A Fast Analytical Algorithm for MDPs with Continuous State Spaces

Janusz Marecki, Zvi Topol and Milind Tambe
Computer Science Department
University of Southern California
Los Angeles, CA 90089
{marecki, topol, tambe}@usc.edu

ABSTRACT

Many real-world domains require that agents plan their future actions despite uncertainty, and that such plans deal with continuous space states, i.e. states with continuous values. While finite-horizon continuous state MDPs enable agents to address such domains, finding an optimal policy is computationally expensive. Although previous work provided approximation techniques to reduce the computational burden (particularly in the convolution process for finding optimal policies), computational costs and error incurred remain high. In contrast, we propose a new method, *CPH*, to solve continuous state MDPs for both finite and infinite horizons. *CPH* provides a fast analytical solution to the convolution process and assumes that continuous state values change according to Phase-type distributions. This assumption allows our method to approximate arbitrary probability density functions of continuous state transitions. Our experiments show that *CPH* achieves significant speedups compared to the Lazy Approximation algorithm, which is the leading algorithm for solving continuous state MDPs.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Intelligent Agents—*decision theory*

General Terms

Algorithms

Keywords

Continuous State Markov Decision Process, Convolution, Exponential Probability Distributions, Phase-Type Distributions, Planning with Limited Resources, Value Iteration

1. INTRODUCTION

Recent advances in robotics have made aerial, underwater and terrestrial unmanned autonomous vehicles possible [1]. Many domains for which such unmanned vehicles are constructed are uncertain and exhibit inherent continuous characteristics, such as time required to act or other continuous resources at the disposal of the vehicles, e.g. battery power. Therefore, fast construction of efficient plans for agents acting in such domains characterized by constrained and continuous resources has been a major challenge for AI research [3].

A continuous state MDP is a natural way to represent a decision process over continuous space of states, i.e. space for which variables may take on continuous values. Unfortunately, generating optimal policies tractably for such MDPs and addressing their continuous states is a challenge. Three main classes of solutions have been offered to address this challenge: *discretization, uniformization*, and *discretization-free approximation*. The first approach discretizes the continuous state and solves the resulting discrete MDP using standard MDP solvers. Unfortunately, such discretization can lead to an exponential blow-up in the number of states. Another approach of this type discretizes the transition function. For instance, time-dependent MDP or TMDPs [2] include a continuous time dimension in the state space, but assume that the transition probability distribution function is discrete. The authors provide a dynamic programming algorithm for TMDPs to compute an exact solution for a time-dependent value function with finite time horizon. Unfortunately, transition function discretization significantly restricts the domains where TMDPs may be applied.

Uniformization [5, 8] is a discretization-free approach for solving continuous time MDPs. Uniformized continuous time MDP can be solved using standard policy or value iteration techniques. This approach was further generalized to propose a discretization-free planning algorithm for Generalized Semi Markov Decision Processes [9]. However, these approaches do not allow the planner to keep track of the time elapsed since the beginning of plan execution. Therefore, such approaches are not well-suited for continuous state finite horizon problems, i.e. problems with deadlines or bounded resources. More recently, [6] suggested a discretization-free approximation approach called Lazy Approximation. This approach avoids the restrictions of TMDPs by allowing arbitrary transition probability distribution functions to be approximated using piecewise constant functions (PWC). During value iteration, those are convoluted with PWC rewards to produce piecewise linear functions that are then approximated using PWC functions. However, their value iteration computations remain costly.

In contrast, we propose *CPH* ("continuous (= C) state MDPs through phase-type (= PH) distributions"), a novel solution method that adds two major contributions: first, instead of discretizing the transition function or using a PWC approximation, we rely on Phase-type distributions [7], which can approximate arbitrary transition functions using Markov chains with exponential transitions. This allows CPH to be applied to problems with arbitrary probability density functions over continuous state transitions. Second, our al-

gorithm exploits properties of exponential distributions, which are the basic building blocks of Phase-type distributions, for performing very efficient Bellman updates via an analytical solution to the required convolution process. CPH achieves significant speedups over Lazy Approximation, currently the fastest method for solving continuous state MDPs.

2. ILLUSTRATIVE DOMAIN

We use a rover application to illustrate our planning problems, which is a simplified version of the one used in [3]. A rover has to maximize its reward in the next $\Delta = 4$ time units. The states of the MDP correspond to the locations of the rover: its start location ($start$), site 1 ($site_1$), site 2 ($site_2$), site 3 ($site_3$) and its base. The rover can perform two actions at each location: It can move to the next site and collect a rock probe from there. It receives rewards 4, 2 and 1 for collecting rock probes from sites 1, 2 and 3, respectively. It can also move back to its base to perform a communication task, which drains it energy completely and thus makes it impossible for it to perform additional actions. It receives reward 6 for performing the communication task. The times that it takes to move to the next site and take a rock probe or move back to base and perform the communication task are uncertain due to varying navigation times on the rough surface. They are all distributed according to an exponential probability distribution with $\lambda = 1$.

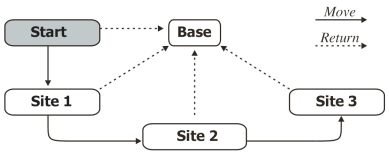

Figure 1: Simplified Mars rover domain

3. CONTINUOUS STATE MDPS

Our continuous state MDPs are similar to the ones used in [6]. For simplicity, we assume that their states have only one continuous dimension, namely action duration time, whose value decreases exponentially as actions are executed. It is important to understand that this is not a restriction since we show how to handle arbitrary action duration times using Phase-type distributions. Our model is extendible to multiple continuous components, similar to [4, 6].

