed concurrently) if in any state 1) they have inconsistent preconditions[3], 2) they have conflicting effects, or 3) the precondition of one conflicts with the (possibly probabilistic) effect of the other. Thus, non-mutex actions don't interact — the effects of executing the sequence $a_1; a_2$ equals those for $a_2; a_1$.

An *action combination*, A, is a set of one or more actions to be executed in parallel. The cost model $\mathcal{C}$ is now a function, $\mathcal{C} : \mathcal{P}(\mathcal{A}) \rightarrow \Re^+$, *i.e.* the domain is the *power-set* of actions. Note that unless there exists a combination A, such that $\mathcal{C}(A) < \sum_{a \in A} \mathcal{C}(\{a\})$, the optimal policy from the single-action MDP would be optimal for the concurrent case as well. However, we believe that in many domains most combinations will obey the inequality. Indeed, the inequality holds when the cost of a combination includes both *resource* and *time* components. Here, one can define the cost model to be comprised of two parts:

- t : Time taken to complete the action.
- r : Amount of resources used for the action.

Assuming additivity, we can think of cost of an action $\mathcal{C}(a) = t(a) + r(a)$, to be sum of its time and resource usage. Hence, the cost model for a combination of actions in terms of these components would be defined as:

$$\mathcal{C}(\{a_1, a_2, ..., a_k\}) = \sum_{i=1}^{k} r(a_i) + \max_{i=1..k}\{t(a_i)\}$$

The applicability function, $Ap(s)$, for concurrent MDPs now has range $\mathcal{P}(\mathcal{P}(\mathcal{A}))$; it is redefined in terms of our original definition, now denoted Ap_1. $Ap(s) = \{A \subseteq \mathcal{A} | \forall a, a' \in A,\ a \in Ap_1(s) \wedge\ \neg \text{mutex(a, a')}\}$

Let $A = \{a_1, a_2, \ldots, a_k\}$ be an action combination applicable in s. Since we only allow concurrent execution of non-interacting actions, the transition function may be calculated as follows:

$$\mathcal{P}r(s'|s,A) = \sum_{s_1, s_2, \ldots s_k \in \mathcal{S}} \cdots \sum \mathcal{P}r(s_1|s,a_1)\mathcal{P}r(s_2|s_1,a_2) \ldots \mathcal{P}r(s'|s_k,a_k)$$

Finally, instead of equations (1), the following set of equations represents the solution to a concurrent MDP:

$$J^*(s) = 0, \text{ if } s \in \mathcal{G} \text{ else}$$

$$J^*(s) = \min_{A \in Ap(s)} \left\{ \mathcal{C}(A) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,A) J^*(s') \right\} (2)$$

These equations are the same as in a traditional MDP, except that instead of considering single actions for backup in a state, we need to consider all applicable action combinations. Thus, only this small change must be made to traditional algorithms (*e.g.*, value iteration, LAO*, Labeled RTDP). However since the number of action combinations is exponential in $|\mathcal{A}|$, efficiently solving a concurrent MDP requires new techniques.

[3]Note that an action's transition function is typically conditioned on various features of the state (*conditional effects*). We consider these features to be part of preconditions for the purpose of mutex calculation.

Pruned RTDP In our AAAI paper, we present two rules for pruning some action combinations from Bellman backups. The first rule is called *Combo-skipping* in which we can bound $Q(s, A)$ values using $Q(s, a)$ values for $a \in A$. Thus, we can skip some combinations that cannot be optimal in the current backup. Unfortunately, combo-skipping has a weakness — it prunes a combination for only a *single iteration*. In contrast, our second rule, *combo-elimination*, prunes irrelevant combinations altogether. We adapt the action elimination theorem from traditional MDPs (Bertsekas 1995) to prove a similar theorem for concurrent MDPs. We use it to eliminate the provably suboptimal combinations for each state for all subsequent iterations. Since combo-skipping does not change any step of labeled RTDP and combo-elimination removes provably sub-optimal combinations, *pruned* labeled RTDP maintains convergence, termination, optimality and efficiency, when used with an admissible heuristic.

