Towards Faster Planning with Continuous Resources in Stochastic Domains

Janusz Marecki and **Milind Tambe**
Computer Science Department
University of Southern California
941 W 37th Place, Los Angeles, CA 90089
{marecki, tambe}@usc.edu

Abstract

Agents often have to construct plans that obey resource limits for continuous resources whose consumption can only be characterized by probability distributions. While Markov Decision Processes (MDPs) with a state space of continuous and discrete variables are popular for modeling these domains, current algorithms for such MDPs can exhibit poor performance with a scale-up in their state space. To remedy that we propose an algorithm called DPFP. DPFP's key contribution is its exploitation of the dual space cumulative distribution functions. This dual formulation is key to DPFP's novel combination of three features. First, it enables DPFP's membership in a class of algorithms that perform forward search in a large (possibly infinite) policy space. Second, it provides a new and efficient approach for varying the policy generation effort based on the likelihood of reaching different regions of the MDP state space. Third, it yields a bound on the error produced by such approximations. These three features conspire to allow DPFP's superior performance and systematic trade-off of optimality for speed. Our experimental evaluation shows that, when run stand-alone, DPFP outperforms other algorithms in terms of its any-time performance, whereas when run as a hybrid, it allows for a significant speedup of a leading continuous resource MDP solver.

Introduction

Recent years have seen a rise of interest in the deployment of unmanned vehicles in dynamic and uncertain environments. Whereas some of these vehicles can be tele-operated, vehicles such as the Mars rovers may have to act autonomously due to inherent communication restrictions (Bresina et al. 2002; Meuleau, Dearden, and Washington 2004). Furthermore, autonomous vehicles must deal with energy requirements, or more generally, resource requirements whose consumption is often non-deterministic. As a result, agents that control these vehicles must be able to plan for dynamic, uncertain and non-discrete environments.

Continuous resource MDPs have emerged as a powerful planning technique to address such domains with uncertain resource consumption and resource limits. Several promising policy iteration algorithms (Lagoudakis and Parr 2003; Nikovski and Brand 2003; Guestrin, Hauskrecht, and Kveton 2004; Hauskrecht and Kveton 2004) and value iteration algorithms (Feng et al. 2004; Li and Littman 2005; Marecki, Koenig, and Tambe 2007) have been identified for policy generation in such MDPs. Furthermore, proposed improvements to policy and value iteration algorithms (Boutilier, Dearden, and Goldszmidt 2000; Mausam et al. 2005) have dramatically increased the applicability of continuous state MDPs to many real-world planning problems. Unfortunately, these techniques can still be inefficient (Meuleau, Dearden, and Washington 2004), especially when the discrete part of the states leads to the combinatorial explosion of the state-space — the problem commonly referred to as the curse of dimensionality.

The key contribution of this paper is the **D**ynamic **P**robability **F**unction **P**ropagation (DPFP) algorithm that enables scale-up of continuous resource MDP solvers. What makes it possible is DPFP's exploitation of the the duality of the planning problem; rather than using the value iteration principle to search for policies in the value function space (Marecki, Koenig, and Tambe 2007), DPFP's search for policies is carried out in a probability function space, referred to as the dual space of cumulative distribution functions. In essence, DPFP is trying to find the best allocation of probability mass that traverses MDP states.

This dual formulation in context of continuous resource MDP is key to DPFP's novel combination of three features. First, similarly to the HAO* algorithm (Mausam et al. 2005), DPFP performs a forward search in a large (possibly infinite) policy space. Second, the cumulative distribution functions enable DPFP's novel approach to discriminate in its policy generation effort spending less time on regions of the MDP state space reachable with low-likelihood. Third, DPFP expresses its error in terms of an arbitrary small parameter κ which allows it to trade-off optimality for speed. As our experimental evaluation shows, the dual formulation translates into DPFP's superior performance: DPFP outperforms leading algorithms by up to two orders of magnitude in terms of its any-time performance and can be run in a hybrid mode with other solvers to significantly speed up the search for high quality solutions.

Background

We use time as an example of a continuous resource and deadlines as an example of resource limits and solve planning problems that are similar to the ones studied in (Feng et al. 2004; Li and Littman 2005; Marecki, Koenig, and Tambe 2007). They are modeled as MDPs with two sources of uncertainty: action durations and action outcomes. S denotes the finite set of states of the MDP, A is the finite set of actions and $A(s) \subset A$ is a set of actions that can be executed in state $s \in S$. Actions can be executed in a time interval $[0, \Delta]$ where Δ is a deadline and as such, time-to-deadline is a monotonically decreasing continuous resource. Assume that an agent is in state $s \in S$ at time $t < \Delta$. It executes an action $a \in A(s)$ of its choice. The execution of the action cannot be interrupted, and the action duration t' is distributed according to a given probability distribution $p_{s,a}(t')$ that can depend on both the state s and the action a. If $t' \geq \Delta - t$, then the time $t + t'$ after the action execution is greater or equal than Δ, which means that the deadline is reached and execution stops. Otherwise, with probability $P(s'|s, a)$, the agent obtains reward $R(s') \geq 0$ and transitions to state $s' \in S$ with time $t + t'$ and repeats the process. The agent starts in state s_0 at time 0 with the objective to maximize its expected reward until execution stops.

