Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

DESPOT-α: Online POMDP Planning With Large State And Observation Spaces

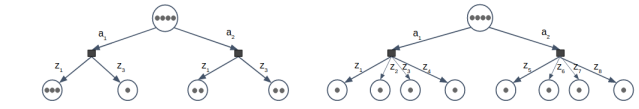
Neha P Garg*, David Hsu† and Wee Sun Lee‡
*†‡School Of Computing, National University Of Singapore
*†NUS Graduate School Of Integrative Sciences And Engineering

***Abstract*—State-of-the-art sampling-based online POMDP solvers compute near-optimal policies for POMDPs with very large state spaces. However, when faced with large *observation* spaces, they may become overly optimistic and compute sub-optimal policies, because of particle divergence. This paper presents a new online POMDP solver DESPOT-α, which builds upon the widely used DESPOT solver. DESPOT-α improves the practical performance of online planning for POMDPs with large observation as well as state spaces. Like DESPOT, DESPOT-α uses the particle belief approximation and searches a determinized sparse belief tree. To tackle large observation spaces, DESPOT-α shares sub-policies among many observations during online policy computation. The value function of a sub-policy is a linear function of the belief, commonly known as α-vector. We introduce a particle approximation of the α-vector to improve the efficiency of online policy search. We further speed up DESPOT-α using CPU and GPU parallelization ideas introduced in HyP-DESPOT. Experimental results show that DESPOT-α/HyP-DESPOT-α outperform DESPOT/HyP-DESPOT on POMDPs with large observation spaces, including a complex simulation task involving an autonomous vehicle driving among many pedestrians.**

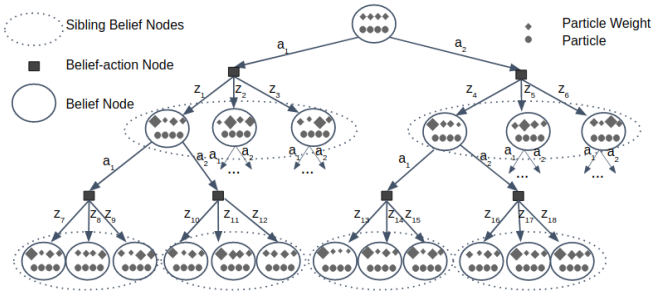
I. Introduction

To work in unstructured and uncontrolled environments like our homes and offices, a robot needs to represent the system with a large number of state variables. In addition to that, it cannot rely on complete information about the system state; the robot has to gather information from noisy sensors and do decision making under uncertainty. Real world sensors like vision, touch, sound, etc. often provide very high dimensional observations. Thus planning under uncertainty with very large state and observation spaces is essential for service robots.

Partially Observable Markov Decision Processes (POMDPs) provide a principled framework for planning under uncertainty. Unfortunately, solving POMDPs exactly is computationally intractable. The computational difficulties come from various sources. First, POMDP solvers need to maintain a conditional distribution of the states given the action-observation history, also called a *belief*, as a sufficient statistic for making future decisions. The state space, and correspondingly the dimensionality of the belief size grows exponentially with the number of state variables; this is an effect of the *curse of dimensionality*. Many POMDP solvers do online planning by doing forward search from the current belief, constructing a tree which branches each time an action is required, and also each time an observation may be observed. The belief tree grows exponentially with depth; this is the *curse of history*.



(a) DESPOT Search Tree: Small (left) and large (right) observation space



(b) DESPOT-α Search Tree

Fig. 1: Each tree node represents a belief. Solid line circles are belief nodes and black squares are belief-action nodes. On taking action a, all particles (circular dots) transition to next state and produce observations. For DESPOT, particles which produce the same observation go to the same child belief node. For DESPOT-α, all particles go to child belief nodes. Each node consists of particle weights (diamond shapes) also and observations only affect particle weights. The belief nodes which differ from each other only in last observation are sibling belief nodes (Marked by dotted circle).

