Planning with Durative Actions in Stochastic Domains

Mausam MAUSAM@CS.WASHINGTON.EDU
Daniel S. Weld WELD@CS.WASHINGTON.EDU
Dept of Computer Science and Engineering
Box 352350, University of Washington
Seattle, WA 98195 USA

Abstract

Probabilistic planning problems are typically modeled as a Markov Decision Process (MDP). MDPs, while an otherwise expressive model, allow only for sequential, non-durative actions. This poses severe restrictions in modeling and solving a real world planning problem. We extend the MDP model to incorporate — 1) simultaneous action execution, 2) durative actions, and 3) stochatic durations. We develop several algorithms to combat the computational explosion introduced by these features. The key theoretical ideas used in building these algorithms are — modeling a complex problem as an MDP in extended state/action space, pruning of irrelevant actions, sampling of relevant actions, using informed heuristics to guide the search, hybridizing different planners to achieve benefits of both, approximating the problem and replanning. Our empirical evaluation illuminates the different merits in using various algorithms, *viz.*, optimality, empirical closeness to optimality, theoretical error bounds, and speed.

1. Introduction

Recent progress achieved by planning researchers has yielded new algorithms which relax, individually, many of the classical assumptions. For example, successful temporal planners like SGPlan, SAPA, *etc.* (Chen, Wah, & Hsu, 2006; Do & Kambhampati, 2003) are able to model actions that take time, and probabilistic planners like GPT, LAO*, SPUDD, *etc.* (Bonet & Geffner, 2005; Hansen & Zilberstein, 2001; Hoey, St-Aubin, Hu, & Boutilier, 1999) can deal with actions with probabilistic outcomes, *etc.* However, in order to apply automated planning to many real-world domains we must eliminate larger groups of the assumptions in concert. For example, NASA researchers note that optimal control for a NASA Mars rover requires reasoning about uncertain, concurrent, durative actions and a mixture of discrete and metric fluents (Bresina, Dearden, Meuleau, Smith, & Washington, 2002). While today's planners can handle large problems with *deterministic* concurrent durative actions, and MDPs provide a clear framework for *non-concurrent* durative actions in the face of uncertainty, few researchers have considered concurrent, uncertain, durative actions — the focus of this paper.

As an example consider the NASA Mars rovers, Spirit and Oppurtunity. They have 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. Furthermore, all of their actions, *e.g.*, moving between locations and setting up experiments, take time. In fact, these temporal durations are themselves uncertain — the rover might lose its way and take a long time to reach another location, *etc.* To be able to solve the planning problems encountered by a rover, our planning framework needs to explicitly model all these domain constructs — concurrency, actions with uncertain outcomes and uncertain durations.

In this paper we present a unified formalism that models all these domain features together. *Concurrent Markov Decision Processes* (CoMDPs) extend MDPs by allowing multiple actions per decision epoch. We use CoMDPs as the base to model all planning problems involving concurrency. Problems with durative actions, *concurrent probabilistic temporal planning* (CPTP), are formulated as CoMDPs in an extended state space. The formulation is also able to incorporate the uncertainty in durations in the form of probabilistic distributions.

Solving these planning problems poses several computational challenges: concurrency, extended durations, and uncertainty in those durations all lead to explosive growth in the state space, action space and branching factor. We develop two techniques, Pruned RTDP and Sampled RTDP to address the blowup from concurrency. We also develop the "DUR" family of algorithms to handle stochastic durations. These algorithms explore different points in the running time *vs.* solution-quality tradeoff. The different algorithms propose several speedup mechanisms such as — 1) pruning of provably sub-optimal actions in a Bellman backup, 2) intelligent sampling from the action space, 3) admissible and inadmissible heuristics computed by solving non-concurrent problems, 4) hybridizing two planners to obtain a hybridized planner that finds good quality solution in intermediate running times, 5) approximating stochastic durations by their mean values and replanning, 6) exploiting the structure of multi-modal duration distributions to achieve higher quality approximations.

The rest of the paper is organized as follows: In section 2 we discuss the fundamentals of MDPs and the real-time dynamic programming (RTDP) solution method. In Section 3 we describe the model of Concurrent MDPs. Section 4 investigates the theoretical properties of the temporal problems. Section 5 explains our formulation of the CPTP problem for deterministic durations. The algorithms are extended for the case of stochastic durations in Section 6. Each section is supported with an empirical evaluation of the techniques presented in that section. In Section 7 we survey the related work in the area. We conclude with future directions of research in Sections 8 and 9.

2. Background

Planning problems under probabilistic uncertainty are often modeled using Markov Decision Processes (MDPs). Different research communities have looked at slightly different formulations of MDPs. These versions typically differ in objective functions (maximizing reward *vs.* minimizing cost), horizons (finite, infinite, indefinite) and action representations (DBN *vs.* parametrized action schemata). All these formulations are very similar in nature, and so are the algorithms to solve them. Though, the methods proposed in the paper are applicable to all the variants of these models, for clarity of explanation we assume a particular formulation, known as the *stochastic shortest path problem* (Bertsekas, 1995).

We define a *Markov decision process* ($\mathcal{M}$) as a tuple $\langle \mathcal{S}, \mathcal{A}, Ap, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0 \rangle$ in which

- $\mathcal{S}$ is a finite set of discrete states. We use factored MDPs, *i.e.*, $\mathcal{S}$ is compactly represented in terms of a set of state variables.

- $\mathcal{A}$ is a finite set of actions.

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

Action	Precondition	Effect	Probability
toggle-x_1	$\neg p_{12}$	$x_1 \leftarrow \neg x_1$	1
toggle-x_2	p_{12}	$x_2 \leftarrow \neg x_2$	1
toggle-x_3	true	$x_3 \leftarrow \neg x_3$	0.9
		no change	0.1
toggle-x_4	true	$x_4 \leftarrow \neg x_4$	0.9
		no change	0.1
toggle-p_{12}	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 MDP with potential parallelism

- Ap defines an applicability function. $Ap : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, denotes the set of actions that can be applied in a given state ($\mathcal{P}$ represents the power set).
- $\mathcal{P}r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is the transition function. We write $\mathcal{P}r(s'|s,a)$ to denote the probability of arriving at state s' after executing action a in state s.
- $\mathcal{C} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re^+$ is the cost model. We write $\mathcal{C}(s,a,s')$ to denote the cost incurred when the state s' is reached after executing action a in state s.
- $\mathcal{G} \subseteq \mathcal{S}$ is a set of absorbing goal states, *i.e.*, the process ends once one of these states is reached.
- s_0 is a start state.

We assume full observability, *i.e.*, the execution system has complete access to the new state *after* an action has been performed. We seek to find an optimal, stationary policy — *i.e.*, a function π: $\mathcal{S} \rightarrow \mathcal{A}$ that minimizes the expected cost (over an indefinite horizon) incurred to reach a goal state. Note that any *cost function*, J: $\mathcal{S} \rightarrow \Re$, mapping states to the expected cost of reaching a goal state defines a policy as follows:

$$\pi_J(s) = \underset{a \in Ap(s)}{\text{argmin}} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left\{\mathcal{C}(s,a,s') + J(s')\right\} \tag{1}$$

The *optimal* policy derives from the optimal cost function, J^*, which satisfies the following pair of *Bellman equations*.

$$\begin{aligned} J^*(s) &= 0, \text{ if } s \in \mathcal{G} \text{ else} \\ J^*(s) &= \min_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a)\left\{\mathcal{C}(s,a,s') + J^*(s')\right\} \end{aligned} \tag{2}$$

For example, Figure 1 defines a simple MDP where four state variables ($x_1, \ldots, x_4$) need to be set using toggle actions. Some of the actions, *e.g.*, toggle-x_3 are probabilistic.

Various algorithms have been developed to solve MDPs. *Value iteration* is a dynamic programming approach in which the optimal cost function (the solution to equations 2) is calculated as the limit of a series of approximations, each considering increasingly long action sequences. If $J_n(s)$

is the cost of state s in iteration n, then the cost of state s in the next iteration is calculated with a process called a *Bellman backup* as follows:

$$J_{n+1}(s) = \min_{a \in Ap(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s,a) \left\{ \mathcal{C}(s,a,s') + J_n(s') \right\} \tag{3}$$

Value iteration terminates when $\forall s \in \mathcal{S},\ |J_n(s) - J_{n-1}(s)| \leq \epsilon$, and this termination is guaranteed for $\epsilon > 0$. Furthermore, in the limit, the sequence of $\{J_i\}$ is guaranteed to converge to the optimal cost function, J^*, regardless of the initial values as long as a goal can be reached from every reachable state with non-zero probability. Unfortunately, value iteration tends to be quite slow, since it explicitly updates every state, and $|\mathcal{S}|$ is exponential in the number of domain features. One optimization restricts search to the part of state space reachable from the initial state s_0. Two algorithms exploiting this *reachability analysis* 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. An RTDP *trial* is a path starting from s_0, following the greedy policy and updating the costs 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 relevant state) is slow because less likely (but potentially important) states get updated infrequently. Furthermore, RTDP is not guaranteed to terminate. *Labeled RTDP* (LRTDP) 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 ϵ-approximation of the optimal cost function (for states reachable using the optimal policy) if the initial cost function is admissible, all costs ($\mathcal{C}$) positive and a goal reachable from all reachable states with non-zero probability.

MDPs are a powerful framework to model stochastic planning domains. However, MDPs make two unrealistic assumptions — 1) all actions need to be executed sequentially, and 2) all actions are instantaneous. Unfortunately, there are many real-world domains where these assumptions are unrealistic. For example, concurrent actions are essential for a Mars rover, since instruments can be turned on, warmed up and calibrated while the rover is moving, and using other instruments for transmitting data. Moreover, the action durations are non-zero and stochastic — the rover might lose its way while navigating and may take a long time to reach its destination; it may make multiple attempts before finding the accurate arm placement. In this paper we successively relax these two assumptions and build models and algorithms that can scale up in spite of the additional complexities imposed by the more general models.

3. Concurrent Markov Decision Processes

We define a new model, *Concurrent MDP* (CoMDP), which allows multiple actions to be executed in parallel. This model is different from semi-MDPs and generalized state semi-MDPs (Younes & Simmons, 2004b) in that it does not incorporate action durations explicitly. CoMDPs focus on adding concurrency in an MDP framework. The input to a CoMDP is slightly different from that of an MDP – $\langle \mathcal{S}, \mathcal{A}, Ap_{\|}, \mathcal{P}r_{\|}, \mathcal{C}_{\|}, \mathcal{G}, s_0 \rangle$. The new applicability function, probability model and cost

($Ap_{\|}$, $\mathcal{P}r_{\|}$ and $\mathcal{C}_{\|}$ respectively) encode the distinction between allowing sequential executions of single actions versus the simultaneous executions of sets of actions.

3.1 The Model

The set of states ($\mathcal{S}$), set of actions ($\mathcal{A}$), goals ($\mathcal{G}$) and the start state (s_0) follow the input of an MDP. The difference lies in the fact that instead of executing only one action at a time, we may execute multiple of them. Let us define an *action combination*, A, as a set of one or more actions to be executed in parallel. With an action combination as a new unit operator available to the agent, the CoMDP takes the following new inputs

- $Ap_{\|}$ defines the new applicability function. $Ap_{\|} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{A}))$, denotes the set of action *combinations* that can be applied in a given state.
- $\mathcal{P}r_{\|} : \mathcal{S} \times \mathcal{P}(\mathcal{A}) \times \mathcal{S} \rightarrow [0, 1]$ is the transition function. We write $\mathcal{P}r_{\|}(s'|s, A)$ to denote the probability of arriving at state s' after executing action combination A in state s.
- $\mathcal{C}_{\|} : \mathcal{S} \times \mathcal{P}(\mathcal{A}) \times \mathcal{S} \rightarrow \Re^{+}$ is the cost model. We write $\mathcal{C}_{\|}(s, A, s')$ to denote the cost incurred when the state s' is reached after executing action combination A in state s.

In essence, a CoMDP takes an action combination as a unit operator instead of a single action. Our approach is to convert a CoMDP into an equivalent MDP ($\mathcal{M}_{\|}$) that can be specified by the tuple $\langle \mathcal{S}, \mathcal{P}(\mathcal{A}), Ap_{\|}, \mathcal{P}r_{\|}, \mathcal{C}_{\|}, \mathcal{G}, s_0 \rangle$ and solve it using the known MDP algorithms.

3.2 Case Study: CoMDP over Probabilistic STRIPS

In general a CoMDP could require an exponentially larger input than does an MDP, since the transition model, cost model and the applicability function are all defined in terms of action combinations as opposed to actions. A compact input representation for a general CoMDP is an interesting, open research question for the future. In this work, we consider a special class of compact CoMDP – one that is defined naturally via a domain description very similar to the *probabilistic STRIPS* representation for MDPs (Boutilier, Dean, & Hanks, 1999).

Given a domain encoded in probabilistic STRIPS we can compute a safe set of co-executable actions. Under this safe semantics, the probabilistic dynamics gets defined in a consistent way as we describe below.

3.2.1 Applicability Function

We first discuss how to compute the sets of actions that can be executed in parallel since some actions may conflict with each other. We adopt the classical planning notion of mutual exclusion (Blum & Furst, 1997) and apply it to the *factored* action representation of probabilistic STRIPS. Two distinct actions are *mutex* (may not be executed concurrently) if in any state one of the following occurs:

1. they have inconsistent preconditions
2. an outcome of one action conflicts with an outcome of the other
3. the precondition of one action conflicts with the (possibly probabilistic) effect of the other.

4. the effect of one action possibly modifies a feature upon which another action's transition function is conditioned upon.

Additionally, an action is never mutex with itself. In essence, the non-mutex actions do not interact — the effects of executing the sequence $a_1; a_2$ equals those for $a_2; a_1$ — and so the semantics for parallel executions is clear.

Example: Continuing with Figure 1, toggle-x_1, toggle-x_3 and toggle-x_4 can execute in parallel but toggle-x_1 and toggle-x_2 are mutex as they have conflicting preconditions. Similarly, toggle-x_1 and toggle-p_{12} are mutex as the effect of toggle-p_{12} interferes with the precondition of toggle-x_1. If toggle-x_4's outcomes depended on toggle-x_1 then they would be mutex too, due to point 4 above. For example, toggle-x_4 toggle-x_1 will be mutex if the effect of toggle-x_4 was as follows: "if toggle-x_1 then the probability of $x_4 \leftarrow \neg x_4$ is 0.9 else 0.1". $\square$

The applicability function is defined as the set of action-combinations, A, such that each action in A is independently applicable in s and all of the actions are pairwise non-mutex with each other. Note that pairwise concurrency is sufficient to ensure problem-free concurrency of all multiple actions in A. Formally $Ap_{||}$ can be defined in terms of our original definition Ap as follows:

$$Ap_{||}(s) = \{A \subseteq \mathcal{A} | \forall a, a' \in A,\ a, a' \in Ap(s) \wedge\ \neg \mathrm{mutex(a, a')}\} \tag{4}$$

3.2.2 Transition Function

Let $A = \{a_1, a_2, \ldots, a_k\}$ be an action combination applicable in s. Since none of the actions are mutex, the transition function may be calculated by choosing any arbitrary order in which to apply them as follows:

$$\mathcal{P}r_{||}(s'|s, A) = \sum_{s_1, s_2, \ldots s_k \in \mathcal{S}} \ldots \sum \mathcal{P}r(s_1|s, a_1) \mathcal{P}r(s_2|s_1, a_2) \ldots \mathcal{P}r(s'|s_{k-1}, a_k) \tag{5}$$

While we define the applicability function and the transition function by allowing only a *consistent* set of actions to be executable concurrently, there are alternative definitions possible. For instance, one might be willing to allow executing two actions together if the probability that they conflict is very small. A conflict may be defined as two actions asserting contradictory effects or one negating the precondition of the other. In such a case, a new state called *failure* could be created such that the system transitions to this state in case of a conflict. And the transition may be computed to reflect a low probability transition to this failure state.

Although we impose that the model be conflict-free, most of our techniques don't actually depend on this assumption explicitly and extend to general CoMDPs.

