Towards Faster Planning with Continuous Resources in Stochastic Domains

Janusz Marecki  
University of Southern California, marecki@usc.edu

Milind Tambe  
University of Southern California, tambe@usc.edu

Follow this and additional works at: http://research.create.usc.edu/published_papers

Recommended Citation
http://research.create.usc.edu/published_papers/40
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; Meuleau, Dearden, and Washington 2005) 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 Dynamic Probability Function Propagation (DPFP) algorithm that enables scale-up of continuous resource MDP solvers. What makes it possible is DPFP’s exploitation of 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 $\kappa$ 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.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
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 \( \Delta \) 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 \( \Delta \), 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 \( \Delta \). 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

\[
\pi^*(s)(t) = \arg \max_{a \in A(s)} \left\{ \sum_{s' \in S} P(s'|s,a) \cdot \int_0^{\Delta - t} p_{s,a}(t') (R(s') + V^*(s')(t + t')) \, dt' \right\}
\]

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) calculated \( V^*(s)(t) \) 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:

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

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 \( \pi^* \) 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 \( \pi^* \) 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 \( t \in [0, \Delta] \). Unfortunately if an MDP contains cycles or action duration distributions, in that dual space, 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 \( \kappa \). We now explain this process.

Dual Problem

There exists an alternative technique for finding a deterministic policy \( \pi^* \) 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 \( \phi \) before time \( t \) when following the optimal policy \( \pi^* \), and \( \Phi^*(\phi,a)(t) \)
be the execution of completing the traversal of path $\phi$ and starting the execution of action $a \in A(s)$ before time $t$ when following policy $\pi^*$—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 $\phi = (s_0, \ldots, s), \Phi(s_0)$ 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 $\Delta$, the expected utility $V^\pi(s_0)(0)$ of a policy $\pi$ is given by:

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

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

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

$$= \sum_{\phi=(s_0, \ldots, s) \in \Phi(s_0)} F^*(\phi)(\Delta) \cdot R(s)$$

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

$$F((s_0, \ldots, s))(t) = 1 \quad (3)$$

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

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

Constraint (3) ensures that the process starts in state $s_0$ at time 0. Constraint (4) can be interpreted as the expected probability of probability mass flow through path $(s_0, \ldots, s)$; applicable only if $|A(s)| > 0$, it ensures that the cumulative distribution function $F((s_0, \ldots, s), a)$ is split into cumulative distribution functions $F((s_0, \ldots, s), a)$ for $a \in A(s)$. Finally, constraint (5) ensures the correct propagation of probability mass $F((s_0, \ldots, s, s'))$ from path $(s_0, \ldots, s)$ to path $(s_0, \ldots, s')$. It ensures that path $(s_0, \ldots, s, s')$ is traversed at time $t$ if path $(s_0, \ldots, s)$ 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:

$$\max \sum_{\phi=(s_0, \ldots, s) \in \Phi(s_0)} F(\phi)(\Delta) \cdot R(s) \mid F \in X.$$

### 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 $\tilde{X} \subset X$ where $\tilde{X}$ differs from $X$ in that values of functions $F$ in $\tilde{X}$ are restricted to integer multiples of a given $\kappa \in \mathbb{Z}^+$. Informally, $\kappa$ creates a step function approximation of $F$. Formally, $\tilde{X} = \{F: (3), (4), (6), (7), (8)\}$ where

$$F'((s_0, \ldots, s, s'))(t) = \sum_{a \in A(s)} P(s, a, s')$$

$$\cdot \int_0^t F((s_0, \ldots, s), a)(t') \cdot p_{s, a}(t - t') dt'$$

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

$$F((s_0, \ldots, s), a)(t) = \kappa \cdot n \mid n \in N$$

The restricted dual problem is then stated as:

$$\max \sum_{\phi=(s_0, \ldots, s) \in \Phi(s_0)} F(\phi)(\Delta) \cdot R(s) \mid F \in \tilde{X}.$$

Note, that since $\tilde{X}$ is finite, we can solve the restricted dual problem optimally by iterating over all elements of $\tilde{X}$. We will show an algorithm that carries out this iteration and returns a policy $\tilde{\pi}^*$ that is guaranteed to be at most $\epsilon$ away from $\pi^*$ where $\epsilon$ can be expressed in terms of $\kappa$. 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 $\tilde{X}$. It starts with $F((s_0, s_1))$ 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 $\tilde{X}$, 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.
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 \( X \), 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 \( F((s_0, s_1, s_2, s_3)) \)). 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 \( \kappa \cdot \pi \cdot R \) where \( R = \sum_{\phi=(s_0, s_1, s_2, s_3 \ldots s)} R(s) \), can be made arbitrary small by choosing a sufficiently small \( \kappa \) (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 DFPFP(\( \phi = (s_0, \ldots, s) \), \( F'(\phi) \))