We use a Mars rover domain that is similar to one used in (Meuleau, Dearden, and Washington 2004; Feng et al. 2004; Li and Littman 2005; Marecki, Koenig, and Tambe 2007). A planetary rover located at the base station is presented each day with a set of M sites of interest, and it receives a one-time reward r_m upon visiting site m. Because the rover relies on solar power, it can only operate during a planetary day whose duration is Δ. As a result, the rover may not have enough time to visit all sites during one day. Moreover, since traversal times between sites are uncertain (only their probability distributions are known), rover decision which site to go next must depend on the current time.

We can solve such planning problems by encoding time in the states, resulting in a continuous state MDP that can be solved with a version of value iteration (Marecki, Koenig, and Tambe 2007) as follows: Let $V^*(s)(t)$ denote the largest expected total reward that the agent can obtain until execution stops if it starts in state $s \in S$ with time $t \in [0, \Delta]$. The agent can maximize its expected total reward by executing the action

$$\pi^*(s)(t) = \arg\max_{a \in A(s)} \{ \sum_{s' \in S} P(s'|s, a) \int_0^{\Delta - t} p_{s,a}(t')(R(s') + V^*(s')(t + t'))\, dt' \} \quad (1)$$

in state $s \in S$ at time $0 \leq t \leq \Delta$, which can be explained as follows: When it executes action $a \in A(s)$ in state $s \in S$, it incurs action duration t' with probability $p_{s,a}(t')$. If $0 \leq t' \leq \Delta - t$, then it transitions to state $s' \in S$ with probability $P(s'|s, a)$ and obtains an expected total future reward of $R(s') + V^*(s')(t + t')$.

Value iteration techniques (Feng et al. 2004; Li and Littman 2005; Marecki, Koenig, and Tambe 2007) calculate $V^*(s)(t)$ needed to determine $\pi^*(s)(t)$ by first calculating the values $V^n(s)(t)$ for all states $s \in S$, times $0 \leq t \leq \Delta$, and iterations $n \geq 0$ using the following Bellman updates:

$$V^0(s)(t) := 0$$

$$V^{n+1}(s)(t) := \begin{cases} 0 & \text{if } t \geq \Delta \\ \max_{a \in A(s)} \{ \sum_{s' \in S} P(s'|s, a) & \text{otherwise} \\ \quad \int_0^{\Delta - t} p_{s,a}(t')(R(s') + V^n(s')(t + t'))\, dt' \}. \end{cases}$$

It then holds that $\lim_{n \to \infty} V^n(s)(t) = V^*(s)(t)$ for all states $s \in S$ and $0 \leq t \leq \Delta$ and π^* can be derived from Equation (1). Note, that since t is a real-valued variable, the integral above cannot be computed exactly and hence, value iteration algorithms are only near-optimal.

DPFP Approach

Value iteration's ability to find π^* comes at a high price. Indeed, value iteration propagates values backwards, and thus, in order to find $\pi^*(s_0)(0)$, it must first find $\pi^*(s)(t)$ for all states $s \in S$ reachable from s_0 and all $t \in [0, \Delta]$ — no matter how likely it is that state s is visited at time t (in our Mars rover domain with 10 sites of interest, value iteration must plan for all 2^{10} states and for all $t \in [0, \Delta]$). In fact, value iteration does not even know the probabilities of transitioning to a state s at time t prior to finding π^*.

DPFP on the other hand, as a forward search algorithm, can determine the probabilities of transitioning to a state s at time t prior to finding π^*. Hence, DPFP can discriminate in its policy generation effort providing only approximate policies for pairs (s, t) encountered with low probability. Unfortunately if an MDP contains cycles or action duration distributions are continuous, standard forward search cannot be carried out in a standard way as it would have to consider an infinite number of candidate policies.

To remedy that, DPFP exploits two insights. First, since each action consumes a certain minimum amount of time, only a finite number of actions can be performed before the deadline (Mausam et al. 2005) and thus, the action horizon of DPFP's forward search can be finite. Second, to avoid having to consider an infinite number of policies when action duration distributions are continuous, DPFP operates on a different search space referred to as the dual space of cumulative distribution functions. In that dual space, DPFP only finds approximate solutions, yet it can express the error of its approximations in terms of an arbitrary small parameter κ. We now explain this process.

Dual Problem

There exists an alternative technique for finding a *deterministic* policy π^* that does not use Equation 1, and thus, does not calculate the values $V^*(s)(t)$ for all $s \in S$ and $t \in [0, \Delta]$. Let $\phi = (s_0, ..., s)$ be an execution path that starts in state s_0 at time t_0 and finishes in state s. $\Phi(s_0)$ is a set of all paths reachable from state s_0. Also, let $F^*(\phi)(t)$ be the probability of completing the traversal of path ϕ *before* time t when following the optimal policy π^*, and $F^*(\phi, a)(t)$

be the probability of completing the traversal of path ϕ and starting the execution of action $a \in A(s)$ *before* time t when following policy π^* — both $F^*(\phi)$ and $F^*(\phi, a)$ are cumulative distribution functions over $t \in [0, \Delta]$. In this context, the optimal *deterministic* policy $\pi^*(s)$ for state s can be calculated as follows:

$$\pi^*(s)(t) = \arg\max_{a \in A(s)} \{\lim_{\epsilon \to 0} F^*(\phi, a)(t + \epsilon) - F^*(\phi)(t)\} \quad (2)$$

Since the derivative of $F^*(\phi, a)$ over time is positive at time t for only one action $a \in A(s)$. The solution F^* to the dual problem is then a set $F^* := \{F^*(\phi); F^*(\phi, a)$ for all $\phi = (s_0, ..., s) \in \Phi(s_0)$ and $a \in A(s)\}$.

