Concurrent Probabilistic Temporal Planning

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 processes (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 efficient methods for solving probabilistic planning problems with concurrent, durative actions. We adapt the formulation of *Concurrent MDPs*, MDPs which allow multiple instantaneous actions to be executed simultaneously. We add explicit action durations into the concurrent MDP model by encoding the problem as a concurrent MDP in an augmented state space. We present two novel admissible heuristics and one inadmissible heuristic to speed up the basic concurrent MDP algorithm. We also develop a novel notion of *hybridizing* an optimal and an approximate algorithm to yield a hybrid algorithm, which quickly generates high-quality policies. Experiments show that all our heuristics speedup the policy construction significantly. Furthermore, our approximate hybrid algorithm runs up to two orders of magnitude faster than other methods, while producing policies whose make-spans are typically within 5% of optimal.

1. Introduction

Recent progress has yielded new planning 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. While today's planners can handle large problems with *deterministic* concurrent durative actions (JAIR Special Issue 2003), and semi-MDPs provide a clear framework for durative actions in the face of uncertainty (Bertsekas 1995), few researchers have considered concurrent, uncertain, durative actions — the focus of this paper.

Consider a Mars rover with 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 to effective execution, 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.

The framework of *Markov decision processes* (MDPs) is the dominant model for formulating probabilistic planning problems. In the traditional case of a single action per decision epoch, state-space heuristic search and dynamic programming have proven quite effective (Bonet & Geffner 2000; Hansen & Zilberstein 2001). However, allowing multiple concurrent actions at a time point inflicts an exponential blowup on all of these techniques. Our previous work on concurrent MDPs (Mausam & Weld 2004) introduced several methods to manage this exponential blowup. However, in their current form concurrent MDPs (CoMDPs) do not handle explicit action durations. The actions are supposed to be instantaneous (or unit length), and the agent may not start a new action while another action is already executing. Instead, the agent must wait for all the recently-started actions to finish before new action(s) can be started. This restriction is fine if all actions are unit length, but leads to suboptimal policies when actions have differing lengths. For example, in order to save execution time, the Mars rover might wish to execute sequential set up actions (*e.g.*, turning on the camera, warming it up, focusing, *etc.*.) *concurrent with navigation to the next location*.

In this paper, we define *concurrent probabilistic temporal planning* - in short, *CPTP*. This model extends our previous CoMDP framework by incorporating explicit action durations. Specifically, we extend the technique of *Sampled real-time dynamic programming* (Sampled RTDP) (Barto, Bradtke, & Singh 1995; Bonet & Geffner 2003; Mausam & Weld 2004) to generate high-quality CPTP policies. This paper makes the following contributions:

- We model a CPTP problem as a concurrent MDP in an augmented state space.
- We present three, novel heuristics that can guide RTDP.
  - We prove the *maximum concurrency* (*MC*) heuristic is admissible by bounding the optimal cost of the solution in terms of the solution of a sequential planning problem and maximum possible concurrency in the domain.
  - The *average concurrency* (*AC*) heuristic is more informed, but inadmissible.

  - The *eager effects* (*EE*) heuristic is computed by solving a relaxed version of the CPTP problem, which assumes that effects of actions are predictable before action completion.

- We propose the novel idea of *hybridization*, *i.e.* of combining two policy creation algorithms to yield a single, fast, approximation algorithm, which has the best of both worlds. Our hybrid algorithm for CPTP combines partial CPTP and CoMDP policies to focus its optimization efforts on the most frequent branches.
- We experimentally compare the speed/quality trade-offs offered by the heuristics on three domains. Our most interesting result is that on large problems our hybrid algorithm may save a factor of 88x in policy-creation time, with only a small sacrifice in the resulting policy quality.

2. Background

Following (Bonet & Geffner 2003), we define a *Markov decision process* as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0, \gamma \rangle$ 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. $\mathcal{C} : \mathcal{A} \rightarrow \Re^+$ is the cost model, $\mathcal{G} \subseteq \mathcal{S}$ is the set of absorbing goal states, s_0 is the start state, and $\gamma \in [0, 1]$ is the discount factor.

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 discounted cost (over an infinite horizon) incurred to reach a goal state. Note that a *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 can be derived from the value function, J^*: $\mathcal{S} \rightarrow \Re$, which satisfies the following pair of *Bellman equations*:

$$J^*(s) = 0, \text{ if } s \in \mathcal{G} \text{ else} \tag{1}$$

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

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: $J_{n+1}(s) = \min_{a \in Ap(s)} \left\{ \mathcal{C}(a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s, a) J_n(s') \right\}$.

Value iteration and other similar algorithms (*e.g.* Policy Iteration) tend to be quite slow since they search the entire state space. *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 RTDP (Barto, Bradtke, & Singh 1995), which is our focus.

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; 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 focuses 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.

Concurrent Markov Decision Processes (CoMDP) In the previous work (Mausam & Weld 2004), we extended traditional MDPs to allow concurrent actions. Since some actions interfere with each other, we ban certain combinations adopting 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[2], 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 of $a_2; a_1$.

An *action combination*, A, is a set of one or more non-mutex actions to be executed in parallel. The cost model for a CoMDP, denoted $\mathcal{C}_\| : \mathcal{P}(\mathcal{A}) \rightarrow \Re^+$, returns the cost of concurrently executing several actions in a state. In (Mausam & Weld 2004), we considered cost models, which were a weighted sum of time and resource components. For the purposes of this paper, we ignore the resource component and focus on the expected *make-span* of a policy — how long it will take to execute. Thus, we define the cost of a combination of actions to be: $\mathcal{C}_\|(\{a_1, a_2, ..., a_k\}) = \max_{i=1..k}\{\mathcal{C}(a_i)\}$. Note that this way, action durations are embedded in the cost function, and the model assumes that a new set of actions may not be executed until all members

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

[2] Note that an action's transition function is typically conditioned on various features of the state (*conditional effects*). These features are considered to be a part of conjunctive preconditions for the purpose of mutex calculation.