3.2.3 Cost model

We make a small change to the probabilistic STRIPS representation. Instead of defining a single cost ($\mathcal{C}$) for each action, we define it additively as a sum of *resource* and *time* components as follows:

- Let t be the *durative* cost, *i.e.*, cost due to time taken to complete the action.

- Let r be the *resource* cost, *i.e.*, cost of resources used for the action.

Assuming additivity we can think of cost of an action $\mathcal{C}(s,a,s') = t(s,a,s') + r(s,a,s')$, to be sum of its time and resource usage. Hence, the cost model for a combination of actions in terms of these components may be defined as:

$$\mathcal{C}_{\|}(s, \{a_1, a_2, ..., a_k\}, s') = \sum_{i=1}^{k} r(s, a_i, s') + \max_{i=1..k} \{t(s, a_i, s')\} \tag{6}$$

For example, a Mars rover might incur lower cost when it preheats an instrument while changing locations than if it executes the actions sequentially, because the total time is reduced while the energy consumed does not change.

3.3 Solving a CoMDP with MDP algorithms

We have taken a concurrent MDP that allowed concurrency in actions and formulated it as an equivalent MDP, $\mathcal{M}_{\|}$, in an extended action space. For the rest of the paper we will use the term CoMDP to also refer to the equivalent MDP $\mathcal{M}_{\|}$.

3.3.1 Bellman equations

We extend Equations 2 to a set of equations representing the solution to a CoMDP:

$$\begin{aligned} &J^*_{\|}(s) = 0, \text{ if } s \in \mathcal{G} \text{ else} \\ &J^*_{\|}(s) = \min_{A \in Ap_{\|}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}r_{\|}(s'|s,A) \left\{ \mathcal{C}_{\|}(s,A,s') + J^*_{\|}(s') \right\} \end{aligned} \tag{7}$$

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 worst-case exponential in $|\mathcal{A}|$, efficiently solving a CoMDP requires new techniques. Unfortunately, there is no structure to exploit easily, since an optimal action for a state from a classical MDP solution may not even appear in the optimal action *combination* for the associated concurrent MDP.

Theorem 1 *All actions in an optimal combination for a CoMDP ($\mathcal{M}_{\|}$) may be individually sub-optimal for the MDP $\mathcal{M}$.*

Proof: In the domain of Figure 1 let us have an additional action toggle-x_{34} that toggles both x_3 and x_4 with probability 0.5 and toggles exactly one of x_3 and x_4 with probability 0.25 each. Let all the actions take one time unit each, and therefore cost of any action combination is one as well. Let the start state be x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0 and p_{12} = 1. For the MDP $\mathcal{M}$ the only optimal action for the start state is toggle-x_{34}. However, for the CoMDP $\mathcal{M}_{\|}$ the optimal combination is {toggle-x_3, toggle-x_4}. $\square$

3.4 Pruned Bellman Backups

Recall that during a trial, Labeled RTDP performs Bellman backups in order to calculate the costs of applicable actions (or in our case, action combinations) and then chooses the best action (combination); we now describe two pruning techniques that reduce the number of backups to be computed.

Let $Q_{\|}(s, A)$ be the expected cost incurred by executing an action combination A in state s and then following the greedy policy, *i.e.*

$$Q_{\|_n}(s, A) = \sum_{s' \in \mathcal{S}} \mathcal{P}r_{\|}(s'|s, A) \left\{ \mathcal{C}_{\|}(s, A, s') + J_{\|_{n-1}}(s') \right\} \tag{8}$$

A Bellman update can thus be rewritten as:

$$J_{\|_n}(s) = \min_{A \in Ap_{\|}(s)} Q_{\|_n}(s, A) \tag{9}$$

3.4.1 Combo-Skipping

Since the number of applicable action combinations can be exponential, we would like to prune suboptimal combinations. The following theorem imposes a lower bound on $Q_{\|}(s, A)$ in terms of the costs and the $Q_{\|}$-values of single actions. For this theorem the costs of the actions may depend only on the action and not the starting or ending state, *i.e.*, for all states $\forall s, s'$ $\mathcal{C}(s, a, s') = \mathcal{C}(a)$.

Theorem 2 *Let $A = \{a_1, a_2, \ldots, a_k\}$ be an action combination which is applicable in state s. For a CoMDP over probabilistic STRIPS, if costs are dependent only on actions and $Q_{\|_n}$ values are monotonically non-decreasing then*

$$Q_{\|}(s, A) \geq \max_{i=1..k} Q_{\|}(s, \{a_i\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\}) \right)$$

Proof:

$$\begin{aligned} Q_{\|_n}(s, A) &= \mathcal{C}_{\|}(A) + \sum_{s'} \mathcal{P}r_{\|}(s'|s, A) J_{\|_{n-1}}(s') && \text{(using Eqn. 8)} \\ \Rightarrow \sum_{s'} \mathcal{P}r_{\|}(s'|s, A) J_{\|_{n-1}}(s') &= Q_{\|_n}(s, A) - \mathcal{C}_{\|}(A) && (10) \end{aligned}$$

$$\begin{aligned} Q_{\|_n}(s, \{a_1\}) &= \mathcal{C}_{\|}(\{a_1\}) + \sum_{s''} \Pr(s''|s, a_1) J_{\|_{n-1}}(s'') \\ &\leq \mathcal{C}_{\|}(\{a_1\}) + \sum_{s''} \Pr(s''|s, a_1) \left[\mathcal{C}_{\|}(\{a_2\}) + \sum_{s'''} \Pr(s'''|s'', a_2) J_{\|_{n-2}}(s''') \right] \\ & && \text{(using Eqns. 8 and 9)} \\ &= \mathcal{C}_{\|}(\{a_1\}) + \mathcal{C}_{\|}(\{a_2\}) + \sum_{s'''} \mathcal{P}r_{\|}(s'''|s, \{a_1, a_2\}) J_{\|_{n-2}}(s''') \\ &\leq \sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\}) + \sum_{s'} \mathcal{P}r_{\|}(s'|s, A) J_{\|_{n-k}}(s') && \text{(repeating for all actions in } A) \\ &= \sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\}) + [Q_{\|_{n-k+1}}(s, A) - \mathcal{C}_{\|}(A)] && \text{(using Eqn. 10)} \end{aligned}$$

Replacing n by $n + k - 1$

$$\begin{aligned} Q_{\|_n}(s,A) &\geq Q_{\|_{n+k-1}}(s,\{a_1\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) \\ &\geq Q_{\|_n}(s,\{a_1\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) && \text{(monotonicity of } Q_{\|_n}) \\ &\geq \max_{i=1..k} Q_{\|_n}(s,\{a_i\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) \end{aligned}$$

□

The proof above assumes equation 5 from probabilistic STRIPS. The following corollary can be used to prune suboptimal action combinations:

Corollary 3 *Let $\lceil J_{\|_n}(s) \rceil$ be an upper bound of $J_{\|_n}(s)$. If*

$$\lceil J_{\|_n}(s) \rceil < \max_{i=1..k} Q_{\|_n}(s,\{a_i\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right)$$

then A cannot be optimal for state s in this iteration.

Proof: Let $A^*_n = \{a_1, a_2, \ldots, a_k\}$ be the optimal combination for state s in this iteration n. Then,

$$\begin{aligned} \lceil J_{\|_n}(s) \rceil &\geq J_{\|_n}(s) \\ J_{\|_n}(s) &= Q_{\|_n}(s, A{*}_n) \end{aligned}$$

Combining with Theorem 2

$$\lceil J_{\|_n}(s) \rceil \geq max_{i=1..k} Q_{\|_n}(s,\{a_i\}) + \mathcal{C}_{\|}(A^*_n) - \left(\sum_{i=1}^{k} \mathcal{C}_{\|}(\{a_i\})\right) \square$$

Corollary 3 justifies a pruning rule, *combo-skipping*, that preserves optimality in any iteration algorithm that maintains cost function monotonicity. This is powerful because all Bellman-backup based algorithms preserve monotonicity when started with an admissible cost function. To apply combo-skipping, one must compute all the $Q_{\|}(s,\{a\})$ values for single actions a that are applicable in s. To calculate $\lceil J_{\|_n}(s) \rceil$ one may use the optimal combination for state s in the previous iteration (A_{opt}) and compute $Q_{\|_n}(s, A_{opt})$. This value gives an upper bound on the value $J_{\|_n}(s)$.

Example: Consider Figure 1. Let a single action incur unit cost, and let the cost of an action combination be: $\mathcal{C}_{\|}(A) = 0.5 + 0.5|A|$. Let state s = (1,1,0,0,1) represent the ordered values x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, and p_{12} = 1. Suppose, after the n^{th} iteration, the cost function assigns the values: $J_{\|_n}(s)$ = 1, $J_{\|_n}(s_1$=(1,0,0,0,1)) = 2, $J_{\|_n}(s_2$=(1,1,1,0,1)) = 1, $J_{\|_n}(s_3$=(1,1,0,1,1)) = 1. Let A_{opt} for state s be {toggle-x_3, toggle-x_4}. Now, $Q_{\|_{n+1}}(s$, {toggle-x_2}) = $\mathcal{C}_{\|}$({toggle-x_2}) + $J_{\|_n}(s_1)$ = 3 and $Q_{\|_{n+1}}(s, A_{opt})$ = $\mathcal{C}_{\|}(A_{opt})$ + 0.81×0 + 0.09×$J_{\|_n}(s_2)$ + 0.09×$J_{\|_n}(s_3)$ + 0.01×$J_{\|_n}(s)$ = 1.69. So now we can apply Corollary 3 to skip combination {toggle-x_2, toggle-x_3} in this iteration, since using toggle-x_2 as a_1, we have $\lceil J_{\|_{n+1}}(s) \rceil$ = $Q_{\|_{n+1}}(s, A_{opt})$ = 1.69 $\leq$ 3 + 1.5 - 2 = 2.5. □

Experiments show that combo-skipping yields considerable savings. 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.

3.4.2 Combo-Elimination

We adapt the action elimination theorem from traditional MDPs (Bertsekas, 1995) to prove a similar theorem for CoMDPs.

Theorem 4 *Let A be an action combination which is applicable in state s. Let $\lfloor Q^*_{\|}(s, A)\rfloor$ denote a lower bound of $Q^*_{\|}(s, A)$. If $\lfloor Q^*_{\|}(s, A)\rfloor > \lceil J^*_{\|}(s)\rceil$ then A is never the optimal combination for state s.*

Proof: Because a CoMDP is an MDP in a new action space, the original proof for MDPs (Bertsekas, 1995) holds after replacing an action by an 'action combination'. □

In order to apply the theorem for pruning, one must be able to evaluate the upper and lower bounds. By using an admissible cost function when starting RTDP search (or in value iteration, LAO* *etc.*), the current cost $J_{\|_n}(s)$ is guaranteed to be a lower bound of the optimal cost; thus, $Q_{\|_n}(s, A)$ will also be a lower bound of $Q^*_{\|}(s, A)$. Thus, it is easy to compute the left hand side of the inequality. To calculate an upper bound of the optimal $J^*_{\|}(s)$, one may solve the MDP $\mathcal{M}$, *i.e.*, the traditional MDP that forbids concurrency. This is much faster than solving the CoMDP, and yields an upper bound on cost, because forbidding concurrency restricts the policy to use a strict subset of legal action combinations. Notice that combo-elimination can be used for all general MDPs and is not restricted to only CoMDPs over probabilistic STRIPS.

Example: Continuing with the previous example, let A={toggle-x_2} then $Q_{\|_{n+1}}(s, A) = \mathcal{C}_{\|}(A) + J_{\|_n}(s_1) = 3$ and $\lceil J^*_{\|}(s)\rceil$ = 2.222 (from solving MDP $\mathcal{M}$). As 3 > 2.222, A can be eliminated for state s in all remaining iterations. □

Used in this fashion, combo-elimination requires the additional overhead of optimally solving the single-action MDP $\mathcal{M}$. Since algorithms like RTDP exploit state-space reachability to limit computation to relevant states, we do this computation incrementally, as new states are visited by our algorithm.

Combo-elimination also requires computation of the current value of $Q_{\|}(s, A)$ (for the lower bound of $Q^*_{\|}(s, A)$); this differs from combo-skipping which avoids this computation. However, once combo-elimination prunes a combination, it never needs to be reconsidered. Thus, there is a tradeoff: should one perform an expensive computation, hoping for long-term pruning, or try a cheaper pruning rule with fewer benefits? Since Q-value computation is the costly step, we adopt the following heuristic: "First, try combo-skipping; if it fails to prune the combination, attempt combo-elimination; if it succeeds, never consider it again". We also tried implementing some other heuristics, such as: 1) If some combination is being skipped repeatedly, then try to prune it altogether with combo-elimination. 2) In every state, try combo-elimination with probability p. Neither alternative performed significantly better, so we kept our original (lower overhead) heuristic.

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.

3.5 Sampled Bellman Backups

Since the fundamental challenge posed by CoMDPs is the explosion of action combinations, sampling is a promising method to reduce the number of Bellman backups required per state. We describe a variant of RTDP, called *sampled RTDP*, which performs backups on a random set of action combinations[1], choosing from a distribution that favors combinations that are *likely to be optimal*. We generate our distribution by:

1. using combinations that 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 cost function) and then biasing the sampling of combinations to choose the ones that contain actions with low $Q_{||}$-values.

Algorithm 1 Sampled Bellman Backup(state, m) *//returns the best combination found*

```
1: list $l = \emptyset$ //a list of all applicable actions with their values
2: for all $\text{action} \in \mathcal{A}$ do
3:     compute $Q_{||}(\text{state}, \{\text{action}\})$
4:     insert $\langle a, 1/Q_{||}(\text{state}, \{\text{action}\})\rangle$ in $l$
5: for all $i \in [1..m]$ do
6:     $\text{newcomb}$ = SampleComb($\text{state}, i, l$);
7:     compute $Q_{||}(\text{state}, \text{newcomb})$
8: clear $\text{memoizedlist}[\text{state}]$
9: compute $Q_{min}$ as the minimum of all $Q_{||}$ values computed in line 7
10: store all combinations $A$ with $Q_{||}(\text{state}, A) = Q_{min}$ in $\text{memoizedlist}[\text{state}]$
11: return the first entry in $\text{memoizedlist}[\text{state}]$
```

Function 2 SampleComb(state, i, l) *//returns i^{th} combination for the sampled backup*

```
1: if $i \leq$ size($\text{memoizedlist}[\text{state}]$) then
2:     return $i^{th}$ entry in $\text{memoizedlist}[\text{state}]$ //return the combination memoized in previous iteration
3: $\text{newcomb} = \emptyset$
4: repeat
5:     randomly sample an action $a$ from $l$ proportional to its value
6:     insert $a$ in $\text{newcomb}$
7:     remove all actions mutex with $a$ from $l$
8:     if $l$ is empty then
9:         $\text{done}$ = $\text{true}$
10:    else if $|\text{newcomb}| == 1$ then
11:        $\text{done}$ = $\text{false}$ //sample at least 2 actions per combination
12:    else
13:        $\text{done}$ = $\text{true}$ with prob. $\frac{|\text{newcomb}|}{|\text{newcomb}|+1}$
14: until $\text{done}$
15: return $\text{newcomb}$
```

This approach exposes an exploration / exploitation trade-off. Exploration, here, refers to testing a wide range of action combinations to improve understanding of their relative merit. Exploitation, on the other hand, advocates performing backups on the combinations that have previously been shown to be the best. We manage the tradeoff by carefully maintaining the distribution over combinations. First, we only memoize best combinations per state; these are always backed-up

1. A similar action sampling approach was also used in the context of space shuttle scheduling to reduce the number of actions considered during value function computation (Zhang & Dietterich, 1995).

in a Bellman update. Other combinations are constructed by an incremental probabilistic process, which builds a combination by first randomly choosing an initial action (weighted by its individual $Q_{\|}$-value), then deciding whether to add a non-mutex action or stop growing the combination. There are many implementations possible for this high level idea. We tried several of those and found the results to be very similar in all of them. Algorithm 1 describes the implementation used in our experiments. The algorithm takes a state and a total number of combinations m as an input and returns the best combination obtained so far. It also memoizes all the best combinations for the state in $\mathrm{memoizedlist}$. Function 2 is a helper function that returns the i^{th} combination that is either one of the best combinations memoized in the previous iteration or a new sampled combination. Also notice line 10 in Function 2. It forces the sampled combinations to be at least size 2, since all individual actions have already been backed up (line 3 of Algo 1).