State-of-the-art online POMDP solvers like DESPOT [19] and POMCP [15] use Monte Carlo methods for both belief tracking and tree search in order to handle the curse of dimensionality. These solvers represent a belief by a set of sampled states called *particles* to overcome the issue of large state space; they further sample action-observation trajectories to compute approximate value of different policies for current belief quickly. Current state-of-the-art solvers can compute near-optimal policies for POMDPs with large state spaces.

However when observation space is large, particles quickly diverge into separate belief nodes in the belief tree, each of which contains only a single particle, due to the very low probability of generating same observation twice. This leads to solvers generating policies that underestimate the uncertainty, leading to poor and over-optimistic actions. This issue has also been highlighted by Sunberg and Kochenderfer [21]. Fig. 1a illustrates this issue for DESPOT solver.

In this paper, we address the large observation space issue within the DESPOT solver, although the ideas used may also be useful for other solvers. To deal with particle divergence, instead of partitioning the particles according to the observation that each particle generates, we use all the particles to represent beliefs for different observations generated (see Fig. 1b). Observations only affect the weight of each particle which is updated according to the relative likelihood of the observation. Thus each node in the tree has the same number of particles as the root of the tree, which prevents the over-optimistic evaluation of value of the belief.

However, by propagating each particle to every child belief node, we lose some of the computational efficiency of DESPOT. To regain some of the efficiency, we use ideas from dynamic programming by maintaining a partial value function. It is known that the value function of a POMDP can be approximated arbitrarily well by a convex piece-wise linear function of the belief, where each linear function is called an α-vector [16] i.e.

$$V(b) = \max_{\alpha \in \Gamma} \sum_{s \in S} b(s)\alpha(s) \tag{1}$$

where Γ is a set of α-vectors. An α-vector is associated with a conditional plan and for each state s, captures the reward of executing the plan starting from state s. If we have an α-vector, then we can use it to evaluate the value of the associated conditional plan for any belief b by simply doing the inner product of alpha vector and belief. However, to apply the concept of α-vectors, we need to know the value of the conditional plan associated with the α-vector for each state. The number of components in an α-vector correspond to the number of states and hence can be exponentially large, and furthermore, we only keep a sample of states in each node of the belief tree making the computation of the α-vector components difficult. The main contribution of this paper is a method to approximate enough of the α-vectors so that it can be used to improve online computation of the policy.

We approximate enough of an α-vector to use for the *sibling belief nodes* i.e nodes representing beliefs which differ from each other only in last observation (see Fig. 1b). In a determinized tree, all the sibling belief nodes share the same particles. Thus α-vector components corresponding to those particles can be shared. While sharing is limited to sibling belief nodes, this provides substantial gains in the case of large observation spaces where many sibling belief nodes may be quite similar in terms of the conditional plan that maximizes their value. We name our algorithm DESPOT-α (Determinized Sparse Partially Observable Tree With α-Vector Update).

DESPOT-α shares the same computational bottlenecks as DESPOT with an additional computation of weight update for every observation and state pair that can be parallelized easily. Therefore we are able to use the ideas in Hyp-DESPOT [4], a parallelized version of DESPOT, to parallelize DESPOT-α as well. In fact as we will see later, we are able to better utilize the GPU parallelization in HyP-DESPOT-α as compared to HyP-DESPOT because each node contains all the particles.

DESPOT-α/HyP-DESPOT-α outperform DESPOT/HyP-DESPOT on POMDPs with large observation spaces, including a complex simulation task involving an autonomous vehicle driving among many pedestrians.

II. Background

A. POMDP

A POMDP is defined by a tuple $< S, A, Z, T, O, R >$ where S is the state space, A is the action space, Z is the observation space. State transition function $T(s, a, s') = p(s'|s, a)$ is the probability of next state s' when action a is taken in state s. The observation function $O(s', a, z) = p(z|s', a)$ is probability of observing z in state s' reached by performing action a. $R(s, a)$ is the immediate reward obtained on taking action a in state s. Uncertainty is modeled by maintaining a belief b, which is a probability distribution over S. The solution to a POMDP is a policy $\pi : B \to A$ which maps belief $b \in B$ to an action $a \in A$ such that the expected total discounted reward $V_\pi(b)$ as defined below is maximized.

$$V_\pi(b) = E\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t))|b_0 = b\right). \tag{2}$$