of the previous set have terminated.[3]

The applicability function for CoMDPs, denoted as $Ap_{\|}$, now has range $\mathcal{P}(\mathcal{P}(\mathcal{A}))$; it is defined in terms of the applicability function for MDPs as $Ap_{\|}(s) = \{A \subseteq \mathcal{A} | \forall a, a' \in A,\ a, a' \in Ap(s) \wedge\ \neg\mathrm{mutex(a, a')}\}$

Let $A = \{a_1, a_2, \ldots, a_k\}$ be an action combination that is applicable in s. Since CoMDPs 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 Bellman equations represents the solution to a CoMDP:

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

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

These equations are the same as in a traditional MDP, except that instead of considering single actions for backup in a state, we consider all applicable action combinations. Thus, only this small change needs to be made to traditional algorithms (*e.g.*, value iteration, LAO*, Labeled RTDP).

Sampled RTDP The number of action combinations in each Bellman backup is exponential in $|\mathcal{A}|$. To efficiently handle this blowup, one may refrain from backing up all combinations when evaluating the "min" in Equation 2. *Sampled RTDP* performs backups on a random set of combinations, choosing from a distribution which favors "likely combinations." This distribution is generated by: 1) using combinations that were previously discovered to have low $Q_{\|}$-values (recorded by *memoizing* best combinations per state, after each iteration); 2) calculating the $Q_{\|}$-values of all applicable single actions (using the current value function) and then biasing the sampling of combinations to choose the ones which contain actions with low $Q_{\|}$-values.

Since the system does not 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 $J_{\|}(s)$ values are neither admissible nor monotonic. As a result, Sampled RTDP no longer guarantees termination and optimality. However, experiments have shown that Sampled RTDP usually terminates quickly, and returns values that are extremely close to the optimal (Mausam & Weld 2004).

3. Extending to Durative Actions

We now incorporate action durations in concurrent probabilistic planning problems. As a start we consider the input model similar to that of concurrent MDPs except that action costs ($\mathcal{C}_{\|}(a)$) are replaced by their durations ($\Delta(a)$). We study the objective of minimizing the expected time (*makespan*) of reaching a goal. For now, we assume deterministic action durations:

Assumption 1 *All actions have deterministic durations.*

Consider the sample domain of the Mars rover in Figure 1. The rover has to accomplish two experiments: sampling the rock and taking an image. But when extended, the arm blocks the camera's field of view. Moreover, the camera needs to be calibrated before capturing the image, and the calibrate action succeeds only half the time. Each action may have a distinct duration as shown in the figure.

Note that a CPTP is distinct from a semi-MDP as a semi-MDP models actions with *uncertain* durations, while in CPTP durations are deterministic. However, a CPTP allows for concurrent executions of actions whereas a semi-MDP does not.

Assumption 2 *All action durations are integer-valued.*

This assumption has a minimal effect on expressiveness because one can convert a problem with rational durations into one that abides Assumption 2 by scaling all durations by the g.c.d. of the denominators. In case of irrational durations, one can always find an arbitrarily close approximation to the original problem by approximating the irrational durations by rational numbers.

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

Assumption 3 *All actions follow the following model:*

- *The effects of an action are realized at some unknown point during action execution, and thus can be used only once the action has completed.*
- *The preconditions must hold at the beginning of an action.*
- *The preconditions and the features on which the action's transition function is conditioned must remain unchanged while the action is being executed, unless the action itself is modifying them.*

These restrictions are consistent with our previous definition of concurrency. Specifically, the mutex definitions (of CoMDPs) hold and are required under these assumptions. As an illustration, consider Figure 2. It describes a situation in which two actions with interfering preconditions and effects can not be executed concurrently. To see why not, suppose initially arm_out was false and two actions take_image and extend_arm were started at time 2 and 4, respectively. As ¬arm_out is a precondition of take_image, whose duration is 5, it needs to remain false until time 7. But extend_arm may produce its effects anytime between 4 and 9, which may conflict with the preconditions of the other executing action. Hence, extend_arm and take_image cannot be executed concurrently.

Aligned Epoch Search Space A simple way to formulate CPTP is to model it as a standard CoMDP, in which action costs are set to their durations and the cost of a combination is the maximum duration of the constituent actions. This formulation is distinct from a CPTP in an important way: Consider the actual executions of these policies. In

[3] The fact that the start of all new actions must be aligned with the termination of previous actions explains why we use the $\|$ symbol to distinguish the cost ($\mathcal{C}_{\|}$) and value functions ($J_{\|}$) *etc.*. of this model.

State variables: calibrated, arm_out, image_taken, sample_taken

Action (a)	$\Delta(a)$	Preconditions	Effects	Prob.
extend_arm	5	true	arm_out	1
take_sample	1	arm_out	sample_taken$\wedge\neg$arm_out	0.9
			no change	0.1
calibrate	1	$\neg$calibrated	calibrated	0.5
			no change	0.5
take_image	5	calibrated $\wedge\neg$arm_out	image_taken	1

Goal: image_taken $\wedge$ sample_taken

Figure 1: Durative Probabilistic STRIPS definition of a simple CPTP problem

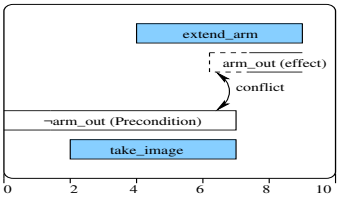

Figure 2: A sample execution demonstrating conflict due to interfering preconditions and effects.