3.1 Model

S denotes the finite set of discrete states of our continuous state MDPs and A their finite set of actions. Assume that the agent is in discrete state $s \in S$ with a deadline $t > 0$ time units away (= with time-to-deadline t). It then executes an action $a \in A$ of its choice, transitions with probability $P(s'|s,a)p(t')$ into discrete state $s' \in S$ with time-to-deadline $t - t'$ and incurs reward $R(s,a,s')$ if $t - t' > 0$. The execution time t' of the action is distributed according to the exponential probability density function $p(t') = \lambda e^{-\lambda t'}$. If $t' \geq t$ then the deadline is reached and execution stops. The agent's objective is to maximize its expected total reward until execution stops.

We can use a version of value iteration [2] to solve this problem. Let $V_s^*(t)$ denote the largest expected total reward that the agent can obtain until execution stops when it starts in discrete state s with time-to-deadline $t \geq 0$. The agent then achieves its objective by executing action:

$$\arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a) \int_0^t p(t')((V_{s'}^*(t-t') + R(s,a,s'))dt')$$

in discrete state s with time-to-deadline $t > 0$. Value iteration can be used to obtain the values $V_s^*(t)$. It calculates the values ${V_s}^n(t)$ using the following Bellman updates for all discrete states s and iterations n. It then holds that $\lim_{n\to\infty} V_s^n(t) = V_s^*(t)$ for all discrete states s and times-to-deadline $t \geq 0$:

$$V_s^0(t) = 0$$

$$V_s^{n+1}(t) = \begin{cases} 0 & \text{if } t \leq 0 \\ \max_{a \in A}(\sum_{s' \in S} P(s'|s,a) & \\ \quad \int_0^t p(t')(V_{s'}^n(t-t') + R(s,a,s'))dt') & \text{otherwise} \end{cases}$$

Unfortunately, value iteration cannot be implemented as stated since the number of values $V_s^n(t)$ is infinite for each n. This paper remedies this situation providing an efficiently implementable version of value iteration, that is based on four key contributions: First, we show that the *value functions* V_s^n can be represented with a small number of real numbers each. Second, we show how the Bellman updates can efficiently transform the representations of the value functions V_s^n into the representations of the value functions V_s^{n+1}. Third, we show for how many iterations n^* to run value iteration to ensure that the error $\max_{s \in S, t \geq 0} |V_s^*(t) - V_s^{n^*}(t)|$ is less than a given constant $\epsilon > 0$. Fourth, we show which actions the agent should execute according to the value functions $V_s^{n^*}$.

3.2 Notation

We define our continuous state MDP for time interval $X = (0, \Delta]$. Policy π is a mapping from $s \in S$ into set of pairs $\langle X_j^s, a_j^s \rangle$, $j = 1...I_s$ for some number of *time intervals* I_s where $\{X_j^s\}_{j=1...I_s}$ is a partitioning of X and $a_j^s \in A$. For each interval X_j^s we have some function $\Gamma_{s,j}(t)$ that models maximum expected reward for a_j^s executed in state s with t time to deadline. $\Gamma_{s,j}$ may be different for $j = 1...I_s$. Value function V_s^n is a Piecewise Gamma (PWΓ) function represented by Υ_s. Formally: $\Upsilon_s =^{def} \{\langle X_j^s, \Gamma_{s,j} \rangle\}_{j=1...I_s}$ such that $\Upsilon_s(t) = \Gamma_{s,j}(t) = V_s^n(t)$ for j s.t. $t \in X_j^s$. We write Γ_s if $I_s = 1$. A policy π^* is optimal if $\Upsilon_s^{\pi^*}(t) = V_s^*(t)$ is greater than $\Upsilon_s^{\pi'}(t)$ for any π' over all $s \in S$ and $t \in X$.

Policy execution

Actions are atomic, i.e. they are not interruptible and last until a transition to a new state or self-transition is observed. Consequently, the policy execution algorithm (i) wakes up every small amount of time to check if the currently executed action has terminated, in which case it (ii) identifies the current discrete state s and time to deadline t (iii) selects the correct pair $\langle X_j^s, a_j^s \rangle$ such that $t \in X_j^s$ and starts the execution of a_j^s.

4. ANALYTICAL SOLUTION

In this section we claim that if all action duration are governed by the same probability distribution function $p(t') = \lambda e^{-\lambda t'}$ then: (i) For each $s \in S, n \geq 0$ value function V_s^n can be expressed in Piecewise Gamma (PWΓ) form $\Upsilon_s = \{\langle X_j^s, \Gamma_{s,j} \rangle\}_{j=1...I_s}$ where: $\Gamma_{s,j}(t) = c_1^{s,j} - e^{-\lambda t}\left(c_2^{s,j} + c_3^{s,j}(\lambda t) + ... + c_{n+1}^{s,j}\frac{(\lambda t)^{n-1}}{(n-1)!}\right)$ for some real numbers $c_i^{s,j}$ $\forall_{s \in S,\ j=1,...,I_s,\ i=1,...,n}$ and λ
(ii) Each $\Gamma_{s,j}$ function can be represented by a vector $\vec{\Gamma}_{s,j}$ of real numbers and it is possible to efficiently determine $\vec{\Gamma}_{s,j}$ at iteration $n+1$ from vectors $\vec{\Gamma}_{s',j} | s' \in S$ at iteration n.

We also present a very fast Bellman update iteration for arbitrary PWΓ function and show how to extend the reasoning to action durations governed by arbitrary probability density functions (**pdf**).

4.1 Value Function

We first show PWΓ form of V_s^n for *Unconditional Policies* where action choice does not depend on current time. We then extend the reasoning to *Conditional Policies* i.e. policies where action choice depends on current time.