We now show how to find F^*. For notational convenience, assume $t_0 = 0$. Since rewards $R(s)$ are earned upon entering states $s \in S$ before time Δ, the expected utility $V^\pi(s_0)(0)$ of a policy π is given by:

$$V^\pi(s_0)(0) = \sum_{\phi=(s_0,...,s) \in \Phi(s_0)} F^\pi(\phi)(\Delta) \cdot R(s)$$

Where F^π differs from F^* in that F^π is associated with policy π rather than π^*. Since solution F^* must yield $V^*(s_0)(0)$, it has to satisfy:

$$\begin{aligned} V^*(s_0)(0) &= \max_\pi V^\pi(s_0)(0) = \max_\pi \sum_{\phi=(s_0,...,s) \in \Phi(s_0)} F^\pi(\phi)(\Delta) \cdot R(s) \\ &= \sum_{\phi=(s_0,...,s) \in \Phi(s_0)} F^*(\phi)(\Delta) \cdot R(s) \end{aligned}$$

In addition, $F^* \in X = \{F : (3), (4), (5)\}$ where:

$$F((s_0))(t) = 1 \quad (3)$$

$$F((s_0, ..., s))(t) = \sum_{a \in A(s)} F((s_0, ..., s), a)(t) \quad (4)$$

$$\begin{aligned} F((s_0, ..., s, s'))(t) &= \sum_{a \in A(s)} P(s, a, s') \\ &\quad \cdot \int_0^t F((s_0, ..., s), a)(t') \cdot p_{s,a}(t - t')dt' \end{aligned} \quad (5)$$

Constraint (3) ensures that the process starts in state s_0 at time 0. Constraint (4) can be interpreted as the conservation of probability mass flow through path $(s_0, ..., s)$; applicable only if $|A(s)| > 0$, it ensures that the cumulative distribution function $F((s_0, ..., s))$ is split into cumulative distribution functions $F((s_0, ..., s), a)$ for $a \in A(s)$. Finally, constraint (5) ensures the correct propagation of probability mass $F(s_0, ..., s, s')$ from path $(s_0, ..., s)$ to path $(s_0, ..., s, s')$. It ensures that path $(s_0, ..., s, s')$ is traversed at time t if path $(s_0, ..., s)$ is traversed at time $t' \in [0, t]$ and then, action $a \in A(s)$ takes time $t - t'$ to transition to state s'. The dual problem is then stated as: $\max \sum_{\phi=(s_0,...,s) \in \Phi(s_0)} F(\phi)(\Delta) \cdot R(s) \mid F \in X$.

Solving the Dual Problem

In general, the dual problem is extremely difficult to solve optimally because when action duration distributions are continuous or the MDP has cycles, the set X where F^* is to be found is infinite. Yet, we now show that even if action duration distributions are continuous and the MDP has cycles, the dual problem can be solved near-optimally with guarantees on solution quality. The idea of the algorithm that we propose is to restrict the search for F^* to finite number of elements in X by pruning from X the elements F that correspond to reaching regions of the state-space with very low probability. In essence, when the probability of reaching certain regions of the state-space is below a given threshold, the expected quality loss for executing suboptimal actions in these regions can be bounded, and we can tradeoff this quality loss for efficiency.

More specifically, our algorithm searches for F^* in set $\widehat{X} \subset X$ where $\widehat{X}$ differs from X in that values of functions F in $\widehat{X}$ are restricted to integer multiples of a given $\kappa \in \Re^+$. Informally, κ creates a step function approximation of F. Formally, $\widehat{X} = \{F : (3), (4), (6), (7), (8)\}$ where

$$\begin{aligned} F'((s_0, ..., s, s'))(t) &= \sum_{a \in A(s)} P(s, a, s') \\ &\quad \cdot \int_0^t F((s_0, ..., s), a)(t') \cdot p_{s,a}(t - t')dt' \end{aligned} \quad (6)$$

$$F((s_0, ..., s, s'))(t) = \lfloor F'((s_0, ..., s, s'))(t)/\kappa \rfloor \cdot \kappa \quad (7)$$

$$F((s_0, ..., s), a)(t) = \kappa \cdot n \text{ where } n \in N \quad (8)$$

The *restricted dual problem* is then stated as: $\max \sum_{\phi=(s_0,...,s) \in \Phi(s_0)} F(\phi)(\Delta) \cdot R(s) \mid F \in \widehat{X}$

Note, that since $\widehat{X}$ is finite, we can solve the restricted dual problem optimally by iterating over all elements of $\widehat{X}$. We will show an algorithm that carries out this iteration and returns a policy $\widehat{\pi}^*$ that is guaranteed to be at most ϵ away from π^* where ϵ can be expressed in terms of κ. We first show our algorithm on an example, then outline the algorithm pseudo-code and finally bound its error.

Figure 1 shows our algorithm in action. Assume, $A(s_0) = \{a_1\}; A(s_1) = A(s_2) = \{a_1, a_2\}$; $A(s_3), A(s_4), A(s_5)$ is arbitrary. Also, $P(s_0, a_1, s_1) = P(s_1, a_1, s_2) = P(s_1, a_2, s_3) = P(s_2, a_1, s_4) = P(s_2, a_2, s_5) = 1$ and $\kappa = 0.2$. The algorithm iterates over all elements in $\widehat{X}$. It starts with $F((s_0))$ which is given by constraint (3), then uses constraints (4), (6) to derive $F'((s_0, s_1))$ (solid gray line for state s_1) and finally uses constraint (7) to approximate $F'((s_0, s_1))$ with a step function $F((s_0, s_1))$ (solid black line for state s_1).