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

In general, we start the algorithm by calling DFPFP(\( s_0, 1 \)) for a globally defined and arbitrary small \( \kappa \). When called for some \( \phi = (s_0, \ldots, s) \) and \( F'(\phi) \), the DFPFP function first derives \( F'(\phi) \) from \( F'(\phi)(t) \) 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—7). To this end, for all paths \( (s_0, \ldots, s', s') \), it uses constraint (6) to create functions \( F' = F'(s_0, \ldots, s') \) (line 6) and then, calls itself recursively for each pair \( (s_0, \ldots, s', s') \), \( F' \) (line 10). Finally, if \( u \) is greater than the total expected utility \( u^* \) of the best splitting analyzed so far, DFPFP updates the BESTSPLITTING (lines 8—10).

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

Taming the Algorithm Complexity

As stated, the DFPFP algorithm can appear to 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 \( |A(s)| \)). However, this exponential complexity is alleviated thanks to the following features of DFPFP:

- **Varying policy expressivity for different states:** The smaller the probability of traversing a path \( \phi = (s_0, \ldots, s) \) 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, DFPFP allows for higher policy expressivity for state \( s_1 \) (2^\( |A(s)| \) policies) than for state \( s_2 \) (2^\( |A(s)| \) 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 \( t \in [t_1, t_2] \) (with probability 2\( \kappa \)) than at time \( t \in [t_3, t_4] \) (with probability 1\( \kappa \)).
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 $\phi = (s_0, \ldots, s)$ is split into functions $\{F(a, \phi) : a \in A(s)\}$ and $F(s, a, s') = 0$, then functions $\{F^*(a, \phi) : a \in A(s)\}$ have no impact on $F^*((s_0, \ldots, s, s'))$. Thus, we have to consider fewer splittings of $F(\phi)$ to determine all possible functions $F^*((s_0, \ldots, s, s'))$. 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 $F^*((s_0, s_1), a_1)$ unremains unaltered when iterating over different splittings of $F^*((s_0, s_1))$, we do not have to recompute the best splittings of $F^*((s_0, s_1), a_1)$.

- Path equivalence. For different paths $\phi = (s_0, \ldots, s)$ and $\phi' = (s_0, \ldots, s')$ 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 $\hat{F}^*$ is the optimal solution to the restricted dual problem returned by DPFP. We now prove that the reward error $\epsilon$ of a policy identified by $\hat{F}^*$ can be expressed in terms of $\kappa$. To this end, we first prove that for all paths $\phi \in \Phi(s_0)$:

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

(9)

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

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

Where $\hat{F}^*(\phi')(t)$ is derived from $\hat{F}^*(\phi')$ using constraint (7):

$$\max_{s \in A(\sigma)} \left\{ \sum_{a \in A(s)} P(s, a, s') \int_0^t F^*(\phi, a)(t') \cdot p_{s,a}(t - t') dt' \right\}$$

$$- \int_0^t \hat{F}^*(\phi, a)(t') \cdot p_{s,a}(t - t') dt' + \kappa$$

$$\leq \max_{s \in A(\sigma)} \left\{ \sum_{a \in A(s)} P(s, a, s') \int_0^t F^*(\phi, a)(t') \cdot p_{s,a}(t - t') dt' \right\} + \kappa$$

$$\leq \max_{s \in A(\sigma)} \left\{ \sum_{a \in A(s)} P(s, a, s') \int_0^t F^*(\phi, a)(t') \cdot p_{s,a}(t - t') dt' \right\} + \kappa$$

And from the induction assumption

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

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

$$\epsilon = R_{\max} \max_{s \in A(s)} \left\{ \sum_{a \in A(s)} P(s, a, s') \int_0^t F^*(\phi, a)(t') \cdot p_{s,a}(t - t') dt' \right\}$$

$$\leq \kappa R_{\max} \sum_{s \in A(s)} \left| \phi \right| \leq \kappa R_{\max} H |A|^d$$

Where $R_{\max} = \max_{s \in A(s)} R(s)$ and $H$ is the action horizon (if the minimal action duration $\delta$ is known than $H \leq \lfloor \Delta / \delta \rfloor$). Hence, by decreasing $\kappa$, 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 $a' \in [0, a_2, a_3]$, transitions to state $s_{a'}$, executes $a'' \in [0, a_2, a_3]$, 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 $m$ can be visited only after site $m+1$ has already been visited; $m = 1, 3, 5, 7, 9$). For all domains, we set $\Delta = 10$, draw action rewards uniformly from set $\{1, 2, \ldots, 10\}$ 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 ($a = 0, b = 4$).
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 51 s to find a solution whereas stand-alone CPH needs 169.5 s 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.

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.
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 $\kappa$ 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