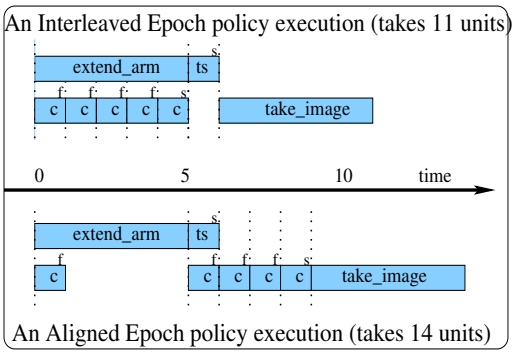

Figure 3: Comparison of times taken in a sample execution of an interwoven-epoch policy and an aligned-epoch policy. In both trajectories the calibrate (c) action fails four times before succeeding. Because the aligned policy must wait for all actions to complete before starting any more, it takes more time than the interwoven policy, which can start more actions in the middle.

the aligned-epoch case, once a combination of actions is started at a state, the next decision can be taken only when the effects of all actions have been observed (hence the name *aligned-epochs*). In contrast, at a decision epoch in the optimal execution for a CPTP problem, many actions may be midway in their execution. We have to explicitly take into account these actions and their remaining execution times when making a subsequent decision. Thus, the state space of CPTP is substantially different from that of the simple aligned-epoch model. The difference is illustrated in Figure 3. The figure compares the trajectories in which the calibrate (c) actions fails for four consecutive times before succeeding. In the figure, "f" and "s" denote failure and success of uncertain actions, respectively and "ts" denotes the take_sample action. The vertical dashed lines represent the time-points when an action is started.

Note: due to Assumption 3, it is sufficient to consider a new decision epoch only at a time-point when one or more actions complete. Thus, using Assumption 2 we infer that these decision epochs will be discrete (integer). 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 Assumptions 1- 3 reduce our search space considerably.

Interwoven Epoch 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 interwoven state[4] $s \in \mathcal{S}_{\underline{-}}$ be an ordered pair $\langle X, Y\rangle$ where $X \in \mathcal{S}$ and $Y = \{(a,\delta)|a \in \mathcal{A}, 0 < \delta \leq \Delta(a)\}$. 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 their remaining times until completion "δ". Thus the overall interwoven-epoch search space is $\mathcal{S}_{\underline{-}} = \mathcal{S} \times \bigotimes_{a\in\mathcal{A}} \left(\{a\} \times Z_{\Delta(a)}\right)$, where $Z_{\Delta(a)}$ represents the set $\{0, 1, \ldots, \Delta(a) - 1\}$ and $\bigotimes$ denotes the Cartesian product over multiple sets.

Also define A_s to be the set of actions already in execution. In other words, A_s is a projection of Y :

$$A_s = \{a|(a,\delta) \in Y \wedge s = \langle X, Y\rangle\}$$

Example: In our domain in Figure 1, a state (say s_1) has all state variables false, and the action extend_arm was started 3 units ago. Such a state would be represented as $\langle X_1, Y_1\rangle$ with X_1=(F, F, F, F)[5] and Y_1={(extend_arm, 2)}. The set A_{s_1} would be {extend_arm}.

To allow the possibility of simply waiting for some action to complete execution, that is, not executing any action at some decision epochs, we augment the set $\mathcal{A}$ with a *no-op* action. We allow no-op to be applicable in all states s = $\langle X, Y\rangle$ where $Y \neq \emptyset$ (*i.e.* states in which some action is still being executed)[6]. 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 interwoven applicability set can be defined as:

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

Transition Function We also need to define the probability transition function, $\mathcal{P}r_{\underline{-}}$, for the interwoven state space. At some decision epoch let the agent be in state s = (X, Y). Suppose that the agent decides to execute an action combination A. Define Y_{new} as the set similar to Y but consisting of the actions just starting. Formally Y_{new} = $\{(a, \Delta(a))|a \in A\}$. In this system, our next decision epoch will be the smallest time after which any executing action

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

[5] The four state variables are listed in the order: calibrated, arm_out, image_taken, and sample_taken.

[6] 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.

completes. Let us call this time $\delta_{next}(s, A)$. Notice that $\delta_{next}(s, A)$ depends on both executing and newly started actions. Formally,

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

Moreover, multiple actions may complete simultaneously. 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=$\langle X, Y\rangle$ and let s'=$\langle X', Y'\rangle$. The transition function for CPTP can now be defined as:

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

In other words, executing an action combination A in state $s = \langle X, Y\rangle$ takes the agent to a decision epoch $\delta_{next}(s, A)$ ahead in time, that is, the first time when some combination $A_{next}(s, A)$ completes. This lets us calculate $Y_{next}(s, A)$: the new set of actions still executing with their remaining times. 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 = {calibrate}. Then $\delta_{next}(s_1, A) = 1$, since calibrate will finish the first. Thus, $A_{next}(s_1, A)$= {calibrate}. $Y_{next}(s_1, A)$ = {(extend_arm,1)}. Hence, the probability distribution of states after executing the combination A in state s_1 will be a uniform distribution over:
$\langle(T, F, F, F), \{(\text{extend\_arm}, 1)\}\rangle$
$\langle(F, F, F, F), \{(\text{extend\_arm}, 1)\}\rangle$

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 a CPTP problem as a CoMDP in our interwoven 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 CoMDPs 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:

$$J^*_{\underline{=}}(s) = 0, \text{ if } s \in \mathcal{G}_{\underline{=}} \text{ else} \tag{3}$$

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

The main bottleneck in inheriting our previous methods (*e.g.* Sampled RTDP) naively is the huge size of the interwoven state space. In the worst case (when all actions can be executed concurrently) the size of the state space is $|\mathcal{S}| \times \left(\prod_{a\in\mathcal{A}} \Delta(a)\right)$. We get this bound by observing that for each action a, there are $\Delta(a)$ number of possibilities : either a is not executing or it is and has remaining times $1, 2, \ldots, \Delta(a) - 1$.

Thus we need to reduce, abstract or aggregate our state space to make the problem tractable. We now present several heuristics which can be used to speed the search.

4. Admissible Heuristics

We present two admissible heuristics that can be used as the initial cost function for our Sampled RTDP algorithm. The first heuristic (maximum concurrency) solves the underlying MDP and is thus quite efficient to compute. However it is typically less informative than our second heuristic (eager effects) which requires the solution of a relaxed CoMDP in a state space larger than the underlying MDP state space. We now discuss the details of the two heuristics.