3.5.1 Termination and Optimality

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 algorithm may assign $J_{\|_n}(s)$ a cost that is greater than the optimal $J^*_{\|}(s)$ — *i.e.*, the $J_{\|_n}(s)$ values are no longer admissible. If a better combination is chosen in a subsequent iteration, $J_{\|_{n+1}}(s)$ might be set a lower value than $J_{\|_n}(s)$, thus sampled RTDP is not *monotonic*. This is unfortunate, since admissibility and monotonicity are important properties required for termination[2] 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 costs that are extremely close to the optimal.

3.5.2 Improving Solution Quality

We have investigated several heuristics in order to improve the quality of the solutions found by sampled RTDP. Our heuristics compensate for the errors due to partial search and lack of admissibility.

- **Heuristic 1:** Whenever sampled RTDP asserts convergence of a state, do not immediately label it as converged (which would preclude further exploration (Bonet & Geffner, 2003)); instead first run a complete backup phase, using all the admissible combinations, to rule out any easy-to-detect inconsistencies.

- **Heuristic 2:** Run sampled RTDP to completion, and use the cost function it produces, $J^s()$, as the initial heuristic estimate, $J_0()$, for a subsequent run of pruned RTDP. Usually, such a heuristic, though inadmissible, is highly informative. Hence, pruned RTDP terminates quite quickly.

- **Heuristic 3:** Run sampled RTDP before pruned RTDP, as in Heuristic 2, except instead of using the $J^s()$ cost function directly as an initial estimate, scale linearly downward — *i.e.*, use $J_0() := cJ^s()$ for some constant $c \in (0, 1)$. While there are no guarantees we hope that this lies on the admissible side of the optimal. In our experience this is often the case for $c = 0.9$, and the run of pruned RTDP yields the optimal policy very quickly.

2. To ensure termination we implemented the policy: *if number of trials exceeds a threshold, force monotonicity on the cost function.* This will achieve termination but will reduce quality of solution.

Experiments showed that Heuristic 1 returns a cost function that is close to optimal. Adding Heuristic 2 improves this value moderately, and a combination of Heuristics 1 and 3 returns the optimal solution in our experiments.

3.6 Experiments: Concurrent MDP

Concurrent MDP is a fundamental formulation, modeling concurrent actions in a general planning domain. We first compare the various techniques to solve CoMDPs, *viz.*, pruned and sampled RTDP. In following sections we use these techniques to model problems with durative actions.

We tested our algorithms on problems in three domains. The first domain was a probabilistic variant of NASA Rover domain from the 2002 AIPS Planning Competition (Long & Fox, 2003), in which there are multiple objects to be photographed and various rocks to be tested with resulting data 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. The cost function includes both resource and time components, so executing multiple actions in parallel is cheaper than executing them sequentially. We generated problems with 20-30 state variables having up to 81,000 reachable states and the average number of applicable combinations per state, $Avg(Ap(s))$, which measures the amount of concurrency in a problem, is up to 2735.

We also tested on a probabilistic version of a machineshop domain with multiple subtasks (*e.g.*, roll, 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. We tested on problems with 26-28 state variables and around 32,000 reachable states. $Avg(Ap(s))$ ranged between 170 and 2640 on the various problems.

Finally, we tested on an artificial domain similar to the one shown in Figure 1 but much more complex. 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 had 19 state variables and about 32,000 reachable states, with $Avg(Ap(s))$ between 1024 and 12287.

We used Labeled RTDP, as implemented in GPT (Bonet & Geffner, 2005), as the base MDP solver. It is implemented in C++. We implemented[3] various algorithms, unpruned RTDP (U-RTDP), pruned RTDP using only combo skipping (P_s-RTDP), pruned RTDP using both combo skipping and combo elimination (P_{se}-RTDP), sampled RTDP using Heuristic 1 (S-RTDP) and sampled RTDP using both Heuristics 1 and 3, with value functions scaled with 0.9 (S_3-RTDP). We tested all of these algorithms on a number of problem instantiations from our three domains, generated by varying the number of objects, degrees of parallelism, and distances to goal. The experiments were performed on a 2.8 GHz Pentium processor with a 2 GB RAM.

We observe (Figure 2(a,b)) that pruning significantly speeds the algorithm. But the comparison of P_{se}-RTDP with S-RTDP and S_3-RTDP (Figure 3(a,b)) shows that sampling has a dramatic speedup with respect to the pruned versions. In fact, pure sampling, S-RTDP, converges extremely quickly, and S_3-RTDP is slightly slower. However, S_3-RTDP is still much faster than P_{se}-RTDP. The comparison of qualities of solutions produced by S-RTDP and S_3-RTDP *w.r.t.* optimal is shown in Table 1. We observe that solutions produced by S-RTDP are always nearly optimal. Since the

3. The code may be downloaded at *http://www.cs.washington.edu/ai/comdp/comdp.tgz*

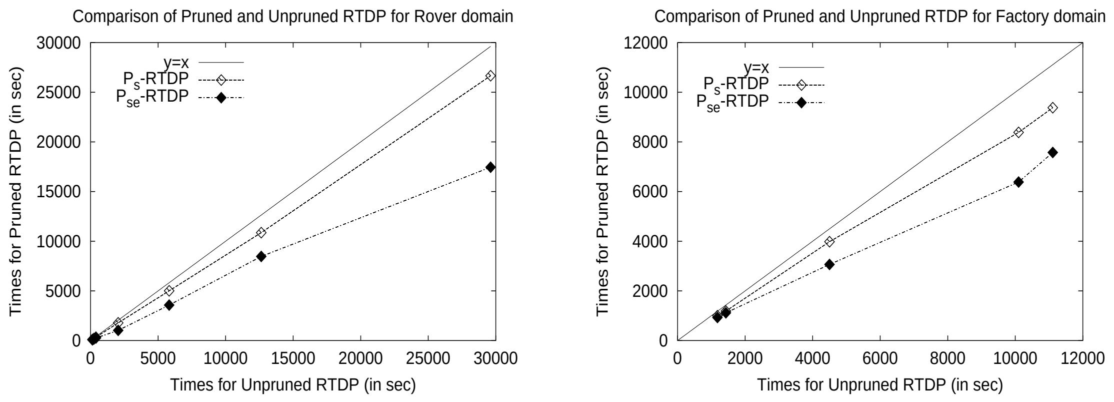


Figure 2: (a,b): Pruned vs. Unpruned RTDP for Rover and MachineShop domains respectively. Pruning non-optimal combinations achieves significant speedups on larger problems.

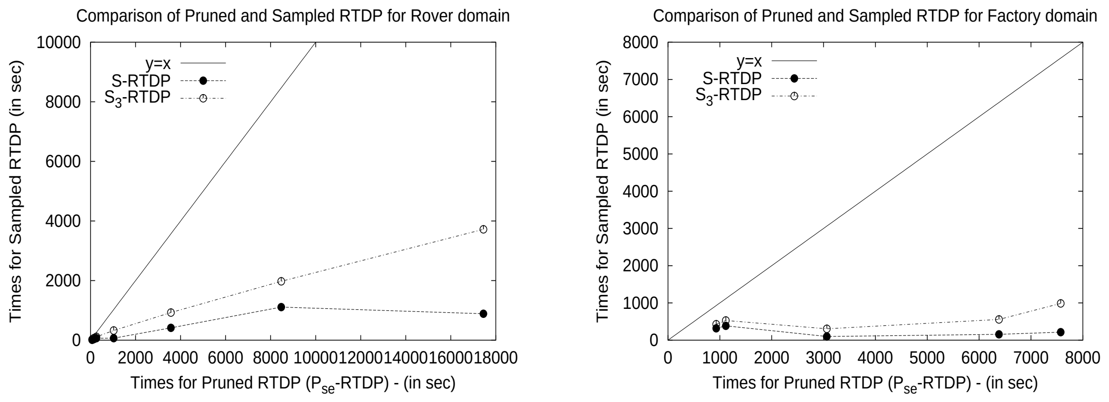
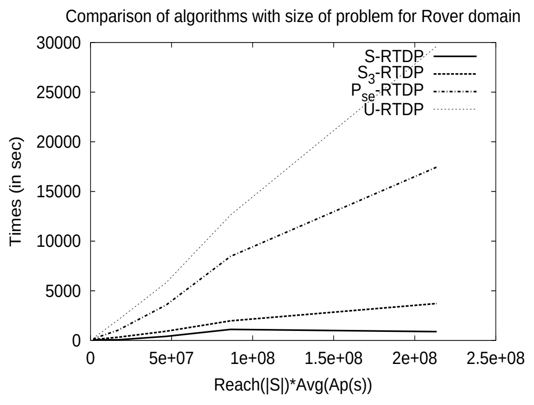


Figure 3: (a,b): Sampled vs Pruned RTDP for Rover and MachineShop domains respectively. Random sampling of action combinations yields dramatic improvements in running times.



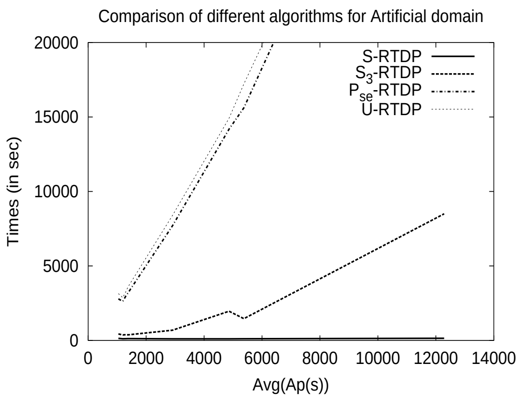
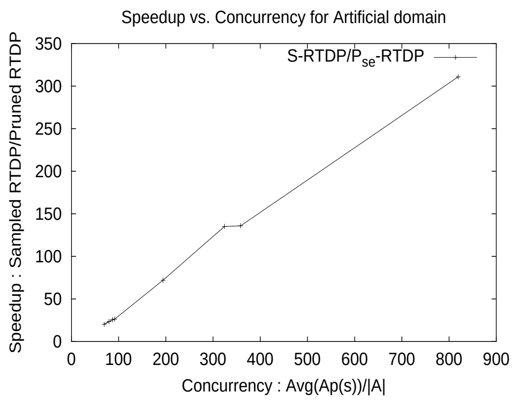


Figure 4: (a,b): Comparison of different algorithms with size of the problems for Rover and Artificial domains. As the problem size increases, the gap between sampled and pruned approaches widens considerably.



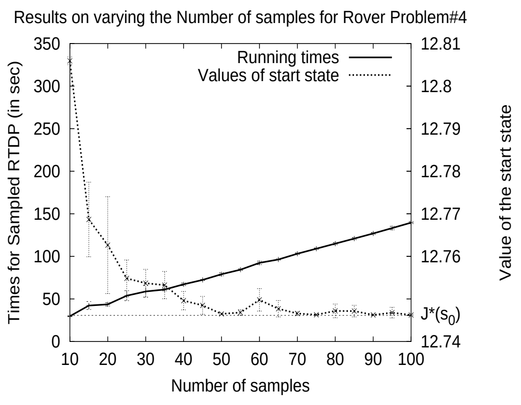


Figure 5: (a): Relative Speed vs. Concurrency for Artificial domain. (b) : Variation of quality of solution and efficiency of algorithm (with 95% confidence intervals) with the number of samples in Sampled RTDP for one particular problem from the Rover domain. As number of samples increase, the quality of solution approaches optimal and time still remains better than P_{se}-RTDP (which takes 259 sec. for this problem).

Problem	$J(s_0)$ (S-RTDP)	$J^*(s_0)$ (Optimal)	Error
Rover1	10.7538	10.7535	<0.01%
Rover2	10.7535	10.7535	0
Rover3	11.0016	11.0016	0
Rover4	12.7490	12.7461	0.02%
Rover5	7.3163	7.3163	0
Rover6	10.5063	10.5063	0
Rover7	12.9343	12.9246	0.08%
Artificial1	4.5137	4.5137	0
Artificial2	6.3847	6.3847	0
Artificial3	6.5583	6.5583	0
MachineShop1	15.0859	15.0338	0.35%
MachineShop2	14.1414	14.0329	0.77%
MachineShop3	16.3771	16.3412	0.22%
MachineShop4	15.8588	15.8588	0
MachineShop5	9.0314	8.9844	0.56%

Table 1: Quality of solutions produced by Sampled RTDP

error of S-RTDP is small, scaling it by 0.9 makes it an admissible initial cost function for the pruned RTDP; indeed, in all experiments, S_3-RTDP produced the optimal solution.

Figure 4(a,b) demonstrates how running times vary with problem size. We use the product of the number of reachable states and the average number of applicable action combinations per state as an estimate of the size of the problem (the number of reachable states in all artificial domains is the same, hence the x-axis for Figure 4(b) is $Avg(Ap(s))$). From these figures, we verify that the number of applicable combinations plays a major role in the running times of the concurrent MDP algorithms. In Figure 5(a), we fix all factors and vary the degree of parallelism. We observe that the speedups obtained by S-RTDP increase as concurrency increases. This is a very encouraging result, and we can expect S-RTDP to perform well on large problems inolving high concurrency, even if the other approaches fail.

In Figure 5(b), we present another experiment in which we vary the number of action combinations sampled in each backup. While solution quality is inferior when sampling only a few combinations, it quickly approaches the optimal on increasing the number of samples. In all other experiments we sample 40 combinations per state.

4. Challenges for Temporal Planning

While the CoMDP model is powerful enough to model concurrency in actions, it still assumes each action to be instantaneous. We now incorporate actual action durations in the modeling the problem. This is essential to increase the scope of current models to real world domains.

Before we present our model and the algorithms we discuss several new theoretical challenges imposed by explicit action durations. Note that the results in this section apply to a wide range of planning problems:

- regardless of whether durations are uncertain or fixed
- regardless of whether effects are stochastic or deterministic.

Actions of uncertain duration are modeled by associating a distribution (possibly conditioned on the outcome of stochastic effects) over execution times. We focus on problems whose objective is to achieve a goal state while minimizing total expected time (*make-span*), but our results extend to cost functions that combine make-span and resource usage. This raises the question of *when* a goal counts as achieved. We require that:

Assumption 1 *All executing actions terminate before the goal is considered achieved.*

Assumption 2 *An action, once started, cannot be terminated prematurely.*

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

Definition 1 *Any time point when a new action is allowed to start execution is called a* decision epoch*. A time point is a* pivot *if it is either 0 or a time when a new effect might occur (e.g., the end of an action's execution) or a new precondition may be needed or an existing precondition may no longer be needed. A* happening *is either 0 or a time when an effect actually occurs or a new precondition is definitely needed or an existing precondition is no longer needed.*

Intuitively, a happening is a point where a change in the world state or action constraints actually "happens" (*e.g.*, by a new effect or a new precondition). When execution crosses a pivot (a possible happening), information is gained by the agent's execution system (*e.g.*, did or didn't the effect occur) which may "change the direction" of future action choices. Clearly, if action durations are deterministic, then the set of pivots is the same as the set of happenings.

Example: Consider an action a whose durations follow a uniform integer duration between 1 and 10. If it is started at time 0 then all timepoints 0, 1, 2,. . ., 10 are pivots. If in a certain execution it finishes at time 4 then 4 (and 0) is a happening (for this execution). □

Definition 2 *An action is a PDDL*$_{2.1}$ *action (Fox & Long, 2003) if the following hold:*

- *The effects are realized instantaneously either* (at start) *or* (at end)*, i.e., at the beginning or the at the completion of the action (respectively).*
- *The preconditions may need to hold instaneously before the start* (at start)*, before the end* (at end) *or over the complete execution of the action* (over all).

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

Figure 6: A domain to illustrate that an expressive action model may require arbitrary decision epochs for a solution. In this example, b needs to start at 3 units after a's execution to reach $Goal$.