Here γ is the discount factor, b_0 is the initial belief. By reasoning in belief space, POMDPs are able to maintain a balance between exploration and exploitation and hence provide a principled framework for decision making under uncertainty. The solution to a POMDP can be found using Bellman's principle of optimality:

$$V^*(b) = \max_{a \in A} Q^*(b, a) \tag{3}$$

where

$$Q^*(b, a) = \sum_{s \in S} b(s)R(s, a) + \gamma \sum_{z \in Z} p(z|b, a)V^*\big(\tau(b, a, z)\big) \tag{4}$$

B. RELATED WORK

Most of the practical solvers try to approximate the value function $V^*(b)$, by sampling the beliefs from belief space. There are two types of solvers: offline and online. Offline solvers, e.g. [2, 8, 11, 13, 18, 20, 22, 23], approximate optimal policies for different beliefs offline and use the pre-computed policy for online execution. Bai et al. [2] use α-vectors for continuous state and observation POMDPs for offline planning – the approach is difficult to modify for online planning. Offline solvers can compute very good policies for the sampled beliefs but they are difficult to scale up to large POMDPs due to large increase in the number of beliefs that should be sampled to cover most situations that will occur in execution.

Online solvers, e.g. [3, 5, 10, 14, 15, 19, 21], compute the best policy only for the current belief at every step and need not pre-compute near-optimal policies for different beliefs. A near-optimal policy is generally computed by doing forward search in a sparse belief tree of depth D. Online solvers can scale to very large POMDPs but are slower than offline solvers as they do forward search at every step. Thus fast online search is critical for online solvers. A sparse sampling method by

Kearns et al. [10] is an online algorithm which can potentially deal with large observation spaces because it samples a fixed number of C observations for each action branch resulting in tree size of $O(C^D|A|^D)$, where $|A|$ is the number of actions. Kearns et al. [10] provide a theoretical performance bound which is independent of the size of the observation space. However the algorithm can be computationally inefficient due to the large tree size and is rarely used in practice.

One of the most widely used online POMDP solver, DESPOT [19], gains computational efficiency by making a tree of size $|A|^D K$, using K sampled scenarios. The idea of sampled scenarios is also used by [9, 12] but for offline planning. K is much smaller than C^D for many problems but DESPOT suffers from the particle divergence problem when the observation space is large. Our approach can be viewed as a mix of SparseSampling and DESPOT. We construct a tree of size $O(C^D|A|^D)$ like sparse sampling but use determinized scenarios like DESPOT. Determinized scenarios allow us to use α-vectors to reduce computation.

Another widely used online POMDP solver, POMCP [15] also suffers from particle divergence when the observation space is large. Sunberg and Kochenderfer [21] extended POMCP for POMDPs with large observation space with two approaches: POMCPOW and PFT-DPW. Like DESPOT-α, both the approaches use the idea of weighted particles to address the problem of particle divergence. However, they do not share the value function calculation among different belief nodes and hence would require a much longer planning time to compute near-optimal policy. This is reflected in their experiments where they are not able to outperform DESPOT on large scale problems like Laser Tag and Multilane.

One way of dealing with the issue of particle divergence is to group observations together. We can merge the observations, when the value of the resulting beliefs is maximized by the same α-vector. Hoey and Poupart [7] try to group observations based on this criteria. However they demonstrate results on POMDPs with very small state space. We are implicitly grouping beliefs whose values are maximised by same α-vector by sharing α-vectors between sibling belief nodes.

III. Our Approach

As our algorithm is based on DESPOT, we first give a brief overview of it. For details please see [19].

A. DESPOT

For fast online search, DESPOT samples K *scenarios* for the current belief. A *scenario* for a belief b is a sequence $\phi = (s_0, \phi_1, \phi_2, ...)$ where s_0 is randomly sampled from b and ϕ_i is a random number sampled from uniform distribution with range $[0, 1]$. Scenarios are used to 1) represent belief as a set of particles; 2) determinize the generation of next state and observation for the sampled state using ϕ_i when different actions are applied during forward search. Forward search does explore and backup operations repeatedly as explained below.