Sampled RTDP We now describe our second algorithm called *sampled RTDP*, which performs backups on a random set of action combinations, choosing from a distribution which favors "likely combinations." We generate our distribution by: 1) using combinations which were previously discovered to have low Q-values (recorded by *memoizing* the best combinations per state, after each iteration); 2) calculating the Q-values of all applicable single actions (using current value function) and then biasing the sampling of combinations to choose the ones which contain actions with low Q-values.

Since the system doesn't consider every possible action combination, sampled RTDP is not guaranteed to choose the best combination to execute at each state. As a result, even when started with an admissible heuristic, the $J_n(s)$ values are no longer admissible or monotonic. This is unfortunate, since admissibility and monotonicity are important properties required for termination and optimality in labeled RTDP; indeed, sampled RTDP loses these important theoretical properties. The good news is that it is extremely useful in practice. In our experiments, sampled RTDP usually terminates quickly, and returns values which are extremely close to the optimal. We also investigated several heuristics to improve the quality of solutions and are discussed in detail in our AAAI paper.

Experiments We tested our algorithms in various domains like a probabilistic variant of the NASA Rover domain from the 2002 AIPS Planning Competition, the traditional MachineShop domain etc. Our problems typically had 20-30 state variables and 15-20 actions with varying degrees of parallelism.

We used Labeled RTDP, as implemented in GPT, as the base MDP solver and implemented our pruned and sampled versions over it. We observed that pruning significantly speeds the algorithm, but sampling yields almost two orders of magnitude speedups with respect to the pruned versions. We also compared the qualities of solutions produced by Sampled-RTDP *w.r.t.* optimal. We found that solutions produced by Sampled-RTDP are always nearly optimal (the maximum error obtained in our runs was less than 1%). We

State variables : $x_1, x_2, x_3, x_4, p_{12}$

Action	Duration	Precond.	Effect	Prob.
toggle-x_1	3	$\neg p_{12}$	$x_1 \leftarrow \neg x_1$	1
toggle-x_2	3	p_{12}	$x_2 \leftarrow \neg x_2$	1
toggle-x_3	5	true	$x_3 \leftarrow \neg x_3$	0.9
			no change	0.1
toggle-x_4	1	true	$x_4 \leftarrow \neg x_4$	0.9
			no change	0.1
toggle-p_{12}	4	true	$p_{12} \leftarrow \neg p_{12}$	1

Goal : $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$

Figure 1: Probabilistic STRIPS definition of a simple CPTP problem

noticed that the speedups obtained increase as concurrency increases. This is a very encouraging result, and we can expect Sampled-RTDP to perform well on large problems involving high concurrency, even if the other approaches fail. Our AAAI paper discusses the experimental methodologies and results in detail.

4. Extending to Durative Actions

We now incorporate action durations in concurrent probabilistic planning problems (CPTP). As a start we consider a model similar to the model of Section 3, except that action costs ($\mathcal{C}(a)$) are replaced by their durations ($\Delta(a)$). Thus minimising expected cost to reach a goal translates to minimising the expected make-span to reach a goal[4]. We assume that actions have fixed durations; relaxing this assumption is an important extension for the future.

CPTP is distinct from a concurrent MDP in several ways. In a concurrent MDP the actions are assumed to be instantaneous; hence at each decision epoch, the stochastic effects of the previous actions can be observed. In CPTP, time is represented as a continuous variable. Hence firstly, the decision epochs can be any point in the continuous time-line. Secondly, various actions may be in the process of executing at a decision epoch. We have to explicitly take into account these actions and their finishing times when making a subsequent decision. Finally, a semi-MDP allows for a distribution over transition times while CPTP assumes fixed durations. Thus, the state space of CPTP is substantially different from that of a concurrent MDP. Without loss of generality (given fixed durations known in advance), we assume that the decision epochs are discrete i.e. all action durations are integer-valued, and the agent can only start an action at these integer time-points.