Theorem 5 *For a PDDL$_{2.1}$ domain restricting decision epochs to pivots causes incompleteness (i.e., a problem may be incorrectly deemed unsolvable).*

Proof: Consider the deterministic temporal planning domain in Figure 6 that uses PDDL$_{2.1}$ notation (Fox & Long, 2003). If the initial state is P=true and Q=false, then the only way to reach $Goal$ is to start a at time t (*e.g.*, 0), and b at some timepoint in the open interval $(t + 2, t + 4)$. Clearly, no new information is gained at any of the time points in this interval and none of them is a pivot. Still, they are required for solving the problem. □

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

Temporal planners may be classified as having one of two architectures: constraint-posting approaches in which the times of action execution are gradually constrained during planning (*e.g.*, Zeno and LPG (Penberthy & Weld, 1994; Gerevini & Serina, 2002)) and extended state-space methods (*e.g.*, TP4 and SAPA (Haslum & Geffner, 2001; Do & Kambhampati, 2001)). Theorem 5 holds for both architectures but has strong computational implications for state-space planners because limiting attention to a subset of decision epochs can speed these planners. (The theorem also shows that planners like SAPA and Prottle (Little, Aberdeen, & Thiebaux, 2005) are incomplete.) Fortunately, an assumption restricts the set of decision epochs considerably.

Definition 3 *An action is a TGP-style action*[4] *if all of the following hold:*

- *The effects 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 its transition function is conditioned) must not be changed during an action's execution, except by an effect of the action itself.*

Thus, two TGP-style actions may not execute concurrently if they clobber each other's preconditions or effects. For the case of TGP-style actions the set of happenings is nothing but the set of time points when some action terminates. TGP pivots are the set of points when an action might terminate. (Of course both these sets additionally include zero).

Theorem 6 *If all actions are TGP-style, then the set of decision epochs may be restricted to pivots without sacrificing completeness or optimality.*

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

In the case of deterministic durations, the set of happenings is same as the set of pivots; hence the following corollary holds:

4. While the original TGP (Smith & Weld, 1999) considered only deterministic actions of fixed duration, we use the phrase "TGP-style" in a more general way, without these restrictions.

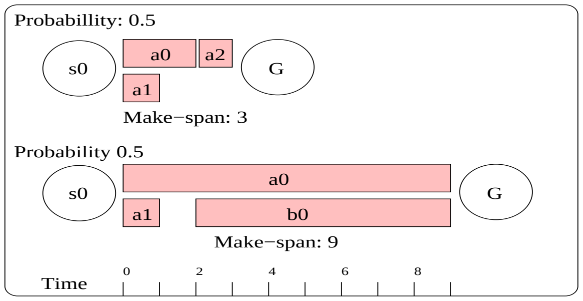


Figure 7: Pivot decision epochs are necessary for optimal planning in face of nonmonotonic continuation. In this domain, $Goal$ can be achieved by $\langle\{a_0, a_1\}; a_2\rangle$ or $\langle b_0\rangle$; a_0 has duration 2 or 9; and b_0 is mutex with a_1. The optimal policy starts a_0 and then, if a_0 does *not* finish at time 2, it starts b_0 (otherwise it starts a_1).

Corollary 7 *If all actions are TGP-style with deterministic durations, then the set of decision epochs may be restricted to happenings without sacrificing completeness or optimality.*

When planning with uncertain durations there may be a huge number of pivots; it is useful to further constrain the range of decision epochs.

Definition 4 *An action has* independent duration *if there is no correlation between its probabilistic effects and its duration.*

Definition 5 *An action has* monotonic continuation *if the expected time until action termination is nonincreasing during execution.*

Actions without probabilistic effects, by nature, have independent duration. Actions with monotonic continuations are common, *e.g.* those with uniform, exponential, Gaussian, and many other duration distributions. However, actions with bimodal or multi-modal distributions don't have monotonic continuations. For example consider an action with uniform distribution over [1,3]. If the action doesn't terminate until 2, then the expected time until completion is calculated as 2, 1.5, and 1 for times 0, 1, and 2 respectively, which is monotonically decreasing. For an example of non-monotonic continuation see Figure 18.

Conjecture 8 *If all actions are TGP-style, have independent duration and monotonic continuation, then the set of decision epochs may be restricted to happenings without sacrificing completeness or optimality.*

If an action's continuation is nonmonotonic then failure to terminate can increase the expected time remaining and cause another sub-plan to be preferred (see Figure 7). Similarly, if an action's duration isn't independent then failure to terminate changes the probability of its eventual effects and this may prompt new actions to be started.

By exploiting these theorems and conjecture we may significantly speed planning since we are able to limit the number of decision epochs needed for decision-making. We use this theoretical understanding in our models. First, for simplicity, we consider only the case of TGP-style actions with deterministic durations. In Section 6, we relax this restriction by allowing stochastic durations, both unimodal as well as multimodal.

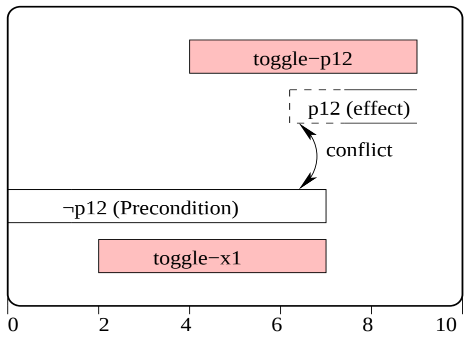


Figure 8: A sample execution demonstrating conflict due to interfering preconditions and effects. (The actions are shaded to disambiguate them with preconditions and effects)

5. Temporal Planning with Deterministic Durations

We use the abbreviation *CPTP* (short for *Concurrent Probabilistic Temporal Planning*) to refer to the probabilistic planning problem with durative actions. A CPTP problem has an input model similar to that of CoMDPs except that action costs, $\mathcal{C}(s, a, s')$, are replaced by their *deterministic* durations, $\Delta(a)$, *i.e.*, the input is of the form $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}r, \Delta, \mathcal{G}, s_0 \rangle$. We study the objective of minimizing the expected time (*make-span*) of reaching a goal. For the rest of the paper we make the following assumptions:

Assumption 3 *All action durations are integer-valued.*

This assumption has a negligible effect on expressiveness because one can convert a problem with rational durations into one that satisfies Assumption 3 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 reasons discussed in the previous section we adopt the TGP temporal action model of Smith and Weld (1999), rather than the more complex PDDL$_{2.1}$ (Fox & Long, 2003). Specifically:

Assumption 4 *All actions follow the TGP model.*

These restrictions are consistent with our previous definition of concurrency. Specifically, the mutex definitions (of CoMDPs over probabilistic STRIPS) hold and are required under these assumptions. As an illustration, consider Figure 8. It describes a situation in which two actions with interfering preconditions and effects can not be executed concurrently. To see why not, suppose initially p_{12} was false and two actions toggle-x_1 and toggle-p_{12} were started at time 2 and 4, respectively. As $\neg p_{12}$ is a precondition of toggle-x_1, whose duration is 5, it needs to remain false until time 7. But toggle-p_{12} may produce its effects anytime between 4 and 9, which may conflict with the preconditions of the other executing action. Hence, we forbid the concurrent execution of toggle-x_1 and toggle-p_{12} to ensure a completely predictable outcome distribution.

Because of this definition of concurrency, the dynamics of our model remains consistent with Equation 5. Thus the techniques developed for CoMDPs derived from probabilistic STRIPS actions may be used.

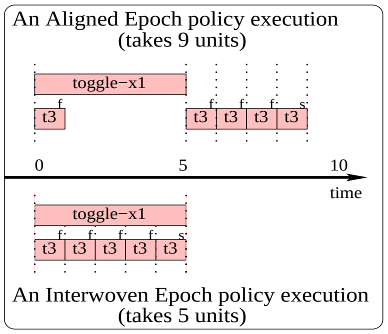


Figure 9: Comparison of times taken in a sample execution of an interwoven-epoch policy and an aligned-epoch policy. In both trajectories the toggle-x_3 (t3) action fails four times before succeeding. Because the aligned policy must wait for all actions to complete before starting any more, it takes more time than the interwoven policy, which can start more actions in the middle.

5.1 Formulation as a CoMDP

We can model a CPTP problem as a CoMDP, and thus as an MDP, in more than one way. We list the two prominent formulations below. Our first formulation, *aligned epoch* CoMDP models the problem approximately but solves it quickly. The second formulation, *interleaved epochs* models the problem exactly but results in a larger state space and hence takes longer to solve using existing techniques. In subsequent subsections we explore ways to speed up policy construction for the interleaved epoch formulation.

5.1.1 Aligned Epoch Search Space

A simple way to formulate CPTP is to model it as a standard CoMDP over probabilistic STRIPS, in which action costs are set to their durations and the cost of a combination is the maximum duration of the constituent actions (as in Equation 6). This formulation introduces a substantial approximation to the CPTP problem. While this is true for deterministic domains too, we illustrate this using our example involving stochastic effects. Figure 9 compares the trajectories in which the toggle-x_3 (t3) actions fails for four consecutive times before succeeding. In the figure, "f" and "s" denote failure and success of uncertain actions, respectively. The vertical dashed lines represent the time-points when an action is started.

Consider the actual executions of the resulting policies. In the aligned-epoch case (Figure 9 top), 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, Figure 9 bottom shows that 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 actual state space for CPTP decision making is substantially different from that of the simple aligned-epoch model.

Note that due to Corollary 7 it is sufficient to consider a new decision epoch only at a happening, *i.e.*, a time-point when one or more actions complete. Thus, using Assumption 3 we infer that these decision epochs will be discrete (integer). Of course, not *all* optimal policies will have this property.

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

Action	$\Delta(a)$	Precondition	Effect	Probability
toggle-x_1	5	$\neg p_{12}$	$x_1 \leftarrow \neg x_1$	1
toggle-x_2	5	p_{12}	$x_2 \leftarrow \neg x_2$	1
toggle-x_3	1	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}	5	true	$p_{12} \leftarrow \neg p_{12}$	1

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

Figure 10: The domain of Example 1 extended with action durations.

But it is easy to see that there exists at least one optimal policy in which each action begins at a happening. Hence our search space reduces considerably.

5.1.2 Interwoven Epoch Search Space

We adapt the search space representation of Haslum and Geffner (2001), which is similar to that in other research (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 passed since they were started. Formally, let the new interwoven state[5] $s \in \mathcal{S}_{\underline{-}}$ be an ordered pair $\langle X, Y \rangle$ where:

- $X \in \mathcal{S}$
- $Y = \{(a, \delta) | a \in \mathcal{A}, 0 \leq \delta < \Delta(a)\}$

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 the times passed since their start "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 ignoring execution times in progress:

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

Example: Continuing our example with the domain of Figure 10, suppose state s_1 has all state variables false, and suppose the action toggle-x_1 was started 3 units ago from the current time. Such a state would be represented as $\langle X_1, Y_1 \rangle$ with X_1=(F, F, F, F, F) and Y_1=$\{$(toggle-x_1,3)$\}$ (the five state variables are listed in the order: x_1, x_2, x_3, x_4 and p_{12}). The set A_{s_1} would be $\{$toggle-$x_1\}$.

To allow the possibility of simply waiting for some action to complete execution, that is, deciding at a decision epoch not to start any additional action, we augment the set $\mathcal{A}$ with a *no-op* action, which is applicable in all states $s = \langle X, Y \rangle$ where $Y \neq \emptyset$ (*i.e.* states in which some action is still being executed). For a state s, the no-op action is mutex with all non-executing actions, *i.e.*, those in $\mathcal{A} \setminus A_s$. In other words, at any decision epoch either a no-op will be started or any combination not

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

involving no-op. We define no-op to have a variable duration[6] 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:

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

Transition Function: We also need to define the probability transition function, $\mathcal{P}r_{\underline{\text{-}}}$, 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, the next decision epoch will be the next time that an executing action terminates. Let us call this time $\delta_{next}(s, A)$. Notice that $\delta_{next}(s, A)$ depends on both executing and newly started actions. Formally,

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

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 exactly in $\delta_{next}(s, A)$ timesteps. The Y-component of the state at the decision epoch after $\delta_{next}(s, A)$ time will be

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

Let $s = \langle X, Y \rangle$ and let $s' = \langle X', Y' \rangle$. The transition function for CPTP can now be defined as:

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

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, specifically to 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 times elapsed. Also, because of TGP-style actions, the probability distribution of different state variables is modified independently. Thus the probability transition function due to CoMDP over probabilistic STRIPS 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_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_1,4)}. 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

6. A precise definition of the model will create multiple no-op$_t$ actions with different constant durations t and the no-op$_t$ applicable in an interwoven state will be the one with $t = \delta_{next}(s, A)$.

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

By redefining the start and goal states, the applicability function, and the probability transition function, we have finished modeling a CPTP problem as a CoMDP in the interwoven state space. Now we can use the techniques of CoMDPs (and MDPs as well) 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:

$$J^*_{\underline{-}}(s) = 0, \text{ if } s \in \mathcal{G}_{\underline{-}} \text{ else} \tag{11}$$

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

We will use DUR$_{\text{samp}}$ to refer to the sampled RTDP algorithm over this search space. The main bottleneck in naively inheriting algorithms like DUR$_{\text{samp}}$ 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 $|\mathcal{S}| \times (\prod_{a \in \mathcal{A}} \Delta(a))$. 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 or abstract/aggregate our state space in order to make the problem tractable. We now present several heuristics which can be used to speed the search.

5.2 Heuristics

We present both an admissible and an inadmissible heuristics that can be used as the initial cost function for DUR$_{\text{samp}}$ algorithm. The first heuristic (maximum concurrency) solves the underlying MDP and is thus quite efficient to compute. The second heuristic (average concurrency) is inadmissible, but tends to be more informed than the maximum concurrency heuristic.

5.2.1 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 $Q(X, A)$ denote the expected cost to reach the goal if initially all actions in the combination A are executed and the greedy serial policy is followed thereafter. Formally, $Q(X, A) = \sum_{X' \in \mathcal{S}} \mathcal{P}r_{||}(X'|X, A)J(X')$. 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 number of actions that can be concurrently executed in any world state in the domain. The following theorem can be used to provide an admissible heuristic for CPTP problems.

Theorem 9 *Let* $s = \langle X, Y\rangle$*,*

$$\begin{aligned} J^*_{\underline{-}}(s) &\geq \frac{J^*(X)}{c} \quad \textit{for } Y = \emptyset \\ J^*_{\underline{-}}(s) &\geq \frac{Q^*(X, A_s)}{c} \quad \textit{for } Y \neq \emptyset \end{aligned} \tag{12}$$

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 and is a one-time expense. The complete heuristic function can be evaluated by solving the MDP for all states. However, many of these states may never be visited. 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. We refer to DUR$_{\text{samp}}$ initiated with the *MC* heuristic by DUR$^{\text{MC}}_{\text{samp}}$.

5.2.2 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. We call the resulting algorithm DUR$^{\text{AC}}_{\text{samp}}$. The *AC* heuristic is not admissible, but in our experiments it is typically a more informed heuristic. Moreover, in the case where all the actions have the same duration, the *AC* heuristic equals the *MC* heuristic.

5.3 Hybridized Algorithm

We present an approximate method to solve CPTP problems. While there can be many kinds of 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 dead-ends, we explore the desirable notion of *propriety*.

Definition 6 *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 et al., 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.

5.3.1 Hybridized Planner

Our algorithm, $\mathrm{DUR_{hyb}}$, 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 *hybridized* policy.

Algorithm 3 Hybridized Algorithm $\mathrm{DUR_{hyb}}(r, k, m)$

```
1: for all $s \in \mathcal{S}_{\underline{=}}$ do
2:     initialize $J_{\underline{=}}(s)$ with an admissible heuristic
3: repeat
4:     perform $m$ RTDP trials
5:     compute hybridized policy ($\pi_{hyb}$) using interwoven-epoch policy for $k$-familiar states and aligned-
       epoch policy otherwise
6:     clean $\pi_{hyb}$ by removing all dead-ends and cycles
7:     $J^{\pi}_{\underline{=}}\langle s_0, \emptyset\rangle \leftarrow$ evaluation of $\pi_{hyb}$ from the start state
8: until $\left(\frac{J^{\pi}_{\underline{=}}(\langle s_0,\emptyset\rangle) - J_{\underline{=}}(\langle s_0,\emptyset\rangle)}{J_{\underline{=}}(\langle s_0,\emptyset\rangle)} < r\right)$
9: return hybridized policy $\pi_{hyb}$
```

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.