1) Explore: Exploration consists of multiple *trials* to build a determinized sparse belief tree incrementally. Initially the tree contains only the root node with K particles representing the current belief b_0. Then multiple trials are done with each trial traversing a path from root node to a leaf node. The leaf node is then expanded by applying all actions to each particle in the leaf node. The particles which produce same observation for a given action are grouped together and represent the belief for a new belief node. Thus the resulting belief tree (See Fig. 1a) contains all the action edges but only the observation edges that are reachable through sampled scenarios.

To make sure that even the partially constructed tree is able to compute a good policy, heuristics based on upper bound and lower bound [17] on the value of belief nodes are used to guide the search. Default policies can be used to calculate a lower bound and state based heuristics can be used to calculate upper bound on the value of each belief node using Eq. 2 and each belief-action node using Eq. 4. To explore the most promising parts of the tree first, each trial picks the action a^* that maximizes the upper bound ($\overline{Q}(b, a)$) for the belief node b and the observation z^* which maximizes weighted excess uncertainty (WEU) for child belief node $b' = \tau(b, a^*, z)$. WEU is defined as follows:

$$WEU(b') = p(z|b, a^*)excess(b') \quad (5)$$

$$= p(z|b, a^*)(\epsilon(b') - \xi\epsilon(b_0)) \quad (6)$$

where b' is reachable from b through a^*, z. $\epsilon(b) = \overline{V}(b) - \underline{V}(b)$ is the gap in the lower bound and upper bound at belief node b and ξ is a parameter used to control the desired uncertainty level at leaf nodes.

2) Backup: The trial ends when either the maximum depth is reached or WEU becomes negative. After that, the lower bound and upper bound are updated for each node encountered during trial using Bellman update Eq. 3, 4.

The trials continue until the desired gap $\epsilon(b)$ at the root node is reached or the time limit is reached.

B. DESPOT-α

DESPOT-α also does a similar anytime forward search through trials consisting of exploration and backup on sampled scenarios. However instead of propagating only the particles producing the same observation to the child of a belief-action node, we propagate all the particles to the child nodes (Fig. 1b) and update the weights of particles according to relative likelihood of observation $p(z|s, a)$. This is similar to a particle filter. $p(z|s, a)$ values are also generally available for particle filtering. For a belief b, represented by the particle set Φ_b, with each particle having weight $w_b(s)$, the weight of particles in child belief node $\tau(b, a, z)$ is:

$$w_{\tau(b,a,z)}(s') = \frac{p(z|s', a)\sum_{s\in\Phi_b} p(s'|s, a)w_b(s)}{p(z|b, a)} \quad (7)$$

$$\text{where } p(z|b, a) = \sum_{s'\in\Phi_{\tau(b,a,z)}} p(z|s', a) \sum_{s\in\Phi_b} p(s'|s, a)w_b(s).$$

In our determinized tree, a particle s transitions to only one particle s' i.e. Φ_b has one to one correspondence with

$\Phi_{\tau(b,a,z)}$. Let s'_- be the particle in Φ_b that transitions to particle s' and let s_+ be the particle in $\Phi_{\tau(b,a,z)}$ to which particle s transitions. Then

$$p(z|b,a) = \sum_{s' \in \Phi_{\tau(b,a,z)}} w_b(s'_-) p(z|s',a) = \sum_{s \in \Phi_b} w_b(s) p(z|s_+,a) \tag{8}$$

and

$$w_{\tau(b,a,z)}(s') = \frac{p(z|s',a) w_b(s'_-)}{p(z|b,a)} \tag{9}$$

The resulting tree is a determinized sparse belief tree as it still contains only the observation branches reachable by sampled scenarios. However every belief-action node can have up to C child belief nodes: as we do not use observations to decide which particles will go into each child node, we can sample only $C(\leq K)$ instead of K observations from K scenarios by using only C out of K *scenarios* to generate observations. Always having C child belief nodes prevents over optimistic evaluation of value of belief but also makes the tree size $(C|A|)^D$. As we will see later in this section, we can use α-vectors to share the computation done for one trial among sibling belief nodes for improving lower bounds.