For simplicity, we adopt the temporal action model of (Smith & Weld 1999), rather than the more complex PDDL2.1 (Fox & Long 2003). Specifically, we assume :

- The effects of an action take place at some point in the interior of the action execution but can be known to hold (and hence used) only once the action has completed.
- The preconditions (and features on which the transition function of the action is conditioned) should remain true (unchanged) at the beginning of an action and while the action is being executed, unless the action itself is modifying them.

[4]We will consider CPTP problems with mixed costs, and non absorbing goals in the future.

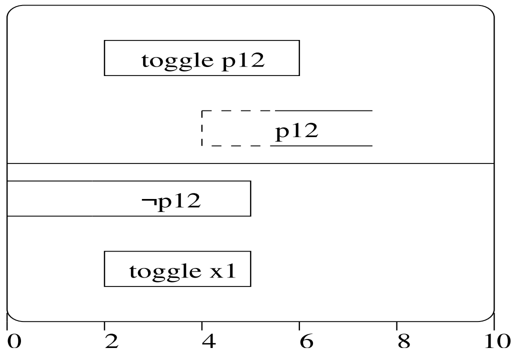

Figure 2: A sample execution demonstrating that two actions with interfering preconditions and effects cannot be executed concurrently.

These definitions are consistent with our previous definition of concurrency. Specifically, it is easy to see that our mutex definitions (from Section 3) hold and are required under these assumptions. As an illustration, consider figure 2. The figure describes a situation in which two actions with interfering preconditions and effects can not be executed concurrently. In our figure, action toggle-p_{12} and toggle-x_1 were started at time 2 units. Lets assume that initially p_{12} was false. As it is a precondition of toggle-x_1, p_{12} needs to remain false until time 5 units (as toggle-x_1 has duration 3). However toggle-p_{12} may produce its effects anytime before 6 units, which may conflict with the preconditions of the other executing action. Therefore, toggle-p_{12} and toggle-x_1 cannot be executed concurrently.

In our model, considering only the time-points when one or more actions complete, as decision epochs, is sufficient for optimal policy construction. Of course not all optimal policies will have this property. But it is easy to see that there exists an optimal policy in which each action begins at one such time-point. Hence this assumption reduces our search space.

Search space We adapt the search space representation of (Haslum & Geffner 2001), similar to (Bacchus & Ady 2001; Do & Kambhampati 2001). Our original state space $\mathcal{S}$ in section 2 is augmented by including the set of actions currently executing and the times remaining for each. Formally, let the new state s be an ordered pair (X, Y) where $X \in \mathcal{S}$ and $Y = \{(a, \delta) | a \in \mathcal{A}, 0 < \delta \leq \Delta(a)\}$. Here X represents the values of the state variables and Y represents the set of actions whose execution hasn't completed and the δ associated with each action represents the amount of time after which the action will complete. Define A_s to be the set of actions already in execution, i.e. A_s is a projection of Y :

$$A_s = \{a | (a, \delta) \in Y\}$$

Example: In our sample domain in figure 1, a possible state (say s_1) is when all state variables are false, and toggle-x_1 was started 1 unit ago. Such a state would be represented as (X_1, Y_1) with X_1=(F, F, F, F, F) and Y_1=$\{$(toggle-$x_1, 2)\}$. The set A_{s_1} would be {toggle-x_1}.

To allow the possibility of simply waiting for some action to complete execution *i.e.* not executing any action at some decision epochs, we increment the set $\mathcal{A}$ with a *no-op* action. We allow no-op to be applicable in all states s = (X, Y) where $Y \neq \emptyset$ (*i.e.* states in which some action is still being executed)[5]. The no-op will have a variable duration equal to the time after which another already executing action completes ($\delta_{next}(s, A)$ as defined below).