4.1.1 Unconditional Policies

Assume initially, that transitions between $s_1, s_2, ..., s_n$, where $\{s_1, ..., s_n\} \subset S$, are deterministic. In such a case, unconditional policy for a starting state s_1 will be a sequence of actions $\vec{a} = (a_1, a_2, ..., a_{n-1})$ moving the process from s_1 to some ending state s_n, i.e., $s_i \xrightarrow{\text{a}_\text{i}} s_{i+1}\ \forall_{i=1,2,..,n-1}$. Let r_i be the reward for a_i accumulated upon entering $s_{i+1}\ \forall_{i=1,2,..,n-1}$ before deadline. Denoting for simplicity the convolution operation by $\diamond$ we have $(p \diamond V_s)(t) = \int_0^t p(t-y)V_s(y)dy$, and thus $(p \diamond r_i)(t) = \int_0^t r_i \cdot p(t-y)dy$. We unfold Bellman equations for state s_1:

$$V_{s_1}^{n-1} = p \diamond r_1 + p \diamond V_{s_2}^{n-2} = p \diamond r_1 + p \diamond p \diamond r_2 + p \diamond p \diamond V_{s_3}^{n-3}$$
$$= \ldots = p \diamond r_1 + p \diamond p \diamond r_2 + \ldots + \underbrace{p \diamond \ldots \diamond p}_{n-1} \diamond r_{n-1}$$

Since $p \diamond \ldots \diamond p$ is similar to the *Euler's Incomplete Gamma Function*, for some constant c we have:

$$(\underbrace{p \diamond \ldots \diamond p}_{n} \diamond c)(t) = c\left(1 - e^{-\lambda t}\left(1 + \frac{(\lambda t)^1}{1!} + ... + \frac{(\lambda t)^{n-1}}{(n-1)!}\right)\right)$$

Thus $V_{s_1}^{n-1}$ can be represented in a compact way:

$$V_{s_1}^{n-1} = c_1 - e^{-\lambda t}\left(c_2 + c_3(\lambda t) + ... + c_n\frac{(\lambda t)^{n-2}}{(n-2)!}\right) = \Gamma_{s_1}$$

$$[c_1, c_2, ..., c_n] = [\textstyle\sum_{i=1}^{n-1} r_i, \sum_{i=1}^{n-1} r_i, \sum_{i=2}^{n-1} r_i, ..., \sum_{i=n-1}^{n-1} r_i]$$

Thus, $V_{s_1}^{n-1}$ is a function in PWΓ form with just one component Γ_{s_1}. In general, we can store such Γ functions in vector forms, $[c_1, c_2, ..., c_n]$, e.g. We can store Γ_{s_1} in a vector $\vec{\Gamma}_{s_1}$:

$$\vec{\Gamma}_{s_1} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \ldots \\ c_n \end{pmatrix} = \begin{pmatrix} r_1 + r_2 + ... + r_{n-1} \\ r_1 + r_2 + ... + r_{n-1} \\ r_2 + ... + r_{n-1} \\ \ldots \\ r_{n-1} \end{pmatrix}$$

For our domain, the expected reward at horizons $h = 1, 2, ...$ for returning to base from any state except "Base" is modeled by $\Gamma_s^1(t) = 6 - 6e^{-t}$ (since $\lambda = 1$) and therefore $\vec{\Gamma}_s^1 = [6, 6]$. For $h = 2$ expected reward for moving from $Site\ 2$ consists of the reward for scanning $Site\ 3$ and future reward for returning to base from $Site\ 3$, i.e, $\vec{\Gamma}_{site_2}^2 = [1, 1, 0] + [6, 6, 6] = [7, 7, 6]$.

4.1.2 Conditional Policies

Conditional policy for $s \in S$ allows us to choose a different action a_j^s depending on which interval X_j^s the remaining time t belongs to. We will show that for conditional policies, $V_s^n\ \forall_{s \in S}$ stay in PWΓ form. Let $s_0, s_1 \in S$ and $a \in A$ such that $s_0 \xrightarrow{\text{a}} s_1$. We now show how to derive $\Gamma_{s_0,i}$ functions of $p \diamond \Gamma_{s_1,i}$ for each $\langle X_j^{s_1}, \Gamma_{s_1,j}\rangle \in \Upsilon_{s_1}$. We refer to points where two adjacent Γ functions meet as *breakpoints* [2]. We now show the construction scheme for subsequent $\Gamma_{s_0,i}$. For illustrative purposes let I_{s_1}=3 and $\Upsilon_{s_1} = \{\langle[0, t_1), f\rangle, \langle[t_1, t_2), g\rangle, \langle[t_2, \Delta], h\rangle\}$ as shown in Figure 2 where x-axis represents time-to-deadline, i.e if $t \leq 0$ the policy execution terminates. Also, since $p \diamond (r_1 + V_{s_1}) = p \diamond r_1 + p \diamond V_{s_1}$ we may start with deriving $p \diamond V_{s_1}$ and later augment it by the expected reward for the first action: $p \diamond r_1$ (Section 4.2).

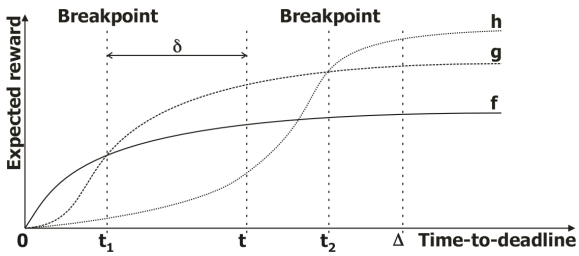

Figure 2: Value functions for state s_1

We first observe that if $t \in [0, t_1)$ then $\Gamma_{s_0,1}(t) = (p \diamond f)(t)$. This formula holds, since after taking action a in s_0 having t time left, we will end up in discrete state s_1 with $\bar{t}$ time left, where $t > \bar{t}$.