We have not yet said how the hybridized 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, $J_{\underline{=}}(\langle s_0, \emptyset\rangle)$, remains admissible (thus less than or equal to optimal). In contrast, the hybridized 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.

7. In our implementation cycles are detected using simulation.

Finally, evaluation of the hybridized policy is done using simulation, which we perform after a fixed number of m RTDP trials. Algorithm 3 summarizes the details of the algorithm. 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 the aligned-epoch model was run to convergence, and 2) for the rest of the states it has explicitly ensured that there are no cycles or dead-ends.

5.4 Experiments: Planning with Deterministic Durations

Continuing from Section 3.6, in this set of experiments we evaluate the various techniques for solving problems involving explicit deterministic durations. We compare the computation time and solution quality of five methods: interwoven Sampled RTDP with no heuristic ($\mathrm{DUR}_{\mathrm{samp}}$), with the maximum concurrency ($\mathrm{DUR}^{\mathrm{MC}}_{\mathrm{samp}}$), and average concurrency ($\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$) heuristics, the hybridized algorithm ($\mathrm{DUR}_{\mathrm{hyb}}$) and Sampled RTDP on the aligned-epoch model ($\mathrm{DUR}_{\mathrm{AE}}$). We test on our Rover, MachineShop and Aritificial domains. We also use our Artificial domain to see if the relative performance of the techniques varies with the amount of concurrency in the domain.

5.4.1 Experimental Setup

We modify the domains used in Section 3.6 by additionally including action durations. For NASA Rover and MachineShop domains, we generate problems with 17-26 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, $\mathcal{S}_{\underline{-}}$.

We use Artificial domain for control experiments to study the effect of 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] and implement all heuristics: maximum concurrency (H_{MC}), average concurrency (H_{AC}), 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 hybridized 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.

5.4.2 Comparison of Running Times

Figures 11(a, b) and 12(a) show the variations in the running times for the algorithms on different problems in Rover, Machineshop and Artificial domains, respectively. The first three bars represent the base Sampled RTDP without any heuristic, with H_{MC}, and with H_{AC}, respectively. The fourth bar represents the hybridized algorithm (using the H_{MC} heuristic) and the fifth 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 taken for the aligned-epoch RTDP computations in the hybridized algorithm. The error bars represent 95% confidence intervals on the running times. Note that the plots are on a log scale.

8. Note that policies returned by $\mathrm{DUR}_{\mathrm{samp}}$ are not guaranteed to be optimal. Thus all the implemented algorithms are approximate. We can replace $\mathrm{DUR}_{\mathrm{samp}}$ by pruned RTDP ($\mathrm{DUR}_{\mathrm{prun}}$) if optimality is desired.

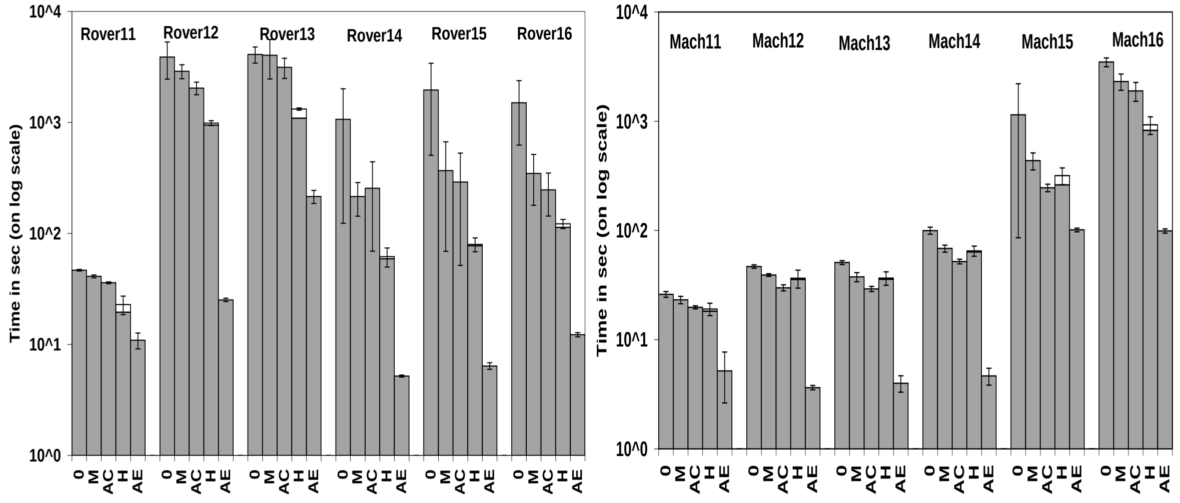


Figure 11: (a,b): Running times (on a log scale) for the Rover and Machineshop domain, respectively. For each problem the five bars represent the times taken by the algorithms: $\mathrm{DUR}_{\mathrm{samp}}$ (0), $\mathrm{DUR}^{\mathrm{MC}}_{\mathrm{samp}}$ (AE), $\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$ (AC), $\mathrm{DUR}_{\mathrm{hyb}}$ (H), and $\mathrm{DUR}_{\mathrm{AE}}$ (AE), respectively. The white bar on $\mathrm{DUR}_{\mathrm{hyb}}$ denotes the portion of time taken by aligned-epoch RTDP.

Algos	Speedup compared with $\mathrm{DUR}_{\mathrm{samp}}$			
	Rover	Machineshop	Artificial	Average
$\mathrm{DUR}^{\mathrm{MC}}_{\mathrm{samp}}$	3.016764	1.545418	1.071645	1.877942
$\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$	3.585993	2.173809	1.950643	2.570148
$\mathrm{DUR}_{\mathrm{hyb}}$	10.53418	2.154863	16.53159	9.74021
$\mathrm{DUR}_{\mathrm{AE}}$	135.2841	16.42708	241.8623	131.1911

Table 2: The ratio of the time taken by $\mathcal{S}_{\underline{-}}$ S-RTDP with no heuristics to that of each algorithm. Our heuristics produce 2-3 times speedups. The hybridized algo produces about a 10x speedup. Aligned epoch search produces 100x speedup, but sacrifices solution quality.

We notice that $\mathrm{DUR}_{\mathrm{AE}}$ solves the problems extremely quickly; this is natural since the aligned-epoch space is much smaller. Use of both H_{MC} and H_{AC} always speeds search in the $\mathcal{S}_{\underline{-}}$ model. Comparing the heuristics amongst themselves, we find that average concurrency heuristic mostly performs faster than maximum concurrency — 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 Rover domain, the hybridized 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, hybridized 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 $\mathrm{DUR}_{\mathrm{hyb}}$ and $\mathrm{DUR}_{\mathrm{AE}}$ converge to a solution.

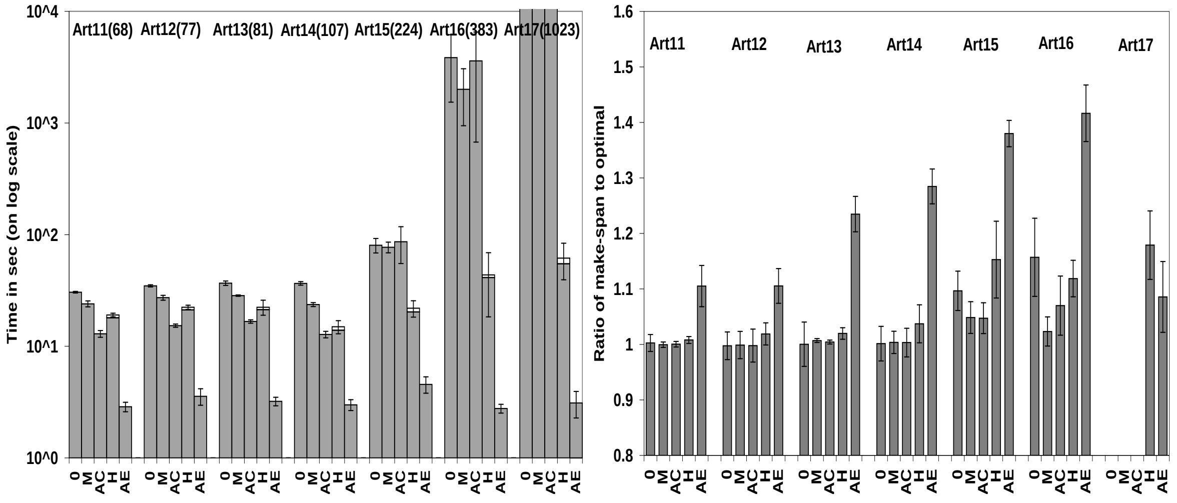


Figure 12: (a,b): Comparison of the different algorithms (running times and solution quality respectively) for the Artificial domain. As degree of parallelism increases the problems become harder; the largest problem is solved only by DUR_{hyb} and DUR_{AE}.

Table 2 shows the speedups obtained by various algorithms compared to the basic DUR_{samp}. In the Rover and Artificial domains the speedups obtained by DUR_{hyb} and DUR_{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.

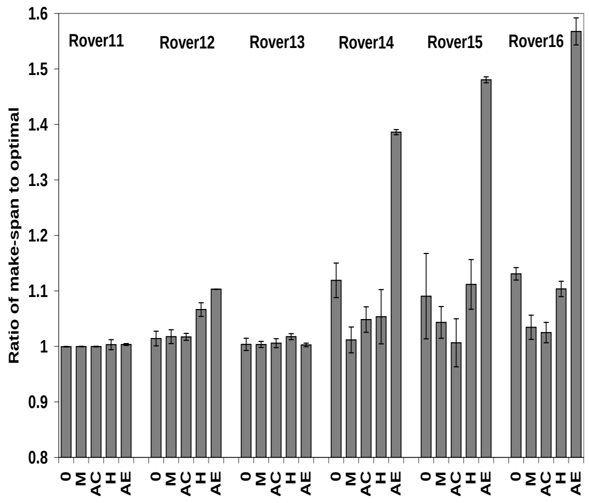
5.4.3 Comparison of Solution Quality

Figures 13(a, b) and 12(b) show the quality of the policies obtained by the same five 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]. Table 3 presents solution qualities for each method, averaged over all problems in a domain. We note that the aligned-epoch 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 hybridized algorithm extorts only a small sacrifice in quality in exchange for its speed.

5.4.4 Variation with Concurrency

Figure 12(a) 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, $Avg_{s\in\underline{\mathcal{S}}_{-}}|Ap(s)|$, and range from 68 to 1023 concurrent actions. Note that for the difficult problems with a lot of parallelism, DUR_{samp} slows dramatically, regardless of heuristic. In contrast, the DUR_{hyb} is still able to quickly produce a policy, and at almost no loss in quality (Figure 12(b)).

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



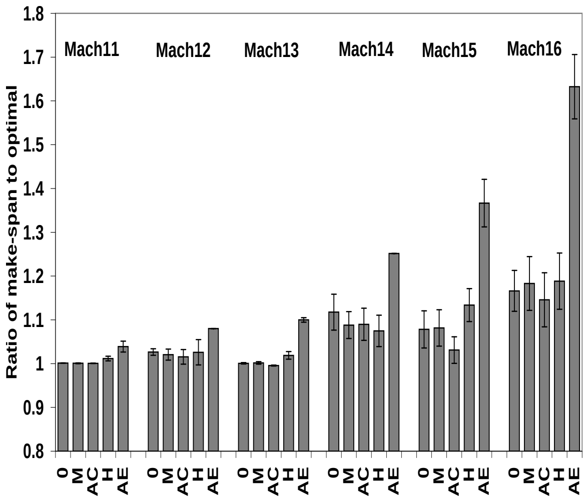


Figure 13: (a,b): Comparison of make-spans of the solution found with the optimal(plotted as 1 on the y-axes) for Rover and Machineshop domains, respectively. All algorithms except $\mathrm{DUR}_{\mathrm{AE}}$ produce solutions quite close to the optimal.

Algos	Average Quality			
	Rover	Machineshop	Artificial	Average
$\mathrm{DUR}_{\mathrm{samp}}$	1.059625	1.065078	1.042561	1.055704
$\mathrm{DUR}^{\mathrm{MC}}_{\mathrm{samp}}$	1.018405	1.062564	1.013465	1.031478
$\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$	1.017141	1.046391	1.020523	1.028019
$\mathrm{DUR}_{\mathrm{hyb}}$	1.059349	1.075534	1.059201	1.064691
$\mathrm{DUR}_{\mathrm{AE}}$	1.257205	1.244862	1.254407	1.252158

Table 3: Overall solution quality produced by all algorithms. Note that all algorithms except $\mathrm{DUR}_{\mathrm{AE}}$ produce policies whose quality is quite close to optimal. On average $\mathrm{DUR}_{\mathrm{AE}}$ produces make-spans that are about 125% of the optimal.

6. Optimal Planning with Uncertain Durations

We now extend the techniques of previous section for the case when action durations are not deterministic. As before, we consider TGP-style actions and a discrete temporal model. We assume independent durations, and monotonic continuations, but Section 6.3 relaxes the latter, extending our algorithms to handle multimodal duration distributions. As before we aim to minimize the expected time required to reach a goal.

6.1 Formulating as a CoMDP

We now formulate our planning problem as a CoMDP similar to Section 5.1. While some of the parameters of the CoMDP can be used directly from our work on deterministic durations, we need to recompute the transition function.

State Space: Both the aligned epoch state space as well as the interwoven epoch space, as defined in Section 5.1 are adequate to model this planning problem. To determine the size of the interwoven space, we replace the duration of an action by its *max* duration. Let $\Delta_M(a)$ denote the maximum time within which action a will complete. The overall interwoven-epoch search space is $\mathcal{S}_{\underline{-}} = \mathcal{S} \times \bigotimes_{a\in\mathcal{A}} \left(\{a\} \times Z_{\Delta_M(a)}\right)$, where $Z_{\Delta_M(a)}$ represents the set $\{0, 1, \ldots, \Delta_M(a) - 1\}$ and $\bigotimes$ denotes the Cartesian product over multiple sets.

Action Space: At any state we may apply a combination of actions with the applicability function reflecting the fact that the combination of actions is safe *w.r.t* itself (and *w.r.t.* already executing actions in case of interwoven space) as in the previous sections. While the previous state space and action space work well for our problem, the transition function definition needs to change, since we now need to take into account the uncertainty in durations.

Transition Function: Uncertain durations require significant changes to the probability transition function ($\mathcal{P}r_{\underline{-}}$) for the interwoven space from the definitions of Section 5.1.2. Since our assumptions justify Conjecture 8, we need only consider happenings when choosing decision epochs.

Algorithm 4 ComputeTransitionFunc(s=$\langle X, Y\rangle$,A)

```
1: $\mathcal{Y} \leftarrow Y \cup \{(a, 0)\}\ \ \forall a \in A$
2: $mintime \leftarrow \min_{(a,\delta)\in\mathcal{Y}}$ minimum remaining time for $a$
3: $maxtime \leftarrow \min_{(a,\delta)\in\mathcal{Y}}$ maximum remaining time for $a$
4: for all integer $t \in [mintime, maxtime]$ do
5:    $A_t \leftarrow$ set of actions that could possibly terminate after $t$
6:    for all non-empty subsets $Asub_t \subseteq A_t$ do
7:       $p_c \leftarrow$ (prob. that exactly $Asub_t$ terminates after $t$ (see Equation 13).
8:       $\mathcal{W} \leftarrow \{(X_t, p_w) \mid X_t$ is a world state; $p_w$ is the probability that $Asub_t$ terminates yielding $X_t\}$.
9:       for all $(X_t, p_w) \in \mathcal{W}$ do
10:          $Y_t \leftarrow \{(a, \delta + t) \mid (a, \delta) \in \mathcal{Y}, a \notin Asub_t\}$
11:          insert $(\langle X_t, Y_t\rangle, p_w \times p_c)$ in $output$
12: return $output$
```

The computation of transition function is described in Algorithm 4. Although the next decision epoch is determined by a happening, we still need to *consider* all pivots for the next state calculations as all these are potential happenings. *mintime* is the minimum time when an executing action could terminate, *maxtime* is the minimum time by which it is guaranteed that at least one action will terminate. For all times between *mintime* and *maxtime* we compute the possible combinations that could terminate then and the resulting next interwoven state. The probability, p_c, (line 7) may be computed using the following formula:

$$\begin{aligned} p_c \;=\; & \prod_{(a,\delta_a)\in\mathcal{Y}, a\in Asub_t} (\text{prob. } a \text{ terminates at } \delta_a + t | a \text{ hasn't terminated till } \delta_a) \times \\ & \prod_{(b,\delta_b)\in\mathcal{Y}, b\notin Asub_t} (\text{prob. } b \text{ doesn't terminate at } \delta_b + t | b \text{ hasn't terminated till } \delta_b) \qquad (13) \end{aligned}$$

Considering all pivots makes the algorithm computationally intensive because there may be many pivots and many action combinations could end at each one, and with many outcomes each. In our implementation, we cache the transition function so that we do not have to recompute the information for any state.