Maximum Concurrency Heuristic

We prove that the optimal expected cost in a traditional (serial) MDP divided by the maximum number of actions that can be executed in parallel is a lower bound for the expected make-span of reaching a goal in a CPTP problem. Let $J(X)$ denote the value of a state $X \in \mathcal{S}$ in a traditional MDP with costs of an action equal to its duration. Let $J^*_{\underline{=}}(s)$ be the value for equivalent CPTP problem with s as in our interwoven-epoch state space. Let *concurrency* of a state be the maximum number of actions that could be executed in the state concurrently. We define *maximum concurrency of a domain* (c) as the maximum concurrency of any state in the domain. Thus, c represents the maximum number of actions that could possibly execute in parallel at any point. The following theorem can be used to provide an admissible heuristic for CPTP problems.

Theorem 1 *Let $s = \langle X, Y\rangle$,*

$$J^*_{\underline{=}}(s) \geq \frac{J^*(X)}{c} \quad \textit{for } Y = \emptyset$$

$$J^*_{\underline{=}}(s) \geq \frac{Q^*(X, A_s)}{c} \quad \textit{for } Y \neq \emptyset$$

Proof Sketch: Consider any trajectory of make-span L (from a state $s = \langle X, \emptyset\rangle$ 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 $J(X)$ using this (possibly non-stationary) policy would be at most $cJ^*_{\underline{=}}(s)$. Thus $J^*(X) \leq cJ^*_{\underline{=}}(s)$. The second inequality can be proven in a similar way. ■

There are cases where these bounds are tight. For example, consider a deterministic planning problem in which the optimal plan is concurrently executing c actions each of unit duration (make-span = 1). In the sequential version, the same actions would be taken sequentially (make-span = c).

Following this theorem, the maximum concurrency (*MC*) heuristic for a state $s = \langle X, Y\rangle$ is defined as follows:

$$\text{if } Y = \emptyset\ H_{MC}(s) = \frac{J^*(X)}{c} \text{ else } H_{MC}(s) = \frac{Q^*(X, A_s)}{c}$$

The maximum concurrency c can be calculated by a static analysis of the domain. Thus the time taken for each heuristic is the time required for solving the MDP. In our implementation, we do this calculation on demand, as more states

are visited, by starting the MDP from the current state. Each RTDP run can be seeded by the previous value function, thus no computation is thrown away and only the relevant part of the state space is explored.

Average Concurrency Heuristic Instead of using maximum concurrency c in the above heuristic we use the average concurrency in the domain (c_a) to get the average concurrency (*AC*) heuristic. The *AC* heuristic is not admissible, but in our experiments it is typically a more informed heuristic.

Eager Effects Heuristic

Given the CPTP problem, we can generate a relaxed CoMDP by making the effects of actions, which would otherwise be visible only in the future, be known right away — thus the name eager effects (*EE*). A state for this relaxed CoMDP is $\langle X, \delta \rangle$ where X is an MDP state and δ is an integer. Intuitively, $\langle X, \delta \rangle$ signifies that the agent will *reach* state X after time δ units. Thus, we have discarded the information about which actions are executing and when they will individually end; we only record that all of them will have ended after time δ units and that the agent will reach the state X (possibly with some probability).

The applicable set of a relaxed state is defined as $Ap_{EE}(\langle X, \delta \rangle) = Ap_{\|}(X)$. Note that this new problem really is a relaxation because certain actions are applicable that would be mutex to the currently executing actions (in the original problem). We explain this in detail in the discussion of Theorem 2. The goal states in the relaxed problem are $\{\langle X, 0 \rangle | X \in \mathcal{G}\}$, *i.e.* all states that are goals in the underlying MDP and no action is executing.

Finally, the transition probabilities are redefined. The state-component of the resulting relaxed states denotes that the effects of the combination currently started have been realized. Whereas, the time component does not advance to the end of all actions, rather it advances to the completion of the shortest action, generating a new decision epoch for starting new actions.

Formally, suppose that we execute a combination A in state $s = \langle X, \delta \rangle$. Let δ^{EE}_{last} be the length of the longest action in A. Let δ^{EE}_{first} be the the length of the shortest action in A. Define δ^{EE}_{next} as

$$\begin{aligned}
\delta^{EE}_{next} &= \delta^{EE}_{last} - \delta^{EE}_{first} && \text{if } \delta = 0 \\
&= \delta^{EE}_{last} - \delta && \text{if } 0 < \delta \leq \delta^{EE}_{first} \\
&= \delta^{EE}_{last} - \delta^{EE}_{first} && \text{if } \delta^{EE}_{first} < \delta \leq \delta^{EE}_{last} \\
&= \delta - \delta^{EE}_{first} && \text{if } \delta > \delta^{EE}_{last}
\end{aligned}$$

The transition function can now be defined using the above definition of δ^{EE}_{next}.

$$\begin{aligned}
\mathcal{P}r_{EE}(\langle X', \delta' \rangle | \langle X, \delta \rangle, A) &= 0 && \text{if } \delta' \neq \delta^{EE}_{next} \\
&= \mathcal{P}r_{\|}(X' | X, A) && \text{if } \delta' = \delta^{EE}_{next}
\end{aligned}$$

The cost of executing a combination in the relaxed state represents the duration by which the current time moves forward. It is equal to $\max(\delta, \delta^{EE}_{last}) - \delta^{EE}_{next}$.