If $t \in [t_1, t_2)$ then $\Gamma_{s_0,2}$ must be greater than $p \diamond g$, because transition from s_0 to s_1 might have taken more than δ of time. In that case it is more profitable to perform an action whose expected reward is described by f (not g). The function $\Gamma_{s_0,2}(t)$ therefore breaks down into two parts: $\Gamma_{s_0,2}(t) = \alpha(t) + \beta(t)$ with $\alpha(t)$ associated with the expected reward given the transition takes less than δ of time and $\beta(t)$ in the other case.

$$\alpha(t) = \int_0^t p(t-y)g(y)dy - \int_0^{t_1} p(t-y)g(y)dy$$
$$= (p \diamond g)(t) - \int_0^{t_1} \lambda e^{-\lambda(t_1+\delta-y)}g(y)dy$$
$$= (p \diamond g)(t) - e^{-\lambda\delta}(p \diamond g)(t_1)$$

$$\beta(t) = \int_0^{t_1} p(\delta+y)f(t_1-y)dy = \int_0^{t_1} e^{-\lambda(\delta+y)}f(t_1-y)dy$$
$$= e^{-\lambda\delta}\int_0^{t_1} \lambda e^{-\lambda y}f(t_1-y)dy = e^{-\lambda\delta}(p \diamond f)(t_1)$$

$$\Gamma_{s_0,2}(t) = (p \diamond g)(t) + e^{-\lambda\delta}[(p \diamond f)(t_1) - (p \diamond g)(t_1)]$$
$$= (p \diamond g)(t) + e^{-\lambda t}\underbrace{e^{\lambda t_1}[(p \diamond f)(t_1) - (p \diamond g)(t_1)]}_{constant_1}$$

If $t \in [t_2, \Delta]$ we apply the same reasoning reusing the optimal value function for previous interval. In particular, for $\delta_2 = t - t_2$ value function for previous interval has to be decreased by a factor of $e^{-\lambda\delta_2}$, since δ_2 time has to pass before we start using $\Gamma_{s_0,2}(t)$. Consequently, finding $\beta(t)$ for the current interval involves multiplying the value of $\Gamma_{s_0,2}$ at time t_2 by $e^{-\lambda\delta_2}$:

$$\Gamma_{s_0,3}(t) = (p \diamond h)(t) + e^{-\lambda\delta_2}[\Gamma_{s_0,2}(t_2) - (p \diamond h)(t_2)]$$
$$= (p \diamond h)(t) + e^{-\lambda t}\underbrace{e^{\lambda t_2}[\Gamma_{s_0,2}(t_2) - (p \diamond h)(t_2)]}_{constant_2}$$

One can then use the same construction scheme for subsequent intervals $X_i^{s_1}$ if $I_{s_1} > 3$. Later on we refer to $constant_i$ as a *breakpoint smoothing factor* for interval i. As we have observed, given that f, g, h are in Γ form, also $\Gamma_{s_0,1}$, $\Gamma_{s_0,2}$ and $\Gamma_{s_0,3}$ are in Γ form and hence $\Upsilon_{s_0} = \{\langle[0, t_1), \Gamma_{s_0,1}\rangle, \langle[t_1, t_2), \Gamma_{s_0,2}\rangle, \langle[t_2, \Delta], \Gamma_{s_0,3}\rangle\}$ stays in PWΓ form.

For our domain, the expected reward in state $Site\ 2$ at horizon $h = 2$ consisted of two segments: $\Upsilon_{s_2} = \{\langle(0, 3.06], f\rangle\langle(3.06, 4], g\rangle\}$ where $\vec{f} = [6, 6]$ and $\vec{g} = [7, 7, 6]$. We then have $\Upsilon_{s_1} = \{\langle(0, 3.06], f'\rangle\ \langle(3.06, 4], g'\rangle\}$ where $\vec{f'} = p \diamond f = [6, 6, 6]$ and $\vec{g'} = p \diamond g + e^{-t} \cdot e^{3.06} \cdot (p \diamond f(3.06) - p \diamond g(3.06)) = [7, -4.09, 7, 6]$. Note that $\vec{f'}$ and $\vec{g'}$ have yet to be increased by vector $[2, 2]$ to account for the reward for collecting samples from site 2.

4.2 Implementing Value Iteration

We now propose a fast recursive procedure, that derives value functions using simple vector transformations. First though, we present a technique for fast convolution of an arbitrary Γ function with p.

THEOREM 1. *If $\vec{\Gamma} = [c_1, c_2, ..., c_n]$ then for function $p \diamond \Gamma$ we have the vector of coefficients $\overrightarrow{(p \diamond \Gamma)} = [c_1, c_1, c_2, ..., c_n]$*