At this point the algorithm knows the probability $F((s_0, s_1))(t)$ that s_1 will be visited before time t but does not know the probabilities $F((s_0, s_1), a_1)(t)$ $F((s_0, s_1), a_2)(t)$ that actions a_1 or a_2 will be started from s_1 before time t (dotted black lines for state s_1). Thus, to iterate over all elements in $\widehat{X}$, it must iterate over all $|A(s_1)|^{F((s_0,s_1))(\Delta)/\kappa} = 16$ different sets of functions $\{F((s_0, s_1), a_1); F((s_0, s_1), a_2)\}$ (called *splittings* of $F((s_0, s_1))$). A splitting determines the policy (see Equation 2): For the specific splitting shown, action a_1 is started at times t_1, t_2, t_4 whereas action a_2 is started at time t_3 (we later show how to extrapolate this policy on $[0, \Delta]$).

At this point, the algorithm calls itself recursively.

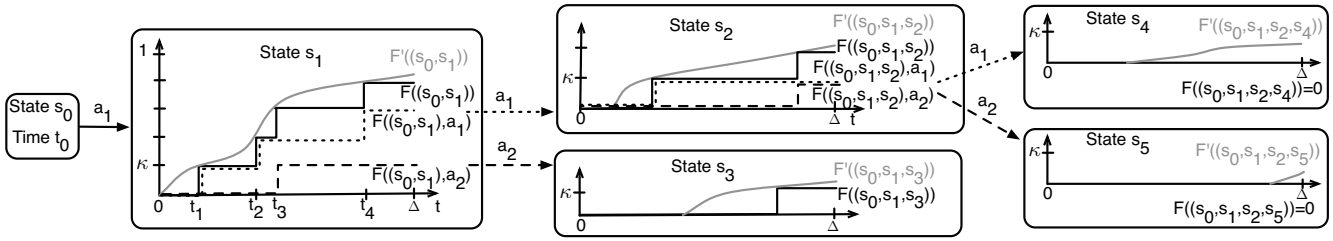

Figure 1: Search for an optimal probabilistic policy in an approximate space of cumulative distribution functions

It now knows $F((s_0, s_1, s_2))$ and $F((s_0, s_1, s_3))$ (derived from $\{F((s_0, s_1), a_1); F((s_0, s_1), a_2)\}$ using constraints (6), (7)) but does not know the probabilities $F((s_0, s_1, s_2), a_1)(t)$; $F((s_0, s_1, s_2), a_2)(t)$ that actions a_1 or a_2 will be started from s_2 before time t. Thus, to iterate over all elements in $\widehat{X}$, it iterates over all sets $\{F((s_0, s_1, s_2), a_1); F((s_0, s_1, s_2), a_2)\}$ (splittings of $F((s_0, s_1, s_2))$) . In this case, for the specific splitting shown, $F((s_0, s_1, s_2, s_4))(\Delta) < \kappa$ and thus, no splittings of $F((s_0, s_1, s_2, s_4))$ are possible (similarly for $F((s_0, s_1, s_2, s_5))$). In such case, the algorithm stops iterating over policies for states following s_4, because the maximum reward loss for not planning for these states, bounded by $\kappa \cdot R$ where $R = \sum_{\phi=(s_0,s_1,s_2,s_4,...,s)} R(s)$, can be made arbitrary small by choosing a sufficiently small κ (to be shown later). Thus, the algorithm evaluates the current splitting of $F((s_0, s_1, s_2))$ and continues iterating over remaining splittings of $F((s_0, s_1, s_2))$ after which it backtracks and picks another splitting of $F((s_0, s_1))$ etc.

Algorithm 1 DPFP$(\phi = (s_0, ..., s), F'(\phi))$

1: $F(\phi)(t) \leftarrow \lfloor F'(\phi)(t)/\kappa \rfloor \cdot \kappa$
2: $u^* \leftarrow 0$
3: **for all** sets $\{F(\phi, a) : a \in A(s)\}$ such that $F(\phi)(t) = \sum_{a \in A(s)} F(\phi, a)(t)$ and $F(\phi, a)(t) = \kappa \cdot n; n \in N$ **do**
4: $\quad u \leftarrow 0$
5: $\quad$ **for all** $s' \in S$ **do**
6: $\quad\quad F' \leftarrow \sum_{a \in A(s)} P(s, a, s') \int_0^t F(\phi, a)(t') p_{s,a}(t - t') dt'$
7: $\quad\quad u \leftarrow u +$ **DPFP**$((s_0, ..., s, s'), F')$
8: $\quad$ **if** $u > u^*$ **then**
9: $\quad\quad$ BESTSPLITTING $\leftarrow \{F(\phi, a) : a \in A(s)\}$
10: $\quad\quad u^* \leftarrow u$
11: **for all** $F(\phi, a) \in$ BESTSPLITTING **and all** $t \in [0, \Delta]$ **do**
12: $\quad$ **if** $\lim_{\epsilon \to 0} F(\phi, a)(t + \epsilon) - F(\phi, a)(t) > 0$ **then**
13: $\quad\quad \widehat{\pi}^*(s)(t) \leftarrow a$
14: **return** $u^* + F(\phi)(\Delta) \cdot R(s)$