Based on the solution of the relaxed CoMDP we can compute a heuristic value for our original CPTP problem. Let $s = \langle X, Y \rangle$ be a state in the interwoven-epoch space. Let J^*_{EE} be the optimal cost function for the relaxed CoMDP. Then, the *EE* heuristic function is computed as follows:

$$H_{EE}(s) = \sum_{X' \in \mathcal{S}} \mathcal{P}r_{\|}(X' | X, A_s) J^*_{EE}(\langle X', \delta^{EE}_{last} \rangle)$$

Theorem 2 *The* EE *heuristic value is non-overestimating, thus admissible.*

The admissibility stems from the fact that, in the relaxed problem, we have eased two essential features. First, we have assumed that the present state contains the results of actions that would actually complete in the future. So, there is actually more information in the relaxed problem, than in the original problem; thus the decisions taken are more informed and lead to a goal in less time. Secondly, since we lose the information of which actions were executing in the domain, we have to allow for all applicable combinations in the MDP state. That is, all the actions that were mutex with the actions executing (in the real problem) are also allowed. Thus, this is a relaxation of the original problem, and the time taken to reach the goal will be shorter. Hence, overall the heuristic value is admissible.

We further speed the convergence of the relaxed problems by initializing its value function with simple heuristics. It is easy to show that the following inequalities hold:

$$\begin{aligned}
J^*_{EE}(\langle X, \delta \rangle) &\geq \delta \\
J^*_{EE}(\langle X, \delta \rangle) &\geq J^*_{EE}(\langle X, \delta' \rangle) && \text{if } \delta' \leq \delta \\
J^*_{EE}(\langle X, \delta \rangle) &\geq J^*_{EE}(\langle X, \delta' \rangle) - (\delta' - \delta) && \text{if } \delta' > \delta
\end{aligned}$$

So for any state $\langle X, \delta \rangle$ we can set the initial value function to δ or max it with other values computed using the above equations for the states $\langle X, \delta' \rangle$ that have already been visited. We can use the current values of these states instead of J^*_{EE} to compute these seed values.

Comparing the Two Heuristics

Theorem 3 *Neither of the two heuristics (eager effects or maximum concurrency) dominates the other.*

Proof: Consider a deterministic problem in which two parallel sets of actions in the order a_1, a_2, a_3 and b_1, b_2, b_3 need to be executed to achieve the goal. Let a_1, a_3, b_2, and b_3 be of duration n and the rest be unit duration. If the maximum concurrency in the domain is 2, then H_{MC} value of start state is $(4n+2)/2$ which is also the optimal value $(2n+1)$. The H_{EE} value of the start state calculates to $n+2$. This is an example of a problem in which the *MC* heuristic is more informative. If however in a similar problem, the only actions that could be executed concurrently are a_3 and b_3 then the maximum concurrency remains 2. So the H_{MC} does not change, although the optimal plan is now longer. But the H_{EE} value calculates to $3n+2$ which is optimal. ■

In spite of the theorem, in practice *EE* is consistently more informative than *MC* on the domains we tried. But, the computation times required for the two heuristics are quite different. *MC* requires the computation of the underlying MDP which is a relatively easy problem to solve. Whereas, *EE* requires the computation of a problem which has a larger

search space than even the underlying CoMDP. Thus the computation of *EE* heuristic can take a long time, at times to the extent that the advantage of the more informative heuristic is lost in the complex heuristic computation.

5. Hybrid Algorithm

In this section we present an approximate method to solve CPTP problems. While there are many possible approximation methods, our technique exploits the intuition that it is best to focus computation on the most probable branches in the current policy's reachable space. The danger of this approach is the chance that, during execution, the agent might end up in an unlikely branch, which has been poorly explored; indeed it might blunder into a dead-end in such a case. This is undesirable, because such an apparently attractive policy might have a true expected make-span of infinity. Since, we wish to avoid this case, we explore the desirable notion of *propriety*.

Propriety: A policy is *proper* at a state if it is guaranteed to lead, eventually, to the goal state (*i.e.*, it avoids all dead ends and cycles) (Barto, Bradtke, & Singh 1995). We define a planning algorithm *proper* if it always produces a proper policy (when one exists) for the initial state.

We now describe an anytime approximation algorithm, which quickly generates a proper policy and uses any additional available computation time to improve the policy, focusing on the most likely trajectories.

Hybridization Our algorithm is created by *hybridizing* two other policy creation algorithms. Indeed, our novel notion of hybridization is both general and powerful, applying to many MDP-like problems; however, in this paper we focus on the use of hybridization for CPTP. Hybridization uses an anytime algorithm like RTDP to create a policy for frequently visited states, and uses a faster (and presumably suboptimal) algorithm for the infrequent states.

For the case of CPTP, our algorithm hybridizes the RTDP algorithms for interwoven-epoch and aligned-epoch models. With aligned-epochs, RTDP converges relatively quickly, because the state space is smaller, but the resulting policy is suboptimal *for the CPTP problem*, because the policy waits for *all* currently executing actions to terminate before starting any new actions. In contrast, RTDP for interwoven-epochs generates the optimal policy, but it takes much longer to converge. Our insight is to run RTDP on the interwoven space long enough to generate a policy which is good on the common states, but stop well before it converges in every state. Then, to ensure that the rarely explored states have a proper policy, we substitute the aligned policy, returning this *hybrid* policy.

Thus the key question is how to decide which states are well explored and which are not. We define the *familiarity* of a state s to be the number of times it has been visited in previous RTDP trials. Any reachable state whose familiarity is less than a constant, k, has an aligned policy created for it. Furthermore, if a dead-end state is reached using the greedy interwoven policy, then we create an aligned policy for the immediate precursors of that state. If a cycle is detected[7], then we compute an aligned policy for all the states which are part of the cycle.

```
Algorithm Hybrid(r, k, m) {
  ∀s ∈ S_= initialize J_=(s) with an admissible heuristic;
  Repeat {
    Perform m RTDP trials;
    Compute Hybrid policy (π) using interwoven-epoch policy
      for k-familiar states and aligned-epoch policy otherwise;
    Clean π by removing all dead-ends and cycles;
    J^π_=⟨s_0, ∅⟩ ← Evaluation of π from the start state;
  } Until ( (J^π_=(⟨s_0,∅⟩) − J_=(⟨s_0,∅⟩)) / J_=(⟨s_0,∅⟩) < r )
  Return hybrid policy π;
}
```

Figure 4: Pseudo-code for the hybrid algorithm

We have not yet said how the hybrid algorithm terminates. Use of RTDP helps us in defining a very simple termination condition with a parameter that can be varied to achieve the desired *closeness* to optimality as well. The intuition is very simple. Consider first, optimal Labeled RTDP. This starts with an admissible heuristic and guarantees that the value of the start state, $J_=(\langle s_0, \emptyset \rangle)$, remains admissible (thus less than or equal to optimal). In contrast, the hybrid policy's make-span is always longer than or equal to optimal. Thus as time progresses, these values approach the optimal make-span from opposite sides. Whenever the two values are within an *optimality ratio (r)*, we know that the algorithm has found a solution, which is close to the optimal.