PROOF. Let there be functions $\Gamma_1, \Gamma_2, ..., \Gamma_n$ such that:
$\vec{\Gamma}_1 = [c_n]_{k=1}^{n}$, $\vec{\Gamma}_2 = [c_{n-1} - c_n]_{k=1}^{n-1}$, $\vec{\Gamma}_3 = [c_{n-2} - c_{n-1}]_{k=1}^{n-2}$

$$\dots$$

$\vec{\Gamma}_{n-1} = [c_2 - c_3, c_2 - c_3]$; $\quad \vec{\Gamma}_n = [c_1 - c_2]$
Observe that $\Gamma = \Gamma_1 + \Gamma_2 + ... + \Gamma_n$. Since convolution is distributive: $p \diamond \Gamma = p \diamond (\Gamma_1 + ... + \Gamma_n) = p \diamond \Gamma_1 + ... + p \diamond \Gamma_n$. Because $\overrightarrow{(p \diamond \Gamma_1)} = [c_n]_{k=1}^{n+1}$ and $\overrightarrow{(p \diamond \Gamma_i)} = [c_{n-i+1} - c_{n-i+2}]_{k=1}^{n-i+1}$ we get $\overrightarrow{(p \diamond \Gamma)} = [c_1, c_1, c_2, ..., c_n]$. $\square$

Bellman update iteration for PWΓ value functions and deterministic discrete state-to-state transitions has the form: $V_s^n = p \diamond (r + V_{\bar{s}}^{n+1})$ where r is the immediate reward upon entering $\bar{s}$. Given $V_{\bar{s}}^n$ is composed of segments $\Gamma_{\bar{s},i} \mid_{i \in I_{\bar{s}}}$ we construct segments $\Gamma_{s,i} \mid_{i \in I_{\bar{s}}}$ of V_s^{n+1} in the following way:
(i) For each interval, we add the convoluted future value functions by techniques described in Theorem 1:

$$\vec{\Gamma}_{s,i} \leftarrow [\vec{\Gamma}_{\bar{s},i}[1] \mid \vec{\Gamma}_{\bar{s},i}] \tag{1}$$

Where $[A \mid B]$ is a concatenation of A and B and $\vec{\Gamma}_{\bar{s},i}[1]$ accesses the first coefficient c_1 of $\vec{\Gamma}_{\bar{s},i}$
(ii) For intervals $i > 1$ we add to $\vec{\Gamma}_{s,i}$ breakpoint smoothing value for interval i [1]. We modify $\vec{\Gamma}_{s,i}$ as follows:

$$\vec{\Gamma}_{s,i} \leftarrow \vec{\Gamma}_{s,i} - [0, constant_i, 0, ..., 0] \tag{2}$$

(iii) Finally, for each interval we add the $p \diamond r$. Since from Theorem 1, $\overrightarrow{(p \diamond r)} = [r, r]$ we have:

$$\vec{\Gamma}_{s,i} \leftarrow \vec{\Gamma}_{s,i} + [r, r, 0, ..., 0] \tag{3}$$

In the non-deterministic case if $P(s_i|s,a) = p_i \ \forall_{s_i \in S}$ then $\vec{\Gamma}_s = \sum_{s_i \in S} p_i \cdot \vec{\Gamma}_{s_i}$ It follows from equations (1), (2) and (3) that, in the general case, when immediate rewards and non-deterministic transitions are considered, value functions stay within the PWΓ form.

4.3 Extension to Phase-type distributions

If action duration time pdfs are exponential with different λ values, we use uniformization to establish a common λ.

Whenever a duration time pdf for an action a at state s_i moving the process to state s_j is non-exponential, for each link (s_i, a, s_j) we use a Phase-type distribution graph [7] (denoted by *PH*). A PH graph is directed and often cyclic, its nodes are called phases and links between phases are transitions with exponentially distributed durations (Figure 3). We then use the techniques described by equations (1), (2) and (3) for each phase, in order to derive $V_{s_i}^n$ from $V_{s_j}^n$. If the original MDP is acyclic, the new MDP can have cycles leading to infinite horizon policies. The next section shows that despite such cycles, we can plan for a finite horizon while bounding policy error.

[1]Because 2^{nd} coefficient of $\vec{\Gamma}$ actually decreases Γ function (it is multiplied by $-e^{-\lambda t}$), breakpoint smoothing value should decrease this coefficient so as to increase the value of Γ.

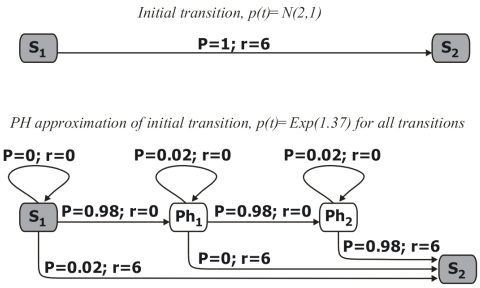

Figure 3: Phase-type approximation for an action with duration $p(t) = N(2,1)$. P is the discrete state transition probability and r is the reward for successful transition

4.4 Error Analysis

Infinite Horizon Error

Both Phase-type distributions and uniformization introduce self-transitions. Consequently, we have to plan for infinite action horizon. We now calculate for how many iterations n^* value iteration needs to run at least to ensure that the infinite horizon approximation error is smaller than a given constant. This analysis does not take into account the error introduced by approximating the probability distributions over the execution time with phase transitions.

We now introduce the notation needed to determine n^*. We define $R_{max} := \max_{s', s \in S, a \in A} R(s, a, s')$, $\alpha := \lambda\Delta$ and $s_n := \sum_{i=n}^{\infty} \frac{\alpha^i}{i!}$. (Notice that $s_0 = e^\alpha$ and $s_1 = s_0 - 1 = e^\alpha - 1$.) Then, the probability $p_n(t)$ that the execution time of a sequence of $n \geq 1$ actions is no more than t is

$$\begin{aligned} p_n(t) \leq p_n(\Delta) &= \int_0^\Delta e^{-\lambda t'} \frac{t'^{n-1}\lambda^n}{n!}\, dt' \\ &= \frac{1}{e^\alpha}(e^\alpha - \sum_{i=0}^{n-1} \frac{\alpha^i}{i!}) = \frac{1}{e^\alpha}\left(\sum_{i=0}^{\infty} \frac{\alpha^i}{i!} - \sum_{i=0}^{n-1} \frac{\alpha^i}{i!}\right) \\ &= \frac{1}{e^\alpha} \sum_{i=n}^{\infty} \frac{\alpha^i}{i!} = \frac{s_n}{e^\alpha}. \end{aligned}$$