In general, we start the algorithm by calling DPFP$((s_0), 1)$ for a globally defined and arbitrary small κ. When called for some $\phi = (s_0, ..., s)$ and $F'(\phi)$, the DPFP function first derives $F(\phi)$ from $F'(\phi)$ using constraint (7). It then iterates over all sets of functions $\{F(\phi, a) : a \in A(s)\}$ in order to find the best splitting of $F(\phi)$ *(lines 3—10)*. For a particular splitting, the DPFP function first makes sure that this splitting satisfies constraints (4) and (8) *(line 3)* upon which it calculates the total expected utility u of this splitting *(lines 4—7)*. To this end, for all paths $(s_0, ..., s, s')$, it uses constraint (6) to create functions $F' = F'((s_0, ..., s, s'))$ *(line 6)* and then, calls itself recursively for each pair $((s_0, ..., s, s'), F')$ *(line 10)*. Finally, if u is greater than the total expected utility u^* of the best splitting analyzed so far, DPFP updates the BESTSPLITTING *(lines 8—10)*.

Upon finding the BESTSPLITTING, the DPFP function uses Equation (2) to extract the best *deterministic* policy $\widehat{\pi}^*$ from it *(lines 11—13)* and terminates returning u^* plus the expected reward for entering s before time Δ (computed in line 14 by multiplying the immediate reward $R(s)$ by the probability $F(s_0, ..., s)(\Delta)$ of entering s before time Δ). As soon as DPFP$((s_0), 1)$ terminates, the algorithm extrapolates its already known point-based policies onto time interval $[0, \Delta]$ using the following method: If $\widehat{\pi}^*(s)(t_1) = a_1$, $\widehat{\pi}^*(s)(t_2) = a_2$, and $\widehat{\pi}^*(s)(t)$ is not defined for $t \in (t_1, t_2)$, the algorithm puts $\widehat{\pi}^*(s)(t) = a_1$ for all $t \in (t_1, t_2)$. For example, if splitting in Figure 1 is optimal, $\widehat{\pi}^*(s_1)(t) = a_1$ for $t \in [0, t_1) \cup [t_1, t_2) \cup [t_2, t_3) \cup [t_4, \Delta)$ and $\widehat{\pi}^*(s_1)(t) = a_2$ for $t \in [t_3, t_4)$.

Taming the Algorithm Complexity

As stated, the DPFP algorithm can appear to be inefficient since it operates on large number of paths (exponential in the length of the longest path) and large number of splittings per path (exponential in $\lfloor 1/\kappa \rfloor$). However, this exponential complexity is alleviated thanks to the following features of DPFP:

- Varying policy expressivity for different states: The smaller the probability of traversing a path $\phi = (s_0, ..., s)$ before the deadline, the less expressive the policy for state s has to be (fewer ways in which $F(\phi)$ can be split into $\{F(\phi, a) : a \in A(s)\}$). For example, state s_2 in Figure 1 is less likely to be visited than state s_1 and therefore, DPFP allows for higher policy expressivity for state s_1 (2^4 policies) than for state s_2 (2^2 policies). Sparing the policy generation effort in less likely to be visited states enables faster policy generation.
- Varying policy expressivity for different time intervals: The smaller the probability of traversing to a state inside a time interval, the less expressive the policy for this state and interval has to be. In Figure 1 we are more likely to transition to state s_1 at time $t \in [t_1, t_3]$ (with probability 2κ) than at time $t \in [t_3, t_4]$ (with probability 1κ) and

thus, DPFP considers 2^2 policies for time interval $[t_1, t_3]$ and only 2^1 policies for time interval $[t_3, t_4]$.

- Path independence: When function $F(\phi)$ for a sequence $\phi = (s_0, ...s)$ is split into functions $\{F(\phi, a) : a \in A(s)\}$, functions $\{F(\phi, a) : a \in A(s); P(s, a, s') = 0\}$ have no impact on $F((s_0, ..., s, s'))$. Thus, we have to consider fewer splittings of $F(\phi)$ to determine all possible functions $F((s_0, ..., s, s'))$. For example, in Figure 1, $F((s_0, s_1, s_2))$ is only affected by $F((s_0, s_1), a_1)$. Consequently, as long as $F((s_0, s_1), a_1)$ remains unaltered when iterating over different splittings of $F((s_0, s_1))$, we do not have to recompute the best splittings of $F((s_0, s_1, s_2))$.
- Path equivalence. For *different* paths $\phi = (s_0, ..., s)$ and $\phi' = (s_0, ..., s)$ that coalesce in state s, the best splitting of $F(\phi)$ can be reused to split $F(\phi')$ provided that $\max_{t\in[0,\Delta]} |F'(\phi)(t) - F'(\phi')(t)| \leq \kappa$.

Error Control

Recall that $\widehat{F}^*$ is the optimal solution to the *restricted* dual problem returned by DPFP. We now prove that the reward error ϵ of a policy identified by $\widehat{F}^*$ can be expressed in terms of κ. To this end, we first prove that for all paths $\phi \in \Phi(s_0)$:

$$\max_{t\in[0,\Delta]} |F^*(\phi)(t) - \widehat{F}^*(\phi)(t)| \leq \kappa|\phi| \quad (9)$$

By induction on the length of ϕ: if $|\phi| = 1$, $\max_{t\in[0,\Delta]} |F^*((s_0))(t) - \widehat{F}^*((s_0))(t)| = \max_{t\in[0,\Delta]} |1 - 1| = 0 < \kappa$. Assume now that statement (9) holds for a sequence $\phi = (s_0, ..., s_{n-1})$ of length n. Statement (9) then also holds for all sequences $\phi' = (s_0, ..., s_{n-1}, s_n)$ of length $n + 1$ because:

$$|F^*(\phi')(t) - \widehat{F}^*(\phi')(t)| \leq |F^*(\phi')(t) - \widehat{F'}^*(\phi')(t)| + \kappa$$