Finally, evaluation of the hybrid policy is done using simulation, which we perform after a fixed number of m RTDP trials. The algorithm is summarized in Figure 4. One can see that this combined policy is proper for two reasons: 1) if the policy at a state is from the aligned policy, then it is proper because the RTDP for aligned-epoch model was run to convergence, and 2) for the rest of the states we have explicitly ensured that there are no cycles or dead-ends.

6. Experiments

In this section, we compare the computation time and solution quality of six methods: interwoven Sampled RTDP with no heuristic (0), with the maximum concurrency (*MC*), average concurrency (*AC*), and eager effects (*EE*) heuristics, the hybrid (*H*) algorithm and Sampled RTDP on the aligned-epoch (*AE*) model. We also use an artificial domain to see if the relative performance of the techniques varies with the amount of concurrency in the domain.

Experimental Setup We test our algorithms on problems in three domains. The first domain is a probabilistic, durative variant of NASA Rover domain from the 2002 AIPS Planning Competition, in which there are multiple objects to be photographed and various rocks to be tested with resulting data to be communicated back to the base station. Cameras need to be focused, and arms need to be positioned before usage. Since the rover has multiple arms and multiple cameras, the domain is highly parallel. We generate prob-

[7] In our implementation cycles are detected using simulation.

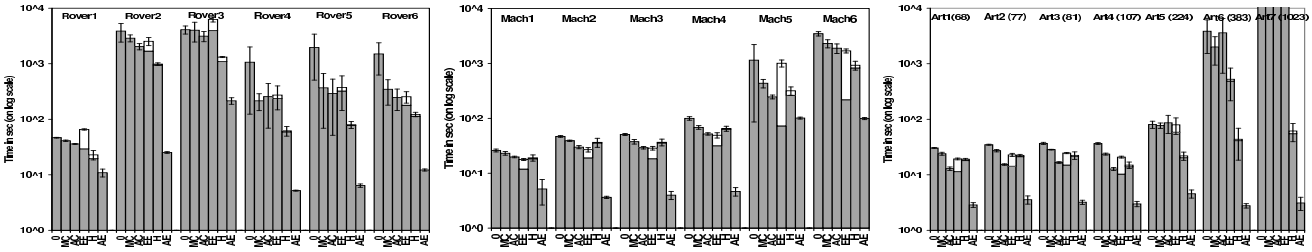

Figure 5: (a,b,c): Running times (on a log scale) for the Rover, Machineshop and Artificial domain, respectively. For each problem the six bars represent the times taken by the algorithms: S-RTDP with no (0) heuristic, with *MC*, *AC*, *EE* heuristics, hybrid algorithm and aligned-epoch RTDP, respectively. The white bar on *EE* denotes the portion of time taken by heuristic computation and on *H* denotes the portion of time taken by aligned-epoch RTDP.

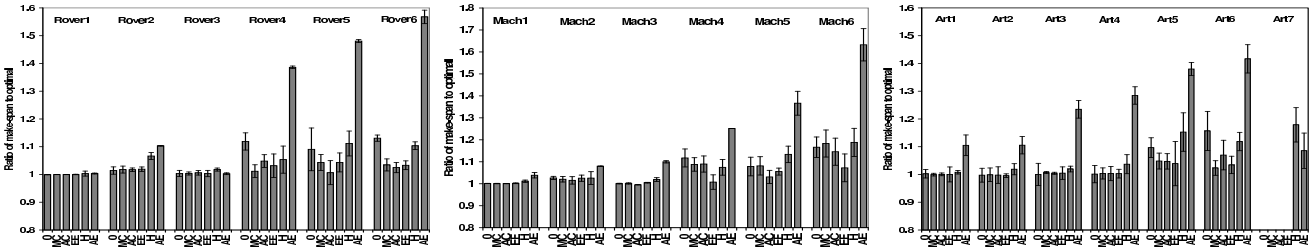

Figure 6: (a,b,c): Comparison of make-spans of the solution found with the optimal (plotted as 1 on the y-axes) for Rover, Machineshop and Artificial domains, respectively. All algorithms except *AE* produce solutions, which are quite close to the optimal.

lems with 17-21 state variables and 12-18 actions, whose duration range between 1 and 20. The problems have between 15,000-700,000 reachable states in the interwoven-epoch state space, $\mathcal{S}_{\pm}$.

We also test on a probabilistic temporal version of Machineshop domain with multiple subtasks (*e.g.*, shape, paint, polish *etc.*), which need to be performed on different objects using different machines. Machines can perform in parallel, but not all are capable of every task and they cannot perform on the same object concurrently. Different pieces need to be transported from one machine to another for different subtasks. We test on problems with 18-26 variables and up to 500,000 reachable states with action durations being 1-10.

Finally, we test on an artificial domain. In this domain, some Boolean variables need to be toggled; however, toggling is probabilistic in nature. Moreover, certain pairs of actions have conflicting preconditions and thus, by varying the number of mutex actions we may control the domain's degree of parallelism. All the problems in this domain have 14 state variables and 17,000-40,000 reachable states and durations of actions between 1 and 3.