Notice that

$$\frac{s_{n+1}}{s_n} = \frac{s_{n+1}}{\frac{\alpha^n}{n!} + s_{n+1}} = \frac{1}{\frac{\alpha^n}{n!\, s_{n+1}} + 1}.$$

$\frac{s_{n+1}}{s_n}$ decreases strictly monotonically in n because

$$\frac{\alpha^n}{n!\, s_{n+1}} = \frac{\alpha^n}{n! \sum_{i=n+1}^{\infty} \frac{\alpha^i}{i!}} = \frac{1}{\frac{\alpha}{n+1} + \frac{\alpha^2}{(n+2)(n+1)} + \frac{\alpha^3}{(n+3)(n+2)(n+1)} + \dots}$$

increases strictly monotonically in n. Consequently,

$$1 > \frac{e^\alpha - 1}{e^\alpha} = \frac{s_1}{s_0} > \frac{s_2}{s_1} > \frac{s_3}{s_2} > \dots > 0.$$

Thus,

$$\begin{aligned} s_n &< \frac{s_{n-1}}{s_{n-2}} s_{n-1} < \frac{s_1}{s_0} s_{n-1} < \frac{s_1}{s_0}\frac{s_{n-2}}{s_{n-3}} s_{n-2} < \left(\frac{s_1}{s_0}\right)^2 s_{n-2} \\ &< \dots < \left(\frac{s_1}{s_0}\right)^{n-1} s_1. \end{aligned}$$

We now use these results to show when $|V_s^*(t) - V_s^{n^*}(t)| \leq \epsilon$ for all $s \in S$ and $0 \leq t \leq \Delta$. Assume that the agent starts in state $s \in S$ with time-to-deadline $0 \leq t \leq \Delta$. Value iteration with n^* iterations determines the highest expected total reward $V_s^{n^*}(t)$ under the restriction that the agent can plan for executing at most n^* actions in any state $s \in S$. The highest expected total reward $V_s^*(t)$ does not have this restriction. It can be larger than $V_s^{n^*}(t)$

because the agent can plan for executing additional actions. On the other hand, reward for the first n^* actions that $V_s^*(t)$ is made of is smaller than $V_s^{n^*}(t)$ because agent that plans for fewer actions ahead will try to maximize its rewards faster. Consequently, we want to bound the reward the agent receives for the (n^*+1)st $[(n^*+2)$nd, ...) action. The probability that the agent can execute the (n^*+i)th action is $p_{n^*+i}(t)$ and for each of this actions the accumulated reward is at most R_{max}. Thus,

$$\begin{aligned} 0 \leq V_s^*(t) - V_s^{n^*}(t) &\leq \sum_{i=n^*+1}^{\infty} R_{max} p_i(t) \\ &\leq \frac{R_{max}}{e^{\alpha}} \sum_{i=n^*+1}^{\infty} s_i \\ &< \frac{R_{max}}{e^{\alpha}} \sum_{i=n^*+1}^{\infty} \left(\frac{s_1}{s_0}\right)^{i-1} s_1 \\ &= \frac{R_{max} s_1}{e^{\alpha}} \sum_{i=0}^{\infty} \left(\frac{s_1}{s_0}\right)^{i+n^*} \\ &= \frac{R_{max} s_1}{e^{\alpha}} \left(\frac{s_1}{s_0}\right)^{n^*} \frac{1}{1-\frac{s_1}{s_0}}. \end{aligned}$$

We want to bound this expression by ϵ from above, resulting in

$$\begin{aligned} \frac{R_{max} s_1}{e^{\alpha}} \left(\frac{s_1}{s_0}\right)^{n^*} \frac{1}{1-\frac{s_1}{s_0}} &\leq \epsilon \\ \left(\frac{s_1}{s_0}\right)^{n^*} &\leq \frac{\epsilon(1-\frac{s_1}{s_0})e^{\alpha}}{R_{max} s_1} \\ n^* &\geq \log_{\frac{s_1}{s_0}} \frac{\epsilon(1-\frac{s_1}{s_0})e^{\alpha}}{R_{max} s_1} \\ n^* &\geq \log_{\frac{e^{\lambda\Delta}-1}{e^{\lambda\Delta}}} \frac{\epsilon}{R_{max}(e^{\lambda\Delta}-1)}. \end{aligned}$$

Finite Horizon Error

Although CPH is an analytical algorithm, it is not error free due to the numerical process involved in finding the breakpoints which we require by *Piecewise Linear* (PWL) approximation of V_s^n. Our experiments show that this error ξ in finding breakpoints is small.

5. CPH ALGORITHM

We now present a novel algorithm referred to as CPHSOLVER that implements the analytical solution techniques for continuous state MDPs. The algorithm is designed to handle one continuous dimension, for example time.

Algorithm 1 CPHSOLVER($\mathcal{M}, \epsilon$)

```
1: M ←APPROXIMATEWITHPH (M)
2: M ←UNIFORMIZE (M)
3: n* ← ⌈log_{(e^{λΔ}−1)/e^{λΔ}} ε/(R_max(e^{λΔ}−1))⌉
4: for all n = 0...n* − 1 do
5:    T_n = GENERATETEMPLATE(n)
6: for all n = 1...n* do
7:    for all state ∈ S do
8:       CPH(M.state)
```

CPH algorithm starts with a continuous state MDP $\mathcal{M}$ and an infinite horizon error ϵ (in case of non-PH transition pdfs, APPROXIMATEWITHPH method has to be called). After the uniformization process establishes the common exit rate λ the algorithm determines a finite horizon n^* within the acceptable error ϵ from the optimal policy generated for the infinite horizon. Next, necessary templates (explained later) are generated and value functions of all states are set to 0. Finally, $\forall_{n=1,...,n^*}$ we loop over all $state \in S$ and execute CPH($\mathcal{M}.state$) subroutine.