Where $\widehat{F}^*(\phi')$ is derived from $\widehat{F'}^*(\phi')$ using constraint (7)

$$= \sum_{a\in A(s)} P(s, a, s') \Big| \int_0^t F^*(\phi, a)(t') \cdot p_{s,a}(t - t')dt' - \int_0^t \widehat{F}^*(\phi, a)(t') \cdot p_{s,a}(t - t')dt' \Big| + \kappa$$

$$\leq \max_{a\in A(s)} \int_0^t |F^*(\phi, a)(t') - \widehat{F}^*(\phi, a)(t')| \cdot p_{s,a}(t - t')dt' + \kappa$$

$$\leq \max_{a\in A(s)} \int_0^t |F^*(\phi)(t') - \widehat{F}^*(\phi)(t')| \cdot p_{s,a}(t - t')dt' + \kappa$$

And from the induction assumption

$$\leq \int_0^t \kappa n \cdot p_{s,a}(t - t')dt' + \kappa \leq \kappa n + \kappa \leq \kappa|\phi'|$$

holds for $t \in [0, \Delta]$ which proves the induction. We now use this result to bound the error ϵ of DPFP:

$$\epsilon = R_{max} \sum_{\phi\in\Phi(s_0)} \max_{t\in[0,\Delta]} |F^*(\phi)(t) - \widehat{F}^*(\phi)(t)|$$

$$\leq \kappa R_{max} \sum_{\phi\in\Phi(s_0)} |\phi| \leq \kappa R_{max} H|A|^H$$

Where $R_{max} = \max_{s\in S} R(s)$ and H is the action horizon (if the minimal action duration δ is known than $H \leq \lfloor\Delta/\delta\rfloor$). Hence, by decreasing κ, DPFP can trade off speed for optimality.

Experimental Evaluation

We compared DPFP with leading near-optimal algorithms for solving continuous resource MDPs: Lazy Approximation (Li and Littman 2005) and CPH (Marecki, Koenig, and Tambe 2007). All algorithms were run on three different domains: Fully ordered domain, unordered domain and partially ordered domain (Figures 2a, 2b, 2c). In the fully ordered domain the agent executes an action $a' \in A(s_0) = \{a_1, a_2, a_3\}$, transitions to state $s_{a'}$, executes $a'' \in A(s_{a'}) = \{a_1, a_2, a_3\}$, transitions to state $s_{a',a''}$ — it repeats this scheme up to $H = 8$ times for a total number of $3^8 = 6561$ states. In the unordered domain (Mars rover domain introduced earlier) the agent visits up to 8 sites in an arbitrary order and hence, the number of states is $2^8 = 256$. In the partially ordered domain the agent can visit up to 10 sites in a partial order (site $m + 1$ can be visited only after site m has already been visited; $m = 1, 3, 5, 7, 9$). For all domains, we set $\Delta = 10$, draw action rewards uniformly from set $\{1, 2, ..., 10\}$ and sample action durations from one of the following probability distribution functions (chosen at random): *Normal*($\mu = 2$,$\sigma = 1$), *Weibull*($\alpha = 2$,$\beta = 1$), *Exponential*($\lambda = 2$) and *Uniform* ($a = 0$,$b = 4$).

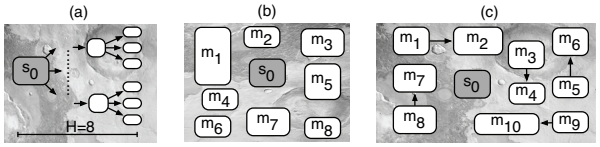

Figure 2: Experimental domains: m_i denote sites of interest

In our first set of experiments we determined how the three algorithms trade off between runtime and error. To this end we varied κ for DPFP, corresponding accuracy parameter for Lazy Approximation and the number of phases (for Phase-type approximation) for CPH. We show our results in Figures 3a, 3b, 3c where runtime (in seconds) is on the x-axis (notice the log scale) and the solution quality (% of the optimal solution) is on the y-axis. The results across all the domains show that DPFP opens up an entirely new area of the solution-quality vs time tradeoff space that was inaccessible to previous algorithms. In particular DPFP dominates Lazy approximiation in this tradeoff, providing higher quality in lower time. DPFP also provides very high quality an order of magnitude faster than CPH, e.g. in Figure 3a for solutions with quality higher than 70%, DPFP will provide an answer in 0.46 seconds, while CPH will take 28.1s for the same task. Finally, DPFP exhibits superior anytime performance, e.g. in Figure 3c, run with $\kappa = 0.3, 0, 25, 0.2$ it attains solution qualities $42\%, 61\%, 72\%$ in just $0.5s, 1.1s, 3.9s$.

Encouraged by DPFP's any-time performance and CPH's superior quality results we then developed a DPFP-CPH hybrid and compared it with a stand-alone CPH. The DPFP-CPH hybrid first uses DPFP to quickly find initial actions to be executed from s_0 and then uses these actions to narrow down CPH's search for a high quality policy. For example, the hybrid we tested uses DPFP (with $\kappa = .2$) to suggest to CPH which action should be executed in s_0 at time 0, and then runs CPH in the narrowed state-space.