We use our implementation of Sampled RTDP[8], which is implemented on top of Labeled RTDP in GPT, as the base CoMDP solver. We implement all heuristics: maximum concurrency (H_{MC}), average concurrency (H_{AC}), and eager effects (H_{EE}) for the initialization of the value function. We calculate these heuristics on demand for the states visited, instead of computing the complete heuristic for the whole state space at once. We also implement the hybrid (*H*) algorithm in which the initial value function was set to the H_{MC} heuristic. The parameters r, k, and m are kept at 0.05, 100 and 500, respectively. We test each of these algorithms on a number of problem instances from the three domains, which we generate by varying the number of objects, degrees of parallelism, durations of the actions and distances to the goal.

Comparison of Running Times Figures 5(a, b, and c) show the variations in the running times for the algorithms on different problems in Rover, Machineshop and Artificial domains, respectively. The first four bars represent the base Sampled RTDP without any heuristic, with H_{MC}, with H_{AC}, and with H_{EE}, respectively. The fifth bar represents the hybrid algorithm (using the H_{MC} heuristic) and the sixth bar is computation of the aligned-epoch Sampled RTDP with costs set to the maximum action duration. The white region in the fourth bar represents the time required for the H_{EE} computation. The white region in the fifth bar represents the time taken for the aligned-epoch RTDP computations in the hybrid algorithm. The error bars represent 95% confidence intervals on the running times. Note that the plots are on a log scale.

We notice that *AE* solves the problems extremely quickly; this is natural since the aligned-epoch space is smaller. Use of both H_{MC} and H_{AC} always speeds search in the $\mathcal{S}_{\pm}$ model. Using H_{EE} speeds up the solutions for most problems, but sometimes the heuristic computation takes a huge amount of time and the overall running time is not competitive. Comparing the heuristics amongst themselves, we find that H_{AC} mostly performs faster than H_{MC} — presumably because H_{AC} is a more informed heuristic in practice, although at the cost of being inadmissible. We find a couple of cases in which H_{AC} doesn't perform better; this could be because it is focusing the search in the incorrect region, given its inadmissible nature. For the the Rover domain

[8]Note that policies returned by Sampled RTDP are not guaranteed to be optimal. Thus all the implemented algorithms are approximate. We can replace Sampled RTDP by pruned RTDP (Mausam & Weld 2004) if optimality is desired.

H_{EE} does not perform as well as H_{MC} whereas for Machineshop domain for most problems H_{EE} outperforms even H_{AC}. For the Artificial domain, the performance typically lies in between H_{MC} and H_{AC}.

For the Rover domain, the hybrid algorithm performs fastest. In fact, the speedups are dramatic compared to other methods. In other domains, the results are more comparable for small problems. However, for large problems in these two domains, hybrid outperforms the others by a huge margin. In fact for the largest problem in Artificial domain, none of the heuristics are able to converge (within a day) and only the hybrid algorithm (and *AE*) converge to a solution.

Figure 7 shows the speedups obtained by various algorithms compared to the basic Sampled RTDP in $\mathcal{S}_{\underline{=}}$. In the Rover and Artificial domains the speedups obtained by *H* and *AE* are much more prominent than in the Machineshop domain. Averaging over all domains, *H* produces a 10x speedup and *AE* produces more than a 100x speedup.

Algos	Speedup compared w/ heuristic-free S-RTDP in $\mathcal{S}_{\underline{=}}$			
	Rover	Machineshop	Artificial	Average
MC	3.016764	1.545418	1.071645	1.877942
AC	3.585993	2.173809	1.950643	2.570148
EE	2.99117	1.700167	2.447969	2.379769
H	10.53418	2.154863	16.53159	9.74021
AE	135.2841	16.42708	241.8623	131.1911

Figure 7: The time taken by each algorithm divided by the time taken by $\mathcal{S}_{\underline{=}}$ S-RTDP with no heuristics. Our heuristics produce 2-3 times speedups. The hybrid algo produces about a 10x speedup. *AE* produces 100x speedup, but sacrifices solution quality.

Comparison of Solution Quality Figures 6(a, b, and c) show the quality of the policies obtained by the same six methods on the same domains. We measure quality by simulating the generated policy across multiple trials, and reporting the average time taken to reach the goal. We plot the ratio of the so-measured expected make-span to the optimal expected make-span[9]. Figure 8 presents solution qualities for each method, averaged over all problems in a domain. We note that the aligned-epoch (*AE*) policies usually yield significantly longer make-spans (e.g., 25% longer); thus one must make a quality sacrifice for their speedy policy construction. In contrast, the hybrid algorithm extorts only a small sacrifice in quality in exchange for its speed.

Variation with Concurrency Figure 5(c) represents our attempt to see if the relative performance of the algorithms changed with increasing concurrency. Along the top of the figure, by the problem names, are numbers in brackets; these list the average number of applicable combinations in each MDP state, $Avg_{s \in \mathcal{S}_{\underline{=}}} |Ap(s)|$, and range from 68 to 1023 concurrent actions. Note that for the difficult problems with a lot of parallelism, S-RTDP in $\mathcal{S}_{\underline{=}}$ slows dramatically, regardless of heuristic. In contrast, the hybrid algorithm is still able to quickly produce a policy, and at almost no loss in quality (Figure 6(c)).

[9]In some large problems, the optimal algorithm did not converge. For those, we take as optimal, the best policy found in our runs.

Algos	Average Quality			
	Rover	Machineshop	Artificial	Average
0	1.059625	1.065078	1.042561	1.055704
MC	1.018405	1.062564	1.013465	1.031478
AC	1.017141	1.046391	1.020523	1.028019
EE	1.021806	1.027887	1.012897	1.020863
H	1.059349	1.075534	1.059201	1.064691
AE	1.257205	1.244862	1.254407	1.252158

Figure 8: Overall solution quality produced by all algorithms. Note that all algorithms except aligned-epoch (*AE*) produce policies whose quality is quite close to optimal. On average *AE* produces make-spans that are about 125% of the optimal.