Algorithm 2 CPH($\mathcal{M}.state$)

```
1: Υ_state ← ∅
2: for all a ∈ Actions do
3:    Υ_a ← ∅
4:    S̄ ←DESTSTATES(a, state)
5:    for all i ∈ FINDCOMMONINTERVALS(S̄) do
6:       Γ_i ← 0;  ASSIGNACTION(Γ_i, a)
7:       for all s̄ ∈ S̄ do
8:          j ← FINDLOCALINTERVAL (Υ_s̄, i)
9:          Γ⃗_i ← Γ⃗_i + P(s̄|state, a) · [Γ⃗_{s̄,j}[1] | Γ⃗_{s̄,j}]
10:         if i ≠FIRSTCOMMONINTERVAL(Υ_S̄) then
11:            t_last =INTERVALEND(i_last)
12:            const ← (Γ_last − Γ_i)(t_last) · e^{λ t_last}
13:            Γ⃗_i[2] ← Γ⃗_i[2] − const
14:      Γ_last ← Γ_i;  i_last ← i
15:      Υ_a ← CONCATENATEΓ (Υ_a, Γ_last)
16:   for all ⟨Γ, X⟩ ∈ Υ_a do
17:      Γ⃗[1] ← Γ⃗[1] + Σ_{s̄∈S̄} P(s̄|state, a)R(state, a, s̄)
18:      Γ⃗[2] ← Γ⃗[2] + Σ_{s̄∈S̄} P(s̄|state, a)R(state, a, s̄)
19:   Υ_state ← FINDDOMINANCY (Υ_state, Υ_a)
```

CPH subroutine is made of 3 blocks: $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. Block $\mathcal{A}$ spanning lines $7-9$ implements equation (1), block $\mathcal{B}$ spanning lines $10-13$ implements equation (2) and block $\mathcal{C}$ spanning lines $16-18$ implements equation (3). The CPH subroutine maintains for each state $state$ the list Υ_{state} of all $\Gamma_{state,i}(t)$ functions that are dominant for at least one point $t \in [0, \Delta]$. First, after resetting value function Υ_{state} each executable action a in state s is analyzed (line 2). Our goal is to construct Υ_a, the expected reward function at state $state$ given a is an action to be executed. Let $\bar{S}$ be the set of future states returned by the DESTSTATES function. Each $\bar{s} \in \bar{S}$ can have different $\Gamma_{\bar{s}}$ functions dominating over different time intervals. In line 5 (FINDCOMMONINTERVALS call) we determine all common intervals in which $\Gamma_{\bar{s}}$ $\forall_{\bar{s}\in\bar{S}}$ do not change (e.g. if $\bar{s_1}$ has different Γ for intervals (0,1),(1,3) and (3,4) whereas $\bar{s_2}$ has different Γ for intervals (0,2) and (2,4), *common intervals* are (0,1),(1,2),(2,3) and (3,4)). After assigning action a to function Γ_i (needed for policy execution phase) in line 6 we then for each $\bar{s}$ collect its $\vec{\Gamma}_{\bar{s},j}$ (where j is the current local interval of $\bar{s}$ returned by a FINDLOCALINTERVAL call), convolute it with p in a way described by Theorem 1 and multiply the newly constructed vector by $P(\bar{s}|s,a)$. After summing over all the states $\bar{s} \in \bar{S}$ all previously multiplied vectors, we obtain a new function Γ_i for the common interval i of the current state s (lines 7-9). If i is not the first common interval we still have to modify the result function by the breakpoint smoothing value (lines 10-13). After concatenating the Γ_{last} function at the end of the Υ_a list (line 15, CONCATENATEΓ function) and repeating lines 5-15 until the last common interval is analyzed, we increase each $\Gamma \in \Upsilon_a$ by the convoluted expected reward for action a (lines 16-18). Finally we call FINDDOMINANCY function which compares the constructed Υ_a against the currently dominant Υ_{state}, finds all the breakpoints and generates a new Υ_{state} list that contains dominant Γ functions over both Υ_a and old Υ_{state}. Note, that since each Γ has an associated action, FINDDOMINANCY also finds the best action for *state* and $t \in [0, \Delta]$ for the current Bellman iteration.

CPH subroutine efficiency depends on the ability to quickly find the dominant segments of two Γ functions (line 19). We achieve this by storing template functions $\mathcal{T}_n$ for $n = 0, 2, ..., n^* - 1$ in a lookup table. Each $\mathcal{T}_n$ is a PWL approximating $e^{-\lambda t}\frac{(\lambda t)^n}{n!}$ within accuracy of ξ. If $\vec{\Gamma}_s$ has the size n, then Γ_s approximation construc-