The new applicability set can be defined as:

$$Ap(s) = \begin{cases} Ap(X) \text{ if } Y = \emptyset \text{ else} \\ \{noop\} \cup \{A | A \cup A_s \in Ap(X) \text{ and } A \cap A_s = \emptyset\} \end{cases}$$

Transition Function We also need to define the probability transition function $\mathcal{P}r$ for the new state space. At some decision epoch let the agent be in state s = (X, Y), with A_s as the set of actions currently executing. Suppose that the agent decides to execute an action combination A. Define Y_{new} as the set equivalent to Y for the actions just started, *i.e.* $Y_{new} = \{(a, \Delta(a)) | a \in A\}$. In this system, our next decision epoch will be the smallest time after which the any executing action completes. Let us call this time $\delta_{next}(s, A)$. Notice that $\delta_{next}(s, A)$ could depend on actions in both A_s and A. Formally,

$$\delta_{next}(s, A) = \min_{(a,\delta) \in Y \cup Y_{new}} \{\delta\}$$

Moreover, multiple actions may complete simultaneously. Thus, define $A_{next}(s, A) \subseteq A \cup A_s$ to be the set of actions that will complete after $\delta_{next}(s, A)$ time. The Y-component of the state at the decision epoch after $\delta_{next}(s, A)$ time will be

$$Y_{next}(s, A) = \{(a, \delta - \delta_{next}(s, A)) | (a, \delta) \in Y \cup Y_{new}, \\ \delta > \delta_{next}(s, A)\}$$

Let s=(X, Y) and let s'=(X', Y'). The transition function (in terms of our original transition function of concurrent MDPs) can be defined as:

$$\mathcal{P}r(s'|s, A) = \begin{cases} \mathcal{P}r(X'|X, A_{next}(s, A)) \text{ if } Y' = Y_{next}(s, A) \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$

That is, executing an action combination A in state s = (X, Y) takes the agent to a decision epoch $\delta_{next}(s, A)$ ahead in time, *i.e.* the first time when some combination $A_{next}(s, A)$ completes. This lets us calculate the new set of actions still executing and their remaining times ($Y_{next}(s, A)$). And the original probability transition function can be used to decide the new distribution of state variables, as if the combination $A_{next}(s, A)$ were taken in state X.

Example: Continuing with the previous example, let the agent in state s_1 execute the action combination A = {toggle-x_3, toggle-x_4}. Then $\delta_{next}(s_1, A) = 1$, since toggle-$x_4$ will finish the first. Thus, $A_{next}(s_1, A)$= {toggle-x_4}. $Y_{next}(s_1, A)$ = {(toggle-x_1,1),(toggle-x_3,4)}. Hence, the probability distribution of states after executing the combination A in state s_1 will be

- $((F, F, F, T, F), Y_{next}(s_1, A))$ probability = 0.9
- $((F, F, F, F, F), Y_{next}(s_1, A))$ probability = 0.1

[5]For a state s, the no-op action is mutex with all actions in $\mathcal{A} \setminus A_s$. In other words, at any decision epoch either a no-op will be started or any combination not involving no-op.

Start and Goal states The start state for our model is $(s_0, \emptyset)$ and the new set of goal states is $\mathcal{G}' = \{(X, \emptyset) | X \in \mathcal{G}\}$. Thus we have modeled a CPTP problem as a concurrent MDP in our new state space. We have redefined the start and goal states, the applicability function, and the probability transition function. Now we can use the techniques of concurrent MDPs to solve our problem. In particular, we can use our Bellman equations as described below.

Bellman equations The set of equations for the solution of a CPTP problem can be written as:

$$J^*(s) = 0, \text{ if } s \in \mathcal{G}' \text{ else} \tag{3}$$
$$J^*(s) = \min_{A \in Ap(s)} \left\{ \delta_{next}(s, A) + \sum_{s'} \mathcal{P}r(s'|s, A) J^*(s') \right\}$$

The main bottleneck in inheriting our previous methods naively is the huge size of the new state space. The fact that all the executing actions with their remaining times appear in the state space blows up the whole state space exponentially. Thus we need to reduce, abstract or aggregate our state space in order to make the problem tractable. In the following subsection, we present an admissible heuristic which can be used to speed the heuristic search.