7. Related Work

Younes and Simmons (2004) handle a formalism with many desirable features *e.g.* continuous time, more expressive goal language, and uncertain action durations. Their planner solves a deterministic version of the problem and uses the resulting plan as a candidate policy which can then be repaired if failure points are identified. While apparently fast, this method does not guarantee convergence, proximity to the optimal, or propriety.

NASA researchers propose a *just-in-case* scheduling algorithm, which incrementally adds branches to a straight-line plan (Bresina *et al.* 2002; Dearden *et al.* 2003). While they handle continuous variables and uncertain continuous effects, their solution is heuristic and the quality of their policies is unknown. Also, since they consider only limited contingencies, their solutions are improper.

More recently, Prottle (Little 2004) solves problems with more expressive domain description language, but for a finite horizon — thus for an acyclic state space. Prottle uses an RTDP type search guided by heuristics computed using probabilistic variants of planning graph. It would be exciting to combine their method with ours, perhaps by using their heuristics to guide our search.

(Aberdeen, Thiebaux & Zhang 2004) develop a CoMDP-like formalism and apply it to military operations planning. The search is guided by domain-specific heuristics.

Rohanimanesh and 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.

Fast generation of parallel plans has also been investigated in deterministic settings, and (Jensen & Veloso 2000) extend it to disjunctive uncertainty. Morris and Muscettola (1999) study dynamic control of plans with temporal uncertainty.

8. Conclusions and Future Work

This paper summarizes our techniques for incorporating concurrency with durative actions in the probabilistic planning models. We formally define the concurrent probabilistic temporal planning problem, and develop two modified state spaces (aligned-epoch ($\mathcal{S}_{\|}$) and interwoven-epoch ($\mathcal{S}_{\underline{=}}$)), which allow us search for an optimal policy, using Sampled RTDP or other means (*e.g.*, Labeled RTDP, LAO*,

value iteration, etc.) Our experiments show that, while we can search the aligned space faster, the interwoven model usually yields a much better policy, one which generates a much lower make-span.

We define two new heuristics (maximum concurrency (*MC*) and eager effects (*EE*)), and prove that they are admissible. *MC* is quite fast to compute and consistently speeds up the policy construction. *EE*, however, requires more time to compute and thus speeds up the convergence only in some domains. We also describe an inadmissible but quite informative heuristic (average concurrency (*AC*); this heuristic speeds the RTDP search even more, and with practically no loss in the solution quality.

We also develop the general technique of hybridizing two MDP algorithms. Hybridizing interwoven-epoch and aligned-epoch policy creation yields a much more efficient algorithm, one which is still proper (*i.e.*, its policies guarantee that the goal will be reached whenever possible). Also, our hybrid algorithm has a parameter, which can be varied to trade-off speed against optimality. In our experiments, the hybrid algorithm quickly produces near-optimal solutions. For larger problems, the speedups over other algorithms are quite significant. The hybrid algorithm can also be used in an anytime fashion thus producing good quality *proper* policies within a desired time. Thus, we expect that the algorithm would be very effective in solving large problems.

Future Work Scaling to significantly larger problems will require new techniques for reducing the huge search space. We are currently looking into approximate search space compression and aggregation techniques.

In order to model actions that temporarily provide resources, we plan to extend our action representation to a probabilistic version of PDDL2.1; this shouldn't be difficult, but will require revisions in our mutex rules.

We also wish to extend our algorithms to richer models like rewards, non-absorbing goals, mixed costs, stochastic action durations *etc.* Our techniques are general enough to be applicable in all these scenarios. For example consider the mixed cost optimization problem, in which the objective function is the sum (or a linear combination) of time and resource costs. Here, an equivalent MC heuristic can be computed by solving another MDP, which minimizes resource costs, and then adding the converged value to the MC heuristic reported herein. A hybrid algorithm can be easily developed in the same manner.

More generally, we believe that our hybridization technique is very general and applicable to a wide range of problems. For instance, we could create a proper, anytime approximation algorithm for Concurrent MDPs (CoMDPs) by hybridizing one of the RTDP algorithms of (Mausam & Weld 2004) with a traditional MDP algorithm. Similarly, a hybrid POMDP algorithm can be constructed by hybridizing RTDP for POMDPs with the policy for the equivalent MDP. We wish to explore these further.

Acknowledgments

We thank Blai Bonet for providing the source code of GPT. We also thank Ronen Brafman, Krzysztof Gajos, Subbarao Kambhampati, Daniel Lowd, Nicolas Meuleau, Parag, Sumit Sanghai, Pradeep Shenoy, David Smith, and all anonymous reviewers for giving useful comments on an earlier draft of the paper. We thank Puneet Khanduri for lending his laptop in the time of need. The work was supported by NSF grant IIS-0307906 and ONR grant N00014-02-1-0932.

References

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

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: A forward chaining approach. *IJCAI'01*, p417.

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72, p81.

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

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. *IJCAI'95*, p1636.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. *AIPS'00*, p52.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. *ICAPS'03*, p12.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *JAIR* 11, p1.

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

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D. E.; and Washington, R. 2003. Incremental Contingency Planning. *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. *ECP'01*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20, p61.

Hansen, E., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129, p35.

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

2003. Special Issue on the 3rd International Planning Competition, JAIR, Volume 20.

Jensen, R. M., and Veloso, M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *JAIR* 13, p189.

Little, I. 2004. Probabilistic temporal planning. *Honours thesis*, Australian National Univesity.

Mausam, and Weld, D. 2004. Solving concurrent Markov decision processes. *AAAI'04*, p716.

Morris, P. H., and Muscettola, N. 1999. Managing temporal uncertainty through waypoint controllability. *IJCAI'99*, p1253.

Rohanimanesh, K., and Mahadevan, S. 2001. Decision-Theoretic planning with concurrent temporally extended actions. *UAI'01*, p472.

Smith, D., and Weld, D. 1999. Temporal graphplan with mutual exclusion reasoning. *IJCAI'99*, p326.

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