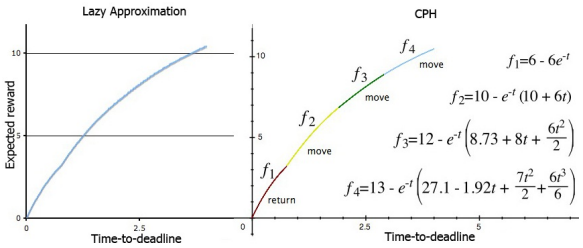

Figure 4: Correctness of CPH solution

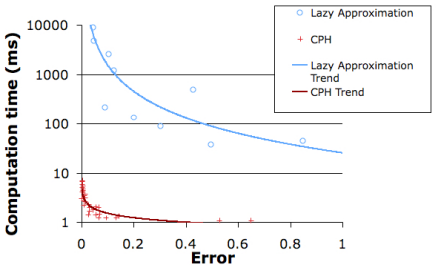

Figure 5: LA vs CPH (Mars Rover domain)

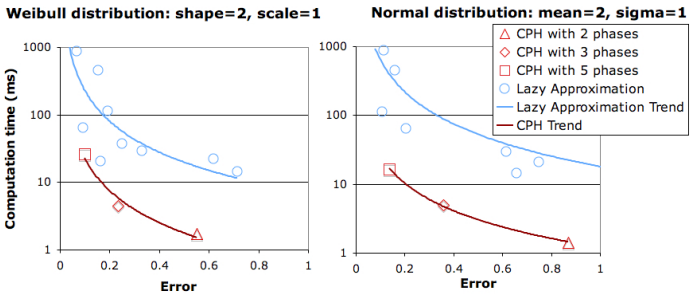

Figure 6: Comparison LA and CPH running time single Bellman update operation

tion follows the scheme: $\Gamma_s = \vec{\Gamma}_s[1] - \sum_{i=0,\ldots,n-2} \vec{\Gamma}_s[i+2] \cdot \mathcal{T}_i$. With two PWLs constructed, we can easily find their approximate dominancy intervals. The template construction (done only once) takes a negligible amount of time in comparison with time needed to perform Bellman update iterations.

6. EXPERIMENTS

We have implemented CPH and tested it in three different ways. Our first experiment tests the correctness of CPH's analytical formulas for the Mars rover domain. Indeed, the optimal policy for the starting state found by CPH consists of 4 segments that perfectly matches the numerical result generated by Lazy Approximation (LA) running significantly slower (two orders of magnitude) due to a very small approximation error of LA. Figure 4 illustrates the optimal value functions for state *Start* computed numerically by LA (left) and analytically by CPH (right) with the x-axis representing the continuous variable (remaining time) and the y-axis representing the expected reward.

Our second experiment provides a head-to-head comparison between LA and CPH for the Mars rover domain. The results are depicted in figure 5 where the x-axis presents the $\mathcal{L}_\infty$ error (max. distance between two functions) between the *correct optimal policy* and optimal policies generated by both algorithms and the y-axis denotes algorithm running time in milliseconds in log scale. The results show that CPH is up to 3 orders of magnitude faster than LA. For example, to compute a policy less than 1% off the optimal, which corresponds to $\mathcal{L}_\infty = 0.13$ for our domain, CPH needed only $2ms$ in contrast to $1000ms$ required by LA.

Our final experiment compares CPH to LA for various continuous state transition pdfs. We approximated a Weibull distribution with $\beta = 2$, $\alpha = 1$ and a Normal curve with $\mu = 2$ and $\sigma = 1$ using Phase-type distributions and tested the accuracy of a Bellman iteration step for both algorithms (depicted in figure 6 with the same axis as in 5). We fixed PWL approximation accuracy of CPH at a small value ξ and varied PH approximation accuracy by choosing $2, 3$ and 5 phases. Our results averaged over 100 runs confirmed that even after introducing PH approximations of Normal and Weibull pdfs in CPH, it still outperformed LA, achieving on average a 10 fold speedup. In particular, for the Normal distribution CPH using 5 phases computed a Bellman update step with en error of less than 0.2 in just 12 ms. In contrast, LA needed at least 100 ms for the same task. The results illustrate that we can improve the accuracy of CPH by introducing more accurate approximations while still outperforming LA.

7. SUMMARY

Despite recent advances in solving continuous state MDPs [6], finding a near-optimal policy incurs significant computational expense. We presented CPH, a novel algorithm for solving continuous state MDPs with either finite or infinite horizon, based on two key ideas. First, instead of discretizing the transition function or using a piecewise constant function approximation (which can be restrictive or lead to high computational costs), we rely on Phase-type distributions for modeling the transition functions. CPH exploits Phase-type distributions for application to problems with arbitrary probability density functions over continuous state transitions. Second, CPH exploits properties of exponential distributions, which are the basic building blocks of Phase-type distributions, to perform Bellman updates very efficiently through the use of an analytical solution to the required convolution process. Experiments on domains with different types of transition functions show a speedup of a few orders of magnitude over the state-of-the-art Lazy Approximation technique.

8. REFERENCES

[1] D. Blidberg. The development of autonomous underwater vehicles (AUVs); a brief summary. In *ICRA*, 2001.

[2] J. Boyan and M. Littman. Exact solutions to time-dependent MDPs. In *NIPS*, pages 1026–1032, 2000.

[3] J. Bresina, R. Dearden, N. Meuleau, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *UAI*, pages 77–84, 2002.

[4] Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous Markov decision problems. In *UAI*, pages 154–161, 2004.

[5] A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skandinavisk Aktuarietidskrift*, 36:87–91, 1953.

[6] L. Li and M. Littman. Lazy approximation for solving continuous finite-horizon MDPs. In *AAAI*, pages 1175–1180, 2005.

[7] M. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. John Hopkins University Press, Baltimore, 1981.

[8] M. Puterman. *Markov decision processes*. John Wiley and Sons, New York, 1994.

[9] H. Younes and R. Simmons. Solving generalized semi-Markov decision processes using continuous phase-type distributions. In *AAAI*, pages 742–747, 2004.