An admissible heuristic (H) Let $J_{MDP}(X)$ denote the value of a state $X \in \mathcal{S}$ in a traditional MDP with costs of an action equal to its duration. Let $J(s)$ be the value for equivalent CPTP problem with s as in our augmented state space. Define *concurrency* as the number of actions executing in parallel. The following theorem can be used to provide an admissible heuristic for CPTP problems.

Theorem 1 *Let c be the maximum possible concurrency in the domain. Then for $s = (X, Y)$,*

$$\begin{aligned} J^*(s) &\geq \frac{J^*_{MDP}(X)}{c} & \textit{for } Y = \emptyset \\ J^*(s) &\geq \frac{Q^*_{MDP}(X, A_s)}{c} & \textit{for } Y \neq \emptyset \end{aligned}$$

Proof Sketch: Consider any trajectory of make-span L (from a state $s = (X, \emptyset)$ to a goal state) in a CPTP problem using its optimal policy. We can make all concurrent actions sequential by executing them in the chronological order of being started. As all concurrent actions are non-interacting, the outcomes at each stage will have similar probabilities. The maximum make-span of this sequential trajectory will be cL (assuming c actions executing at all points in the semi-MDP trajectory). Hence $J_{MDP}(X)$ using this (possibly non-stationary) policy would be at most $cJ^*(s)$. Thus $J^*_{MDP}(X) \leq cJ^*(s)$. The second inequality can be proven in a similar way.

It can be shown that there are cases where these bounds are tight. For e.g., consider a deterministic planning problem in which in the optimal plan a goal is reached by concurrently taking c actions of unit duration each at the start state (make-span = 1). In the sequential version, same actions would be taken sequentially (make-span = c) — obtaining the desired ratio of the make-spans.

We use these bounds as an admissible starting heuristic for Labeled RTDP.

Implementation and Experiments We have extended our implementation of concurrent MDPs to handle CPTP problems. We have implemented our modified algorithm and as well as our heuristic (H). We present our preliminary results on some problems from a version of the NASA domain, and the Artificial domain (introduced in our AAAI paper) that have actions with durations. The NASA Rover problems had 17 state variables, 21 actions and 19-33 average applicable combinations per state ($Avg(Ap(s))$). The Artificial domain had 14 state variables, 11 actions and $Avg(Ap(s))$ was 146. Figure 3 shows the results. We tested four algorithms on our problems – *Opt*-RTDP: RTDP considering all combinations in each backup, S-RTDP: RTDP using sampled combinations in each Bellman backup, S_H-RTDP and Opt_H-RTDP: Sampled RTDP and *Opt*-RTDP respectively, guided by our heuristic H. We observe that Sampled RTDP performs much better than *Opt*-RTDP, and adding our heuristic guidance speeds the solution in both the cases.

5. Related Work

(Meuleau *et al.* 1998) and (Singh & Cohn 1998) deal with a special type of MDP (called a factorial MDP)[6] that can be represented as a set of smaller weakly coupled MDPs — the separate MDPs are completely independent except for some common resource constraints, and the reward and cost models are purely additive. They describe solutions in which these sub-MDPs are independently solved and the sub-policies are merged to create a global policy. Thus, concurrency of actions of different sub-MDPs is a by-product of their work. All of the work in Factorial MDPs assumes that a weak coupling exists and has been identified, but factoring an MDP is a hard problem in itself. In contrast, our algorithm can handle strongly coupled MDPs and does not require any sub-task decomposition as input.

(Rohanimanesh & Mahadevan 2001) investigate a special class of semi-MDPs in which the action space can be partitioned by (possibly concurrent) *Markov options*. They propose an algorithm based on value-iteration, but their focus is calculating joint termination conditions and rewards received, rather than speeding policy construction. Hence, they consider *all* possible Markov option combinations in a backup. They only experiment on a single, small problem with 400 states.