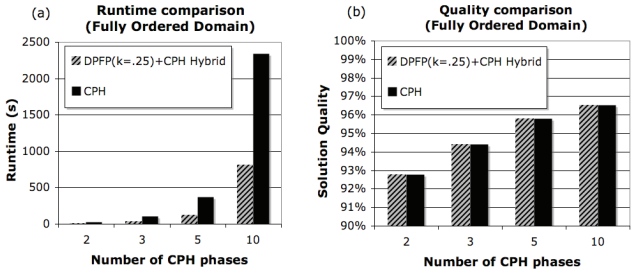

Figure 4: DPFP+CPH hybrid: Fully ordered domain

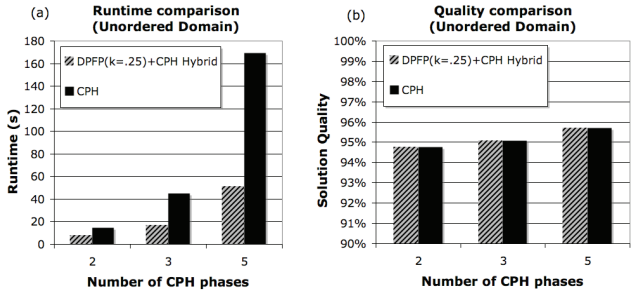

Figure 5: DPFP+CPH hybrid: Unordered domain

Our results are shown in Figures 4, 5 and 6 where on the x-axes we vary the accuracy of CPH (more phases = higher accuracy) and on the y-axes we plot the algorithms' runtime (Figures 4a, 5a, 6a) and solution quality (Figures 4b, 5b, 6b). The results across all the domains show that the DPFP-CPH hybrid attains the same quality as stand-alone CPH, yet requires significantly less runtime (over 3x). For example, when CPH accuracy is fixed at 5 phases, DPFP-CPH hybrid needs only $51s$ to find a solution whereas stand-alone CPH needs $169.5s$ for the same task (Figure 5a).

Related Work

Planning with continuous resources and resource limits has received a lot of attention. (Altman 1999; Dolgov and Durfee 2005) cast such planning problems as constrained MDPs and use linear programming to solve them efficiently. Unfortunately this formulation does not allow the policies to be conditioned on resource levels. This limitation does not apply to policy iteration algorithms (Lagoudakis and Parr 2003; Nikovski and Brand 2003; Guestrin, Hauskrecht, and Kveton 2004; Hauskrecht and Kveton 2004) or value iteration algorithms (Feng et al. 2004; Li and Littman 2005; Marecki, Koenig, and Tambe 2007) which have been successfully applied to many challenging planning problems (Bresina et al. 2002; Meuleau, Dearden, and Washington 2004). Unfortunately, these techniques can still be inefficient when the discrete component of states is large, causing a combinatorial explosion of the state-space.

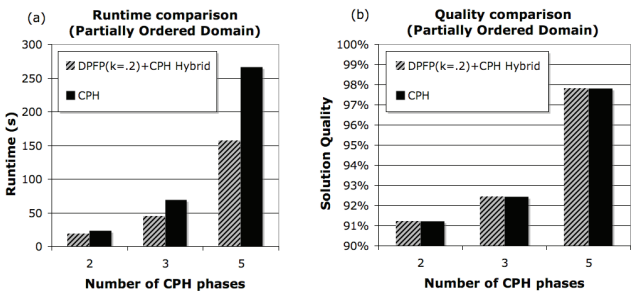

Figure 6: DPFP+CPH hybrid: Partially ordered domain

Several promising techniques have been proposed to alleviate this combinatorial explosion. In particular, (Boutilier, Dearden, and Goldszmidt 2000) suggests using dynamic Bayesian Networks to factor the discrete component of the states when discrete state variables depend on each other. Standard versions of dynamic programming that operate directly on the factored representation can then be used to solve the underlying planning problems. DPFP's contributions are complementary to this work and possible integration of DPFP and factored representations is a worthy topic for future investigation.

More recently, (Mausam et al. 2005) have developed a Hybrid AO* (HAO*) algorithm that significantly improves the efficiency of continuous resource MDP solvers. The idea in HAO* is to prioritize node expansion order based on the heuristic estimate of the node value. Furthermore, HAO* is particularly useful when the starting state, minimal resource consumption and resource constraints are given — HAO* can then prune infeasible trajectories and reduce the number of states to be considered to find an optimal policy.

Thus, at a high level there are some apparent similarities between DPFP and HAO*. However, DPFP differs from HAO* in significant ways. The most important difference is that DPFP's forward search is conducted in a dual space of cumulative distribution functions — in particular, DPFP's key novelty is its search of the different splittings of the cumulative distribution functions entering a particular state (e.g. see the splitting of $F((s_0, s_1))$ in Figure 1). This difference leads to a novel approach in DPFP's allocation of effort in determining a policy — less effort is spent on regions of the state space reachable with lower likelihood (e.g. in Figure 1 more effort is spent on time interval $[t_1, t_3]$ than on time interval $[t_3, t_4]$). While this effort allocation idea in DPFP differs from HAO*'s reachability analysis, comparing the runtime efficiency of DPFP and HAO* remains an exciting issue for future work. Such a comparison may potentially lead to creation of a hybrid algorithm combining these two approaches.