Note that eventually after few information gathering actions, most of the weight would be concentrated around a few particles in the search tree. Particle filters do re-sampling when this happens. However in the search tree, re-sampling is not required as we only need to estimate the reward which gets discounted as depth increases.

1) α-Vector Derivation For DESPOT-α: We can use Eq. 3 to calculate the value of belief node approximating belief b by K particles in set Φ_b as follows:

$$V(b) = \max_{a \in A} Q(b,a) \tag{10}$$

where

$$Q(b,a) = \sum_{s \in \Phi_b} w_b(s) R(s,a) + \gamma \sum_{z \in C_{b,a}} p(z|b,a) V\big(\tau(b,a,z)\big) \tag{11}$$

Unfortunately, the sampled observations may not contain all possible observations. To handle this, we can divide the second term on right hand side in Eq. 11 by a normalization constant $\eta = \sum_{z \in C_{b,a}} p(z|b,a)$. However this makes the value of α-vector derived in Eq. 17 dependent on η, which is dependent on the weights of the particles. With this dependency, α-vectors cannot be shared among sibling belief nodes.

To overcome this issue, we assume a dummy observation z_{res}, that contains all unsampled observations. Thus $|C_{b,a}| = C + 1$. When a large enough set of observations is sampled, the probability of z_{res} is small. We will discuss in section III-B2, how we determine $p(z_{res}|s,a)$.

For faster computation, we wish to define the approximate lower bound on value of belief node using α-vectors as follows:

$$\underline{V}_n(b) = \sum_{s \in \Phi_b} w_b(s) \alpha_{n,b}(s) \tag{12}$$

where n is the depth of the tree below the node. We add it as a subscript only for induction derivation. We show by induction that value function can be defined as above for our determinized sparse belief tree. For leaf nodes i.e. $n = 0$, Eq. 12 holds as we can define

$$\alpha_{0,b}(s) = \alpha^{def}_{b_0,\zeta(b)}(s) = \sum_{t=d}^{\infty} \gamma^d R(s_t, \pi_{def}(b_0, \zeta(b)) | s_d = s) \tag{13}$$

where $\zeta(b) = a_0, a_1 ... a_{d-1}$ is the action sequence followed to reach the leaf node b at depth d. Our default policy π_{def} is only dependent on initial belief b_0 and the action sequence $\zeta(b)$. Any fixed-action policy satisfies this. The value of such a policy can be used as a lower bound (See Hauskrecht [6]).

Now suppose Eq. 12 holds for n. Then

$$\underline{V}_{n+1}(b) = \max_{a \in A} \Big\{ \sum_{s \in \Phi_b} w_b(s) R(s,a) + \gamma \sum_{z \in C_{b,a}} p(z|b,a) \underline{V}_n\big(\tau(b,a,z)\big) \Big\} \tag{14}$$

Substituting value of $\underline{V}_n\big(\tau(b,a,z)\big)$ from Eq. 12 and value of $w_{\tau(b,a,z)}(s')$ from Eq. 9 we get:

$$\begin{aligned} \underline{V}_{n+1}(b) &= \max_{a \in A} \Big\{ \sum_{s \in \Phi_b} w_b(s) R(s,a) + \\ &\quad \gamma \sum_{z \in C_{b,a}} \sum_{s' \in \Phi_{\tau(b,a,z)}} w_b(s'_-) p(z|s',a) \alpha_{n,\tau(b,a,z)}(s') \Big\} \\ &= \max_{a \in A} \Big\{ \sum_{s \in \Phi_b} w_b(s) R(s,a) + \\ &\quad \gamma \sum_{z \in C_{b,a}} \sum_{s \in \Phi_b} w_b(s) p(z|s_+,a) \alpha_{n,\tau(b,a,z)}(s_+) \Big\} \end{aligned} \tag{15}$$