NASA researchers have developed techniques for solving a harder version of the Rover domain (*e.g.*, with uncertain continuous effects). They propose a *just-in-case* scheduling algorithm, which incrementally adds branches to a straight-line plan. While their work is more general than ours, their solution is heuristic and it is unclear how closely their policies approximate optimality (Bresina *et al.* 2002; Dearden *et al.* 2003). It would be exciting to combine their methods with ours, perhaps by using their heuristic to guide S-RTDP.

[6] Guestrin, Koller and Parr (2001) have investigated similar representations in the context of multiagent planning.

Problem	Reach($\|S\|$)	Avg(Ap(s))	Opt-RTDP	Opt_H-RTDP	S-RTDP	S_H-RTDP
Rover1	~40,000	25	177	153	134	107
Rover2	~17,500	19	311	273	218	138
Rover3	~100,000	33	364	251	220	137
Art1	~65,000	146	428	379	190	97

Figure 3: Preliminary experiment: S-RTDP (RTDP with sampled Bellman backups) performs better than Opt-RTDP (RTDP considering all applicable combinations per backup). Our heuristic (H) speeds up both S-RTDP and Opt-RTDP significantly. All times are in seconds. ($Reach(|S|)$ denotes number of states explored by RTDP, and $Avg(Ap(s))$ denotes the average number of applicable combinations per state)

Recently, Younes and Simmons (2004) have developed a generic test and debug approach which converts a continuous time MDP into a deterministic planning problem. The optimal plan of the deterministic problem is converted back into a policy which can then be repaired if any failure points are identified.

6. Conclusions and Future Work

This paper summarises our techniques for incorporating concurrency in probabilistic planning models. We formally define the concurrent MDP problem and describe two techniques (pruning and sampling) to solve them which obtain orders of magnitude speedup over naive methods. We believe that our sampling techniques will be extremely effective on very large, concurrent MDP problems. Moreover, our sampling and pruning techniques are extremely general and can be applied to other base algorithms like value iteration, LAO* *etc.*

We have also described our initial progress developing methods for solving concurrent probabilistic temporal planning problems. We have presented a solution in terms of RTDP in a modified state space which exploits our methods for solving concurrent MDPs. We have proven a lower bound on the value function which can be used as an admissible heuristic. Initial experiments show that sampled RTDP performs much better than the optimal RTDP, and adding our heuristic guidance speeds the solution even more.

We believe that the key to scaling to much larger problems lies in some intelligent search space compression. We also wish to find other more informative (perhaps inadmissible) heuristics for improving efficiency. We also plan to extend our action representation to a probabilistic version of PDDL2.1, which should not be difficult but will require revising our mutex rules. Finally, we would like to incorporate mixed costs, non-absorbing goals, and stochastic action durations (Concurrent Semi-MDPs).

Acknowledgements

We are thankful to Blai Bonet and Sumit Sanghai for their inputs at various stages of this research. We also thank David Smith, Subbarao Kambhampati, Stanley Kok, Julie Letchner, Benson Limketkai, Jayant Madhavan, and Parag for giving useful comments on an earlier draft. We thank the anonymous reviewers for their worthy comments and suggestions on the paper.

References

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

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

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific.

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. AAAI Press.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *J. Artificial Intelligence Research* 11:1–94.

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*.

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.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Multiagent planning with factored MDPs. In *NIPS'01*, 1523–1530. The MIT Press.

Hansen, E., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

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

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

Meuleau, N.; Hauskrecht, M.; Kim, K.-E.; Peshkin, L.; Kaelbling, L.; Dean, T.; Boutilier, C. 1998. Solving very large weakly coupled Markov Decision Processes. In *AAAI'98*, 165–172.

Rohanimanesh, K., and Mahadevan, S. 2001. Decision-Theoretic planning with concurrent temporally extended actions. In *UAI'01*, 472–479.

Singh, S., and Cohn, D. 1998. How to dynamically merge markov decision processes. In *NIPS'98*. The MIT Press.

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

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