Finally, algorithms for efficient propagation of probabilities in continuous state-spaces have been investigated in the past, but not in the specific context of continuous resource MDPs. In particular, (Ng, Parr, and Koller 2000) exploits approximate probability density propagation in context of gradient descent algorithms for searching a space of MDP and POMDP stochastic controllers. For Distributed POMDPs, (Varakantham et al. 2006) use Lagrangian methods to efficiently calculate the admissible probability ranges, to speed up the search for policies. In contrast, DPFP's tailoring to continuous resource MDPs allows it to exploit the underlying dual space of cumulative distribution functions.

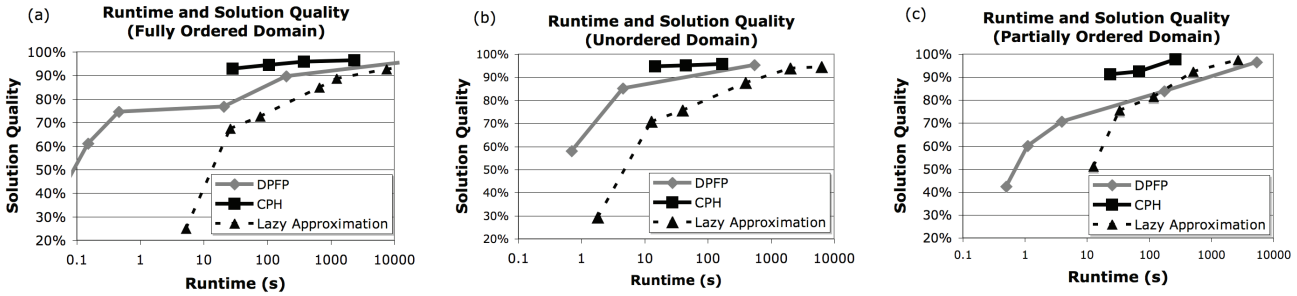

Figure 3: Comparison of runtimes and solution qualities for DPFP, CPH and Lazy Approximation

Summary

In many real-world domains agents have to construct plans that obey resources limits for continuous resources such as time or energy whose consumption is non-deterministic. For modeling such domains, MDPs with a state space of continuous and discrete variables have emerged as a popular planning framework. However, solving such MDP is a challenge and as has been shown (Meuleau, Dearden, and Washington 2004), existing policy and value iteration algorithms may exhibit poor performance with a scale-up in their state space. In this paper we proposed an algorithm called DPFP which begins to alleviate this challenge: (i) DPFP has been demonstrated to outperform leading algorithms by up to two orders of magnitude in terms of its any-time performance and (ii) it significantly speeds up the search for high quality solutions when run in a hybrid mode with a leading continuous resource MDP solver. These results are a consequence of DPFP's main contribution of exploiting the dual space of cumulative distribution functions. In essence, dual formulation allows for novel combination of (i) forward search in a large (possibly infinite) policy space, (ii) varying policy generation effort based on the likelihood of reaching different regions of the MDP state space and (iii) error bound expressed in terms of an arbitrary small parameter κ which allows for a systematic trade-off of DPFP optimality for speed.

Acknowledgments

This research is supported by the United States Department of Homeland Security through Center for Risk and Economic Analysis of Terrorism Events (CREATE). The authors also want to thank the anonymous reviewers for their valuable comments.

References

Altman, E. 1999. *Constrained Markov Decision Processes*. Chapman and Hall.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1-2):49–107.

Bresina, J.; Dearden, R.; Meuleau, N.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of UAI-02*, 77–84.

Dolgov, D. A., and Durfee, E. H. 2005. Stationary deterministic policies for constrained MDPs with multiple rewards, costs, and discount factors. In *Proceedings of IJCAI-05*, 1326–1332.

Feng, Z.; Dearden, R.; Meuleau, N.; and Washington, R. 2004. Dynamic programming for structured continuous MDPs. In *Proceedings of UAI-04*, 154–161.

Guestrin, C.; Hauskrecht, M.; and Kveton, B. 2004. Solving factored MDPs with continuous and discrete variables. In *Proceedings of UAI-04*, 235–242.

Hauskrecht, M., and Kveton, B. 2004. Linear program approximations for factored continuous-state MDPs. In *NIPS 16*. MIT Press.

Lagoudakis, M., and Parr, R. 2003. Least-squares policy iteration. In *JMLR*, volume 4(Dec). 1107–1149.

Li, L., and Littman, M. 2005. Lazy approximation for solving continuous finite-horizon MDPs. In *Proceedings of AAAI-05*, 1175–1180.

Marecki, J.; Koenig, S.; and Tambe, M. 2007. A fast analytical algorithm for solving MDPs with real-valued resources. In *Proceedings of IJCAI-07*.

Mausam; Benazera, E.; Brafman, R. I.; Meuleau, N.; and Hansen, E. A. 2005. Planning with continuous resources in stochastic domains. In *Proceedings of IJCAI-05*, 1244–1251.

Meuleau, N.; Dearden, R.; and Washington, R. 2004. Scaling up decision theoretic planning to planetary rover problems. In *Proceedings of AAAI-04: WS-04-08*, 66–71.

Ng, A. Y.; Parr, R.; and Koller, D. 2000. Policy search via density estimation. In *Advances in Neural Information Processing Systems*, 1022–1028.

Nikovski, D., and Brand, M. 2003. Non-Linear stochastic control in continuous state spaces by exact integration in Bellman's equations. In *Proceedings of ICAPS-03: WS2*, 91–95.

Varakantham, P.; Nair, R.; Tambe, M.; and Yokoo, M. 2006. Winning back the cup for distributed POMDPs: Planning over continuous belief spaces. In *Proceedings of AAMAS-06*, 289–296.