Start and Goal States: The start state and goal set that we developed for the deterministic durations work unchanged when the durations are stochastic. So, the start state is $\langle s_0, \emptyset\rangle$ and the goal set is $\mathcal{G}_{\underline{-}} = \{\langle X, \emptyset\rangle | X \in \mathcal{G}\}$.

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

Bellman Equations for Interwoven-Epoch Space: Define $\delta_{el}(s, A, s')$ as the time elapsed between two interwoven states s and s' when combination A is executed in s. The set of equations for the solution of our problem can be written as:

$$
\begin{aligned}
&J^*_{\underline{-}}(s) = 0, \text{ if } s \in \mathcal{G}_{\underline{-}} \text{ else} \\
&J^*_{\underline{-}}(s) = \min_{A \in Ap_{\underline{-}}(s)} \sum_{s' \in \mathcal{S}_{\underline{-}}} \mathcal{P}r_{\underline{-}}(s'|s, A) \left\{ \delta_{el}(s, A, s') + J^*_{\underline{-}}(s') \right\}
\end{aligned}
\tag{14}
$$

Compare these equations with Equation 11. There is one difference besides the new transition function — the time elapsed is within the summation sign. This is because time elapsed depends also on the next interwoven state.

Having modeled this problem as a CoMDP we again use our algorithms of Section 5. We use ΔDUR to denote the family of algorithms for the CPTP problems involving stochastic durations. The main bottleneck in solving these problem, besides the size of the interwoven state space, is the high branching factor.

6.1.1 Policy Construction: RTDP & Hybridized Planning

Since we have modeled our problem as a CoMDP in the new interwoven space, we may use pruned RTDP ($\Delta\text{DUR}_{\text{prun}}$) and sampled RTDP ($\Delta\text{DUR}_{\text{samp}}$) for policy construction. Since the cost function in our problem (δ_{el}) depends also on the current and the next state, combo-skipping does not apply for this problem. Thus $\Delta\text{DUR}_{\text{prun}}$ refers to RTDP with only combo-elimination.

Furthermore, only small adaptations are necessary to incrementally compute the (admissible) *maximum concurrency* (MC) and (more informed, but inadmissible) *average concurrency* (AC) heuristics. For example, for the serial MDP (in the RHS of Equation 12) we now need to compute the average duration of an action and use that as the action's cost.

Likewise, we can further speed planning by hybridizing ($\Delta\text{DUR}_{\text{hyb}}$) RTDP algorithms for interwoven and aligned-epoch CoMDPs to produce a near-optimal policy in significantly less time. The dynamics of aligned epoch space is same as that in Section 5 with one exception. The cost of a combination, in the case of deterministic durations, was simply the $\max$ duration of the constituent actions. The novel twist stems from the fact that uncertain durations require computation of the *cost* of an action combination as the expected time that the last action in the combination will terminate. For example, suppose two actions, both with uniform duration distributions over [1,3], are started concurrently. The probabilities that both actions will have finished by times 1, 2 and 3 (and no earlier) are 1/9, 3/9, and 5/9 respectively. Thus the expected duration of completion of the combination (let us call it Δ_{AE}) is $1\times$1/9 + $2\times$3/9 + $3\times$5/9 = 2.44.

6.2 Expected-Duration Planner

When modeled as a CoMDP in the full-blown interwoven space, stochastic durations cause an exlposive growth in the branching factor. In general, if n actions are started each with m possible durations and each having r probabilistic effects, then there are $(m-1)[(r+1)^n - r^n - 1] + r^n$ potential successors. This number may be computed as follows: for each duration between 1 and $m-1$ any subset of actions could complete and each action could result in r outcomes. Hence, total number of successors per duration is $\sum_{i\in[1..n]} {}^nC_i r^i = (r+1)^n - r^n - 1$. Moreover, if none of the actions finish until time $m-1$ then at the last step all actions terminate leading into r^n outcomes. So, total number of successors is $(m-1)[(r+1)^n - r^n - 1] + r^n$. Thus, the branching factor is multiplicative in the duration uncertainty and exponential in the concurrency.

To manage this extravagant computation we must curb the branching factor. One method is to ignore duration distributions. We can assign each action a *constant* duration equal to the mean of its distribution, then apply a deterministic-duration planner such as DUR$_{\text{samp}}$. However, when executing the deterministic-duration policy in a setting where durations are actually stochastic, an action will likely terminate at a time *different* than its mean, expected duration. The ΔDUR$_{\text{exp}}$ planner addresses this problem by augmenting the deterministic-duration policy created to account for these unexpected outcomes.

6.2.1 Online Version

The procedure is easiest to understand in its *online* version (Algorithm 5): wait until the unexpected happens, pause execution, and re-plan. If the original estimate of an action's duration is implausible, we compute a revised deterministic estimate in terms of $\mathcal{E}_a(min)$ — the expected value of a's duration given that it has not terminated by time min. Thus, $\mathcal{E}_a(0)$ will compute the expected duration of a.

Algorithm 5 Online ΔDUR$_{\text{exp}}$

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

Example: Let the duration of an action a follow a uniform distribution between 1 and 15. The expected value that gets assigned in the first run of the algorithm ($\lceil \mathcal{E}_a(0) \rceil$) is 8. While running the algorithm, suppose the action didn't terminate by 8 and we reach a state where a has been running for, say, 9 time units. In that case, a revised expected duration for a would be ($\lceil \mathcal{E}_a(8) \rceil$) = 12. Similarly, if it doesn't terminate by 12 either then the next expected duration would be 14, and finally 15. In other words for all states where a has been executing for times 0 to 8, it is expected to terminate at 8. For all times between 8 and 12 the expected completion is at 12, for 12 to 14 it is 14 and if it doesn't terminate at 14 then it is 15. □

6.2.2 Offline Version

This algorithm also has an *offline version* in which re-planning for all contingencies is done ahead of time and for fairness we used this version in the experiments. Although the offline algorithm plans for all possible action durations, it is still much faster than the other algorithms. The reason is that each of the planning problems solved is now significantly smaller (less branching factor, smaller reachable state space), and all the previous computation can be succinctly stored in the form of the $\langle$interwoven state, value$\rangle$ pairs and thus reused. Algorithm 6 describes this offline planner and the subsequent example illustrates the savings.

Algorithm 6 Offline ΔDUR$_{\text{exp}}$

```
1:  build a deterministic-duration policy from the start state $s_0$; get current $J_{\underline{-}}$ and $\pi_{\underline{-}}$ values
2:  insert $s_0$ in the queue open
3:  **repeat**
4:      state = open.pop()
5:      **for all** currstate s.t. $\mathcal{P}r_{\underline{-}}(\text{currstate}|\text{state}, \pi^*_{\underline{-}}(\text{state})) > 0$ **do**
6:          **if** currstate is not goal and currstate is not in the set visited **then**
7:              visited.insert(currstate)
8:              **if** $J_{\underline{-}}(\text{currstate})$ has not converged **then**
9:                  if required, change the expected durations of the actions that are currently executing in
                    currstate.
10:                 solve a deterministic-duration planning problem with the start state currstate
11:             insert currstate in the queue open
12: **until** open is empty
```

Line 9 of Algorithm 6 assigns a new expected duration for all actions that are currently running in the current state and have not completd by the time of their previous termination point. This reassignment follows the similar case in the online version (line 13).

Example: Consider a domain with two state-variables, x_1 and x_2, with two actions set-x_1 and set-x_2. The task is to set both variables (initially they are both false). Assume that set-x_2 always succeeds whereas set-x_1 succeeds with only 0.5 probability. Moreover, let both actions have a uniform duration distribution of 1, 2, or 3. In such a case a complete interwoven epoch search could touch 36 interwoven states (each state variable could be true or false, each action could be "not running", "running for 1 unit", and "running for 2 units"). Instead, if we build a deterministic duration policy then each action's deterministic duration will be 2, and so the total number of states touched will be from the 16 interwoven states (each action could now only be "not running" or "running for 1 unit").

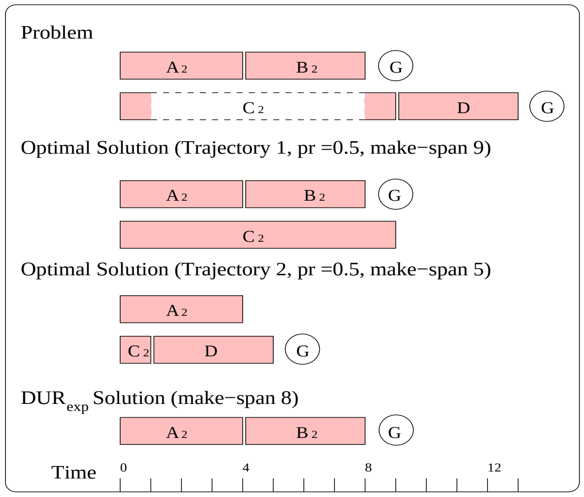

Problem
A 2
B 2
G
C 2
D
G
Optimal Solution (Trajectory 1, pr =0.5, make−span 9)
A 2
B 2
G
C 2
Optimal Solution (Trajectory 2, pr =0.5, make−span 5)
A 2
C 2
D
G
DUR exp Solution (make−span 8)
A 2
B 2
G
Time
0
4
8
12

Figure 14: An example of a domain where the ΔDUR$_{\text{exp}}$ algorithm does not compute an optimal solution.

Now, suppose that the deterministic planner decides to execute both actions in the start state. Having committed to this combination, it is easy to see that certain states will never be reached. For example, the state $\langle(\neg x_1, \neg x_2), \{(\text{set}-\text{x}_1, 2)\}\rangle$ can never be visited, since once set-x_2 completes it is guaranteed that x_2 will be set. In fact, in our example, only 3 new states will initiate offline replanning (line 10 in Algo 6), *viz.*, $\langle(x_1, \neg x_2), \{(\text{set}-\text{x}_2, 2)\}\rangle$, $\langle(\neg x_1, \neg x_2), \{(\text{set}-\text{x}_2, 2)\}\rangle$, and $\langle(\neg x_1, x_2), \{(\text{set}-\text{x}_1, 2)\}\rangle$ $\square$

6.2.3 Properties

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

Example: We consider a domain which has actions $A_{2:n}, B_{2:n}, C_{2:n}$ and D. Each A_i and B_i takes time 2^i. Each C_i has a probabilistic duration: with probability 0.5, C_i takes 1 unit of time, and with the remaining probability, it takes $2^{i+1} + 1$ time. Thus, the expected duration of C_i is $2^i + 1$. D takes 4 units. In sub-problem SP_i, the goal may be reached by executing A_i followed by B_i. Alternatively, the goal may be reached by first executing C_i and then recursively solving the sub-problem SP_{i-1}. In this domain, the ΔDUR$_{\text{exp}}$ algorithm will always compute $\langle A_i; B_i\rangle$ as the best solution. However, the optimal policy starts both $\{A_i, C_i\}$. If C_i terminates at 1, the policy executes the solution for SP_{i-1}; otherwise, it waits until A_i terminates and then executes B_i. Figure 14 illustrates the sub-problem SP_2 in which the optimal policy has an expected make-span of 7 (*vs.* ΔDUR$_{\text{exp}}$'s make-span of 8). In general, the expected make-span of the optimal policy on SP_n is $\frac{1}{3}[2^{n+2} + 2^{4-n}] + 2^{2-n} + 2$. Thus, $\lim_{n\to\infty} \frac{exp}{opt} = \frac{3}{2}$.$\square$

6.3 Multi-Modal Duration Distributions

The planners of the previous two sections benefited by considering the small set of happenings instead of pivots, an approach licensed by Conjecture 8. Unfortunately, this simplification is not warranted in the case of actions with multi-modal duration distributions, which can be common in complex domains where all factors can't be modeled explicitly. For example, the amount of time for a Mars rover to transmit data might have a bimodal distribution — normally it would take little time, but if a dust storm were in progress (unmodeled) it could take much longer. To handle these cases we model durations with a mixture of Gaussians parameterized by the triple $\langle amplitude, mean, variance \rangle$.

6.3.1 CoMDP Formulation

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

Two changes are required to the transition function of Algorithm 4. In line 3, the *maxtime* computation now involves time until the next pivot in the increasing remaining time region for all actions with multi-modal distributions (thus forcing us to take a decision at those points, even when no action terminates). Another change (in line 6) allows a *non-empty* subset $Asub_t$ for $t = maxtime$. That is, next state is computed even without any action termination. By making these changes in the transition function we reformulate our problem as a CoMDP in the interwoven space and thus solve, using our previous methods of pruned/sampled RTDP, hybrid algorithm or expected-duration algorithm.

6.3.2 Archetypal-Duration Planner

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

6.4 Experiments: Planning with Stochastic Durations

We now evaluate our techniques for solving planning problems involving stochastic durations. We compare the computation time and solution quality (make-span) of our five planners for domains with and without multi-modal duration distributions. We also re-evaluate the effectiveness of the maximum- (*MC*) and average-concurrency (*AC*) heuristics for these domains.

6.4.1 Experimental Setup

We modify our Rover, MachineShop, and Artificial domains by additionally including uncertainty in action durations. For this set of experiments, our largest problem had 4 million world states of which 65536 were reachable. Our algorithms explored up to 1,000,000 distinct states in the interwoven state space during planning. The domains contained as many as 18 actions, and some actions had as many as 13 possible durations. For more details on the domains please refer to the longer version (Mausam, 2007).

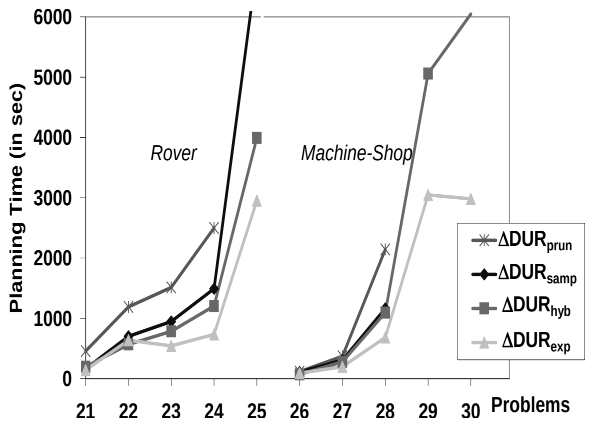


Figure 15: Planning time comparisons for Rover and MachineShop domains: Variation along algorithms when all initialized by the average concurrency (AC) heuristic; $\Delta\text{DUR}_{\text{exp}}$ performs the best.

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

Table 4: All three planners produce near-optimal policies as shown by this table of ratios to the optimal make-span.[11]

6.4.2 Comparing Running Times

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

Figure 15 reports the running times of various algorithms (initialized with the AC heuristic) on the Rover and Machine-Shop domains when all durations are unimodal. $\Delta\text{DUR}_{\text{exp}}$ out-performs the other planners by substantial margins. As this algorithm is solving a comparatively simpler problem, fewer states are expanded and thus the approximation scales better than others — solving, for example, two Machine-Shop problems, which were too large for most other planners. In most cases hybridization speeds planning by significant amounts, but it performs better than $\Delta\text{DUR}_{\text{exp}}$ only for the artificial domain.

6.4.3 Comparing Solution Quality

We measure quality by simulating the generated policy across multiple trials. We report the ratio of average expected make-span and the optimal expected make-span for domains with all unimodal distributions in Table 4. We find that the make-spans of the inadmissible heuristic AC are at par

11. If the optimal algorithm doesn't converge, we use the best solution found across all runs as "optimal".

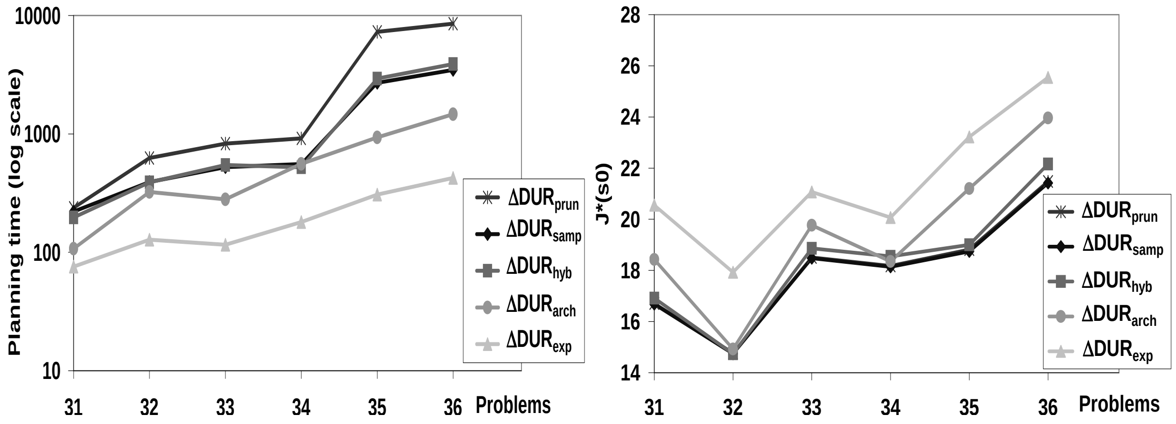


Figure 16: Comparisons in the Machine-Shop domain with multi-modal distributions. (a) Computation Time comparisons: $\Delta\text{DUR}_{\text{exp}}$ and $\Delta\text{DUR}_{\text{arch}}$ perform much better than other algos. (b) Make-spans returned by different algos: Solutions returned by $\Delta\text{DUR}_{\text{samp}}$ are almost optimal. Overall $\Delta\text{DUR}_{\text{arch}}$ finds a good balance between running time and solution quality.

with those of the admissible heuristic MC. The hybridized planner is approximate with a user-defined bound. In our experiments, we set the bound to 5% and find that the make-spans returned by the algorithm are quite close to the optimal and do not always differ by 5%. $\Delta\text{DUR}_{\text{exp}}$ has no quality guarantees, still the solutions returned on the problems we tested upon are nearly as good as other algorithms. Thus, we believe that this approximation will be quite useful in scaling to larger problems without losing solution quality.

6.4.4 Multimodal Domains

We develop multi-modal variants of our domains; *e.g.*, in the Machine-Shop domain, time for fetching paint was bimodal (if in stock, paint can be fetched fast, else it needs to be ordered). There was an alternative but costly paint action that doesn't require fetching of paint. Solutions produced by $\Delta\text{DUR}_{\text{samp}}$ made use of pivots as decision epochs by starting the costly paint action in case the fetch action didn't terminate within the first mode of the bimodal distribution (*i.e.* paint was out of stock).

The running time comparisons are shown in Figure 16(a) on a log-scale. We find that $\Delta\text{DUR}_{\text{exp}}$ terminates extremely quickly and $\Delta\text{DUR}_{\text{arch}}$ is not far behind. However, the make-span comparisons in Figure 16(b) clearly illustrate the approximations made by these methods in order to achieve planning time. $\Delta\text{DUR}_{\text{arch}}$ exhibits a good balance of planning time and solution quality.

7. Related Work

This paper extends our prior work, originally reported in several conference publications (Mausam & Weld, 2004, 2005, 2006a, 2006b).

Temporal planners may be classified as using constraint-posting or extended state-space methods (discussed earlier in Section 4). While the constraint approach is promising, few (if any) *probabilistic* planners have been implemented using this architecture; one exception is Buridan (Kush-

	concurrent		*non-concurrent*	
	durative	*non-durative*	*durative*	*non-durative*
stochastic	ΔDUR, Tempastic, GSMDP, Prottle, FPG, Aberdeen *et al.*	Concurrent MDP, Factorial MDP, Paragraph	Time Dependent MDP, IxTeT, CIRCA, Foss & Onder	MDP (RTDP, LAO*, *etc.*)
deterministic	Temporal Planning (TP4, Sapa, MIPS TLPlan, *etc.*)	Step-optimal planning (GraphPlan, SATPlan)	Planning with Numerical Resources (Sapa, Metric-FF, CPT)	Classical Planning (HSP, FF, *etc.*)

Figure 17: A table listing various planners that implement different subsets of concurrent, stochastic, durative actions.

merick, Hanks, & Weld, 1995), which performed poorly. In contrast, the MDP community has proven the state-space approach successful. Since the powerful deterministic temporal planners, which have won the various planning competitions, also use the state-space approach, we adopt it for our algorithms that combine temporal planning with MDPs. It may be interesting to incorporate constraint-based approaches in a probabilistic paradigm and compare against the techniques of this paper.

7.1 Comparison with Semi-MDPs

A *Semi-Markov Decision Process* is an extension of MDPs that allows durative actions to take variable time. A discrete time semi-MDP can be solved by solving a set of equations that is a direct extension of Equations 2. The techniques for solving discrete time semi-MDPs are natural generalizations of those for MDPs. The main distinction between a semi-MDP and our formulation of concurrent probabilistic temporal planning with stochastic durations concerns the presence of concurrently executing actions in our model. A semi-MDP does not allow for concurrent actions and assumes one executing action at a time. By allowing concurrency in actions and intermediate decision epochs, our algorithms need to deal with large state and action spaces, which is not encountered by semi-MDPs.

Furthermore, Younes and Simmons have shown that in the general case, semi-MDPs are incapable of modeling concurrency. A problem with concurrent actions and stochastic continuous durations needs another model known as Generalized Semi-Markov Decision Process (GSMDP) for a precise mathematical formulation (Younes & Simmons, 2004b).

7.2 Concurrency and Stochastic, Durative Actions

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

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

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

FPG (Aberdeen & Buffet, 2007) learns a separate neural network for each action individually based on the current state. In the execution phase the decision, *i.e.*, whether an action needs to be executed or not, is taken independently of decisions regarding other actions. In this way FPG is able to effectively sidestep the blowup caused by exponential combinations of actions. In practice it is able to very quickly compute high quality solutions.

Rohanimanesh and Mahadevan (2001) investigate concurrency in a hierarchical reinforcement learning framework, where abstract actions are represented by *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.

Aberdeen *et al.* plan with concurrent, durative actions with *deterministic* durations in a specific military operations domain. They apply various domain-dependent heuristics to speed the search in an extended state space (Aberdeen, Thiebaux, & Zhang, 2004).

7.3 Concurrency and Stochastic, Non-durative Actions

Meuleau *et al.* and Singh & Cohn deal with a special type of MDP (called a factorial MDP) 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 (Meuleau, Hauskrecht, Kim, Peshkin, Kaelbling, Dean, & Boutilier, 1998; Singh & Cohn, 1998). 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. Singh & Cohn present an optimal algorithm (similar to combo-elimination used in DUR_{prun}), whereas domain specific heuristics in Meuleau *et al.*have no such guarantees. 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.

Paragraph (Little & Thiebaux, 2006) formulates the planning with concurrency as a regression search over the probabilistic planning graph. It uses techniques like nogood learning and mutex reasoning to speed policy construction.

Guestrin *et al.* solve the multi-agent MDP problem by using a linear programming (LP) formulation and expressing the value function as a linear combination of basis functions. By assuming that these basis functions depend only on a few agents, they are able to reduce the size of the LP (Guestrin, Koller, & Parr, 2001).

7.4 Stochastic, Non-concurrent, Durative Actions

Many researchers have studied planning with stochastic, durative actions in *absence* of concurrency. For example, Foss and Onder (2005) use *simple temporal networks* to generate plans in which the objective function has no time component. Simple Temporal Networks allow effective temporal constraint reasoning and their methods can generate temporally contingent plans.

Boyan and Littman (2000) propose *Time-dependent MDPs* to model problems with (non-concurrent) actions having time-dependent, stochastic durations; their solution generates piece-wise linear value functions.

NASA researchers have developed techniques for generating non-concurrent plans with uncertain continuous durations using a greedy algorithm which incrementally adds branches to a straight-line plan (Bresina et al., 2002; Dearden, Meuleau, Ramakrishnan, Smith, & Washington, 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 not guaranteed to reach the goal.

IxTeT is a temporal planner that uses constraint based reasoning within partial order planning (Laborie & Ghallab, 1995). It embeds temporal properties of actions as constraints and does not optimize make-span. CIRCA is an example of a system that plans with uncertain durations where each action is associated with an unweighted set of durations (Musliner, Murphy, & Shin, 1991).

7.5 Deterministic, Concurrent, Durative Actions

Planning with deterministic actions is a comparitively simpler problem and much of the work in planning under uncertainty is based on the previous, deterministic planning research. For instance, our interwoven state representation and transition function are extensions of the extended state representations in TP4, SAPA, and TLPlan (Haslum & Geffner, 2001; Do & Kambhampati, 2003; Bacchus & Ady, 2001).

Other planners, like MIPS and AltAlt^p, have also investigated fast generation of parallel plans in deterministic settings (Edelkamp, 2003; Nigenda & Kambhampati, 2003) and Jensen and Veloso (2000) extend it to problems with disjunctive uncertainty.

8. Future Work

Having presented a comprehensive set of techniques to handle probabilistic outcomes, concurrent and durative actions in a single formalism, we now direct our attention towards different relaxations and extensions to the proposed model. In particular, we explore other objective functions, infinite horizon problems, continuous-valued duration distributions, temporally expressive action models, degrees of goal satisfaction and interruptibility of actions.

8.1 Extension to Other Cost Functions

For the planning problems with durative actions (sections 4 and beyond) we focused on make-span minimization problems. However, our techniques are quite general and are applicable (directly or with minor variations) to a variety of cost metrics. As an illustration, consider the mixed cost optimization problem in which in addition to the duration of each action, we are also given the amount of resource consumed per action, and we wish to minimize the the sum of make-span and total resource usage. Assuming that the resource consumption is unaffected by concurrent execution, we can easily compute a new max-concurrency heuristic. The mixed-cost counterpart for Equations 12 is:

$$\begin{aligned} J^*_{\underline{\underline{-}}}(s) &\geq \frac{J^*_t(X)}{c} + J^*_r(X) \qquad \text{for } Y = \emptyset \\ J^*_{\underline{\underline{-}}}(s) &\geq \frac{Q^*_t(X, A_s)}{c} + Q^*_r(X, A_s) \quad \text{for } Y \neq \emptyset \end{aligned} \tag{15}$$

Here, J_t is for the single-action MDP assignng costs to be durations and J_r is for the single action MDP assigning costs to be resource consumptions. A more informed average concurrency heuristic can be similarly computed by replacing maximum concurrency by average concurrency. The hybridized algorithm follows in the same fashion, with the fast algorithm being a CoMDP solved using techniques of Section 3.

On the same lines, if the objective function is to minimize make-span given a certain maximum resource usage, then the total amount of resource remaining can be included in the state-space for all the CoMDPs and underlying single-action MDPs *etc.* and the same techniques may be used.

8.2 Infinite Horizon Problems

Until now this paper has defined the techniques for the case of *indefinite* horizon problems, in which an absorbing state is defined as is reachable. For other problems an alternative formulation is preferred that allows for infinite execution but discounts the future costs by multiplying them by a discount factor in each step. Again, our techniques can be suitably extended for such scenario. For example, Theorem 2 gets modified to the following:

$$Q_{\|}(s, A) \geq \gamma^{1-k} Q_{\|}(s, \{a_1\}) + \mathcal{C}_{\|}(A) - \left(\sum_{i=1}^{k} \gamma^{i-k} \mathcal{C}_{\|}(\{a_i\}) \right)$$

Recall that this theorem provides us with the pruning rule, combo-skipping. Thus, we can use Pruned RTDP with the new pruning rule.

8.3 Extensions to Continuous Duration Distributions

Until now we have confined ourselves to actions with discrete durations (refer to Assumption 3). We now investigate the effects of dealing directly with continuous uncertainty in the duration distributions. Let $f_i^T(t)dt$ be the probability of action a_i completing between times $t+T$ and $t+T+dt$, conditioned on action a_i not finishing until time T. Similarly, define $F_i^T(t)$ to be the probability of the action finishing *after* time $t+T$.

Let us consider the extended state $\langle X, \{(a_1, T)\}\rangle$, which denotes that action a_1 started T units ago in the world state X. Let a_2 be an applicable action that is started in this extended state. Define $M = \min(\Delta_M(a_1) - T, \Delta_M(a_2))$, where Δ_M denotes the maximum possible duration of execution for each action. Intuitively, M is the time by which at least one action will complete. Then

$$\begin{aligned} Q_{\underline{\underline{-}}\,n+1}(\langle X, \{(a_1, T)\}\rangle, a_2) = &\int_0^M f_1^T(t) F_2^0(t) \left[t + J_{\underline{\underline{-}}\,n}(\langle X_1, \{a_2, t\}\rangle) \right] dt \, + \\ &\int_0^M F_1^T(t) f_2^0(t) \left[t + J_{\underline{\underline{-}}\,n}(\langle X_2, \{a_1, t+T\}\rangle) \right] dt \end{aligned} \tag{16}$$

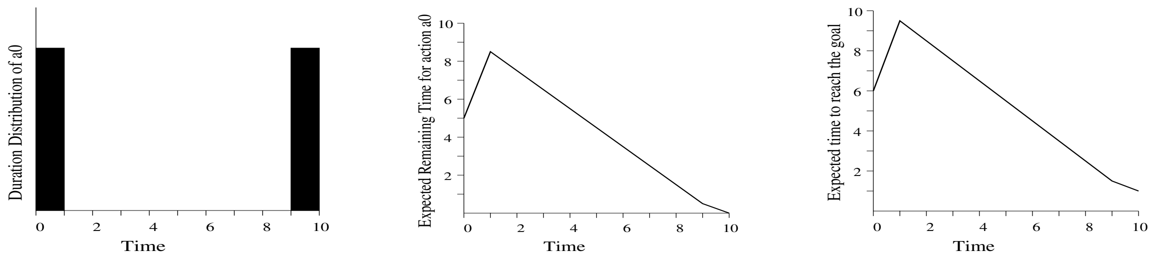


Figure 18: If durations are continuous (real-valued) rather than discrete, there may be an infinite number of potentially important decision epochs. In this domain, a crucial decision epoch could be required at any time in $(0, 1]$ — depending on the length of possible alternate plans.

Here X_1 and X_2 are world states obtained by applying the deterministic actions a_1 and a_2 respectively on X. Recall that $J_{=n+1}(s) = \min_a Q_{=n+1}(s, a)$. For a fixed point computation of this form, we desire that J_{n+1} and J_n have the same functional form[12]. Going by the equation above this seems very difficult to achieve, except perhaps for very specific action distributions in some special planning problems. For example, if all distributions are constant or if there is no concurrency in the domain, then these equations are easily solvable. But for more interesting cases, solving these equations is a challenging open question.

Furthermore, dealing with continuous *multi-modal* distributions worsens the decision epochs explosion. We illustrate this with the help of an example.

Example: Consider the domain of Figure 7 except that let action a_0 have a bimodal distribution, the two modes being uniform between 0-1 and 9-10 respectively as shown in Figure 18(a). Also let a_1 have a very small duration. Figure 18(b) shows the expected remaining termination times if a_0 terminates at time 10. Notice that due to bimodality, this expected remaining execution time increases between 0 and 1. The expected time to reach the goal using plan $\langle \{a_0, a_1\}; a_2 \rangle$ is shown in the third graph. Now suppose, we have started $\{a_0, a_1\}$, and we need to choose the next decision epoch. It is easy to see that the optimal decision epoch could be any point between 0 and 1 and would depend on the alternative routes to the goal. For example, if duration of b_0 is 7.75, then the optimal time-point to start the alternative route is 0.5 (right after the expected time to reach the goal using first plan exceeds 7.75).

Thus, the choice of decision epochs depends on the expected durations of the alternative routes. But these values are not known in advance, in fact these are the ones being calculated in the planning phase. Therefore, choosing decision epochs ahead of time does not seem possible. This makes the optimal continuous multi-modal distribution planning problem mostly intractable for any reasonable sized problem.

8.4 Generalizing the TGP Action Model

The assumption of TGP style actions enables us to compute optimal policies, since we can prune the number of decision epochs. In the case of complex action models like PDDL$_{2.1}$ (Fox & Long, 2003), all old, deterministic state-space planners are incomplete. For the same reasons, our algorithms are

12. This idea has been exploited in order to plan with continuous resources (Feng, Dearden, Meuleau, & Washington, 2004).

incomplete for problems in PPDDL$_{2.1}$. Recently, Cushing *et al.* has introduced Tempo, a state-space planner, which uses lifting over time in to achieve completeness (Cushing, Kambhampati, Mausam, & Weld, 2007). In pursuit of finding a complete, state-space, probabilistic planner for complex action models, a natural step is to consider a Tempo-like representation in a probabilistic setting. While working out the details seems relatively straightforward, the important research challenge will be to find the right heuristics to streamline the search so that the algorithm can scale.

8.5 Other Extensions

There are several other extensions to the basic framework that we have suggested. Each different construct introduces additional structure and we need to exploit the knowledge in order to design fast algorithms. Many times, the basic algorithms proposed in this paper may be easily adapted to such situations, sometimes they may be not. We list two of the important extensions below.

- **Notion of Goal Satisfaction:** Different problems may require slightly different notions of when a goal is reached. For example, we have assumed thus far that a goal is not “officially achieved” until all executed actions have terminated. Alternatively, one might consider a goal to be achieved if a satisfactory world state is reached, even though some actions may be in the midst of execution. There are intermediate possibilities in which a goal requires some *specific* actions to necessarily end. Just by changing the definition of the goal set, these problems can be modeled as a CoMDP. The hybridized algorithm and the heuristics can be easily adapted for this case.

- **Interruptible Actions:** We have assumed that, once started, an action cannot be terminated. However, a richer model may allow preemptions, as well as the continuation of an interrupted action. The problems, in which all actions could be interrupted at will, have a significantly different flavor. Interrupting an action is a new kind of decision and requires a full study of when might an action termination be useful. To a large extent, planning with these is similar to finding different concurrent paths to the goal and starting all of them together, since one can always interrupt all the executing paths as soon as the goal is reached. For instance, example in Figure 7 no longer holds since b_0 can be started at time 1, and later terminated as needed to shorten the make-span.

8.6 Effect of Large Durations

A weakness of all extended-state space approaches, both in deterministic as well as probabilistic settings, is the dependence on absolute durations (or to be more accurate, the greatest common divisor of action durations). For instance, if the domain has an action a with a large duration, say 100 and another concurrently executable action with duration 1, then all world states will be explored with the tuples $(a, 1)$, $(a, 2)$, …, $(a, 98)$, $(a, 99)$. In general, many of these states will behave similarly and there will be certain decision boundaries that will be important. “Start b if a has been executing for 50 units and c otherwise” is one example of such a decision boundary. Instead of representing all these flat discrete states individually, planning in an aggregate space in which each state represents several extended states will help alleviate this inefficiency.

However, it is not obvious how to achieve such an aggregation automatically, since adapting the well-known methods for aggregation do not hold in our case. For instance, SPUDD (Hoey et al., 1999) uses algebraic decision diagrams to represent abstract states that have the same J-value. Aggregating the same valued states may not be enough for us, since the expected time of completion depends linearly on the amount of time left for the longest executing action. So, all the states which differ only by the amount of time an action has been executing will not be able to aggregate together. In a similar way, Feng *et al.* (2004) use piecewise constant and piecewise linear representations to adaptively discretize continuous variables. In our case, we have $|\mathcal{A}|$ of such variables. While only a few of them that are executing are active at a given time, modeling a sparse high-dimensional value function is not easy either. Being able to exploit this structure due to action durations is an essential future direction in order to scale the algorithms to complex real world domains.

9. Conclusions

Although concurrent and durative actions with stochastic effects characterize many real-world domains, few planners can handle all these challenges in concert. This paper proposes a unified state-space based framework to model and solve such problems. State space formulations are popular both in deterministic temporal planning as well as in probabilistic planning. However, each of these features bring in additional complexities to the formulation and afford new solution techniques. We develop the "DUR" family of algorithms to alleviates these complexities. We evaluate the techniques on the running times and qualities of solutions produced. Moreover, we study the theoretical properties of these domains and also identify key conditions under which fast, optimal algorithms are possible. We make the following contributions:

1. We define Concurrent MDPs (CoMDP) — an extension of the MDP model to formulate a stochastic planning problem with concurrent actions. A CoMDP can be cast back into a new MDP with an extended action space. Because this action space is possibly exponential in the number of actions, solving the new MDP naively may take a huge performance hit. We develop the general notions of *pruning* and *sampling* to speed up the algorithms. Pruning refers to pruning of the provably sub-optimal action-combinations for each state, thus performing less computation but still guaranteeing optimal solutions. Sampling-based solutions rely on an intelligent sampling of action-combinations to avoid dealing with their exponential number. This method converges orders of magnitude faster than other methods and produces near-optimal solutions.

2. We formulate the planning with concurrent, durative actions as a CoMDP in two modified state spaces — aligned epoch, and interwoven epoch. While aligned epoch based solutions run very fast, interwoven epoch algorithms yield a much higher quality solutions. We also define two heuristic functions — maximum concurrency (*MC*), and average concurrency (*AC*) to guide the search. *MC* is an admissible heuristic, whereas *AC*, while inadmissible, is typically more-informed leading to better computational gains. We call our algorithms the "DUR" family of algorithms. The subscripts *samp* or *prun* refer to sampling and pruning respectively, optional superscripts *AC* or *MC* refer to the heuristic employed, if any and an optional "Δ" before DUR notifies a problem with stochastic durations. For example, Labeled RTDP for a deterministic duration problem employing sampling and started with *AC* heuristic will be abbreviated as $\mathrm{DUR}^{\mathrm{AC}}_{\mathrm{samp}}$.

3. We also develop the general technique of hybridizing two planners. Hybridizing interwoven-epoch and aligned-epoch CoMDPs yields a much more efficient algorithm, $\mathrm{DUR}_{\mathrm{hyb}}$. The algorithm has a parameter, which can be varied to trade-off speed against optimality. In our experiments, $\mathrm{DUR}_{\mathrm{hyb}}$ quickly produces near-optimal solutions. For larger problems, the speedups over other algorithms are quite significant. The hybridized algorithm can also be used in an anytime fashion thus producing good-quality proper policies (policies that are guaranteed to reach the goal) within a desired time. Moreover, the idea of hybridizing two planners is a general notion; recently it has been applied to solving general stochastic planning problems (Mausam, Bertoli, & Weld, 2007).

4. Uncertainty in durations leads to more complexities because in addition to state and action spaces, there is also a blowup in the branching factor and in the number of decision epochs. We bound the space of decision epochs in terms of pivots (times when actions may potentially terminate) and conjecture further restrictions, thus making the problem tractable. We also propose two algorithms, the expected duration planner ($\Delta\mathrm{DUR}_{\mathrm{exp}}$) and the archetypal duration planner ($\Delta\mathrm{DUR}_{\mathrm{arch}}$), which successively solve small planning problems each with no or limited duration uncertainty, respectively. $\Delta\mathrm{DUR}_{\mathrm{arch}}$ is also able to make use of the additional structure offered by multi-modal duration distributions. These algorithms perform much faster than other techniques. Moreover, $\Delta\mathrm{DUR}_{\mathrm{arch}}$ offers a good balance between planning time *vs.* solution quality tradeoff.

5. Besides our focus on stochastic actions, we expose important theoretical issues related with durative actions which have repercussions to deterministic temporal planners as well. In particular, we prove that all common state-space temporal planners are incomplete in the face of expressive action models, *e.g.*, $\mathrm{PDDL}_{2.1}$, a result that may have a strong impact on the future temporal planning research (Cushing et al., 2007).

Overall, this paper proposes a large set of techniques that are useful in modeling and solving planning problems employing stochastic effects, concurrent executions and durative actions with duration uncertainties. The algorithms range from fast but suboptimal solutions, to relatively slow but optimal. Various algorithms that explore different intermediate points in this spectrum are also presented. We hope that our techniques will be useful in scaling the planning techniques to real world problems in the future.

Acknowledgements

We thank Blai Bonet for providing the source code of GPT as well as for comments in the course of this work. We are thankful to Sumit Sanghai for his theorem proving skills and advice at various stages of this research. We are grateful to Derek Long and the anonymous reviewers of this paper who gave several thoughtful suggestions for generalizing the theory and improving the clarity of the text. We also thank Subbarao Kambhampati, Daniel Lowd, Parag, David Smith and all others who provided useful comments on drafts on parts of this research. This work was performed at University of Washington between 2003 and 2007 and was supported by generous grants from National Aeronautics and Space Administration (Award NAG 2-1538), National Science Foundation (Award IIS-0307906), and Office of Naval Research (Awards N00014-02-1-0932, N00014-06-1-0147) and the WRF / TJ Cable Professorship.

References

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

Aberdeen, D., & Buffet, O. (2007). Concurrent probabilistic temporal planning with policy-gradients. In *ICAPS'07*.

Bacchus, F., & Ady, M. (2001). Planning with resources and concurrency: A forward chaining approach. In *IJCAI'01*, pp. 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., & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1–2), 281–300.

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

Bonet, B., & Geffner, H. (2005). mGPT: A probabilistic planner based on heuristic search. *JAIR*, *24*, 933.

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

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

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

Chen, Y., Wah, B. W., & Hsu, C. (2006). Temporal planning using subgoal partitioning and resolution in sgplan. *JAIR*, *26*, 323.

Cushing, W., Kambhampati, S., Mausam, & Weld, D. S. (2007). When is temporal planning really *temporal*?. In *IJCAI'07*.

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

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

Do, M. B., & Kambhampati, S. (2003). Sapa: A scalable multi-objective metric temporal planner. *JAIR*, *20*, 155–194.

Edelkamp, S. (2003). Taming numbers and duration in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, *20*, 195–238.

Feng, Z., Dearden, R., Meuleau, N., & Washington, R. (2004). Dynamic programming for structured continuous Markov decision processes. In *UAI'04*, p. 154.

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

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

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

Guestrin, C., Koller, D., & Parr, R. (2001). Max-norm projections for factored MDPs. In *IJCAI'01*, pp. 673–682.

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

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

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. In *UAI'99*, pp. 279–288.

Jensen, R. M., & Veloso, M. (2000). OBDD=based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, *13*, 189.

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

Laborie, P., & Ghallab, M. (1995). Planning with sharable resource constraints. In *IJCAI'95*, p. 1643.

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

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

Long, D., & Fox, M. (2003). The 3rd international planning competition: Results and analysis. *JAIR*, *20*, 1–59.

Mausam (2007). Stochastic planning with concurrent, durative actions. Ph.d. dissertation, University of Washington.

Mausam, Bertoli, P., & Weld, D. (2007). A hybridized planner for stochastic domains. In *IJCAI'07*.

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

Mausam, & Weld, D. (2005). Concurrent probabilistic temporal planning. In *ICAPS'05*, pp. 120–129.

Mausam, & Weld, D. (2006a). Challenges for temporal planning with uncertain durations. In *ICAPS'06*.

Mausam, & Weld, D. (2006b). Probabilistic temporal planning with uncertain durations. In *AAAI'06*.

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*, pp. 165–172.

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

Nigenda, R. S., & Kambhampati, S. (2003). Altalt-p: Online parallelization of plans with heuristic state search. *Journal of Artificial Intelligence Research*, *19*, 631–657.

Penberthy, J., & Weld, D. (1994). Temporal planning with continuous change. In *AAAI'94*, p. 1010.

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

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

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

Vidal, V., & Geffner, H. (2006). Branching and pruning: An optimal temporal pocl planner based on constraint programming. *AIJ*, *170(3)*, 298–335.

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

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

Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *IJCAI'95*, pp. 1114–1120.

Appendix

Proof of Theorem 6

We now prove the statement of Theorem 6, *i.e.*, if all actions are *TGP-style* then the set of pivots suffices for optimal planning. In the proof make use of the fact that if all actions are *TGP-style* then a consistent execution of any concurrent plan requires that any two executing actions be non-mutex (refer to Section 5 for an explanation on that). In particular, none of their effects conflict and a precondition of one does not conflict with the effects of the another.

We prove our theorem by contradition. Let us assume that for a problem each optimal solution requires at least one action to start at a non-pivot. Let us consider one of those optimal plans, in which the first non-pivot point at which an action needs to start at a non-pivot is minimized. Let us name this time point t and let the action that starts at that point be a. We now prove by a case analysis that we may, as well, start a at time $t-1$ without changing the nature of the plan. If $t-1$ is also a non-pivot then we contradict the hypothesis that t is the *minimum* first non-pivot point. If $t-1$ is pivot then our hypothesis is contradicted because a does not "need to" start at a non-pivot.

To prove that a can be left-shifted by 1 unit, we take up one trajectory at a time (recall that actions could have several durations) and consider all actions playing a role at $t-1$, t, $t+\Delta(a)-1$, and $t+\Delta(a)$, where $\Delta(a)$ refers to the duration of a in this trajectory. Considering these points suffice, since the system state does not change at any other points on the trajectory. We prove that the execution of none of these actions is affected by this left shift. There are the following twelve cases:

1. $\forall$actions b that start at $t-1$: b can't end at t (t is a non-pivot). Thus a and b execute concurrently after t, implies a and b are non-mutex. Thus a and b may as well start together.

2. $\forall$actions b that continue execution at $t-1$: Use the argument similar to case 1 above.

3. $\forall$actions b that end at $t-1$: Because b is *TGP-style*, its effects are realized in the open interval ending at $t-1$. Therefore, start of a does not conflict with the end of b.

4. $\forall$actions b that start at t: a and b start together and hence are not dependent on each other for preconditions. Also, they are non-mutex, so their starting times can be shifted in any direction.

5. $\forall$actions b that continue execution at t: If b was started at $t-1$ refer to case 1 above. If not, t and $t-1$ are both similar points for b.

6. $\forall$actions b that end at t: Case not possible due to the assumption that t is a non-pivot.

7. $\forall$actions b that start at $t+\Delta(a)-1$: Since a continued execution at this point, a and b are non-mutex. Thus a's effects do not clobber b's preconditions. Hence, b can still be executed after realizing a's effects.

8. $\forall$actions b that continue execution at $t+\Delta(a)-1$: a and b are non-mutex, so a may end earlier without any effect of b.

9. $\forall$actions b that end at $t+\Delta(a)-1$: a and b were executing concurrently. Thus they are non-mutex. So they may end together.

10. $\forall$actions b that start at $t+\Delta(a)$: b may still start at $t+\Delta(a)$, since the state of $t+\Delta(a)$ doesn't change.

11. $\forall$actions b that continue execution at $t+\Delta(a)$: If b was started at $t+\Delta(a)-1$ refer to case 7 above, else there is no state change at $t+\Delta(a)$ to cause any effect on b.

12. $\forall$actions b that end at $t+\Delta(a)$: a and b are non-mutex because they were executing concurrently. Thus, a's effects don't clobber b's preconditions. Hence, a may end earlier.

Since a can be left shifted in all the trajectories, therefore the left-shift is legal. Also, if there are multiple actions a that start at t they may each be shifted one by one using the same argument. Hence Proved. $\square$