For node at level $n+1$, we can re-write Eq. 15 as:

$$\underline{V}_{n+1}(b) = \sum_{s \in \Phi_b} w_b(s) \big(\alpha_{n+1,b}(s)\big) \;\; s.t. \;\; \alpha_{n+1,b}(s) = \alpha^{a^*}_{n+1,b}(s) \tag{16}$$

where $a^* = \arg\max_{a \in A} \Big\{ \sum_{s \in \Phi_b} w_b(s) \alpha^a_{n+1,b}(s) \Big\}$ with

$$\alpha^a_{n+1,b}(s) = R(s,a) + \gamma \sum_{z \in C_{b,a}} p(z|s_+,a) \alpha_{n,\tau(b,a,z)}(s_+) \tag{17}$$

As sibling belief nodes share the same set of *scenarios* with different weights, α-vector calculated for one belief node can be used to calculate approximate lower bound for the sibling belief nodes by simply doing an inner product of weights of the particles and the α-vector. This allows us to provide an approximation without expanding those nodes. If the observations result in similar beliefs in terms of the conditional plan which maximizes their values, which we believe would often hold for very large observation space, the approximation will be effective.

The α-vectors only provide approximate lower bounds for the belief node values. For better exploration, we also need to update the upper bounds on sibling belief node values. We use SAWTOOTH approximation [6] for this.

2) z_{res} Observation Probability: In order to increase the weight on the sampled observations, we try to minimize the probability assigned to z_{res} by re-weighting $p(z|s',a)$. Let

$$\eta_{max} = \max_{s'} \sum_{z \in C_{b,a} \setminus z_{res}} p(z|s',a) \tag{18}$$

Then let $l(z|s',a) = p(z|s',a)/\eta_{max}$ and $l(z_{res}|s',a) = 1 - \sum_{z \in C_{b,a} \setminus z_{res}} l(z|s',a)$. Using $l(z|s',a)$ instead of $p(z|s',a)$ increases the weight on the sampled observations while keeping the probabilities proportional to the original probability distribution. This also allows us to use $p(z|s',a)$ values which are only proportional to true probability values. Experimental results show that this works well in practice.

Algorithms 1-5 provide pseudo-code of DESPOT-α. Since many belief nodes can be reached by same action sequence and differ only in observation branches, we store action sequence data (*AD*) for each action sequence encountered. An entry in *AD* for a sequence $\zeta(b)$ is a tuple $< P, Z, R, L, \alpha^{def}, \overline{\alpha}^{def}, \overline{\alpha}^{pp} >$ where P is the set of particles, Z is a set of up to C sampled observations, R is the immediate rewards, L is observation likelihoods for each sampled observation and particle, $\alpha^{def} = \alpha^{def}_{b_0,\zeta(b)}$ is default lower bound alpha vector, $\overline{\alpha}^{def}$ is the default upper bound vector and $\overline{\alpha}^{pp}$ is upper bound per particle.

Default upper bound is computed as an inner product of default upper bound for each particle with particle weights. Thus we store the default upper bound value for each particle in $\overline{\alpha}^{def}$. $\overline{\alpha}^{pp}$ is stored for sawtooth approximation as it represents upper bound for beliefs at extreme corners. This data can be used by all the child belief nodes which are reached from root node following the same action sequence. Thus we only need to make $K|A|^D$ calls to transition model and $CK|A|^D$ to observation model.

We maintain the best α-vector for each belief node along with the lower bound value. During backup, in addition to updating the lower/upper bounds of the nodes visited during the trial, we also update the lower/upper bounds of sibling nodes (See algorithm 3, 5). Thus we need fewer trials.

Even though we need fewer trials than DESPOT, for a given trial we have to do more computation to calculate lower/upper bounds using default policy for all the K particles while expanding a node. We also need to compute $p(z|s',a)$ for all C child nodes which is $\mathcal{O}(KC)$ operation. If the problem scale in terms of number of actions or number of particles K is large, or if the default policies have very long horizon, we might not be able to do enough number of trials. Therefore, we parallelize the tree search as discussed next.

C. HyP-DESPOT-α

HyP-DESPOT [4] is a parallel tree search algorithm which parallelizes sparse belief tree search in DESPOT. CPU threads do EXPLORE and BACKUP (lines 7-11 in Alg. 1) and invoke GPU kernels for EXPAND i.e transition each particle to next state for different actions and calculate lower/upper bound (Line 4 in Alg. 4). We can use the same framework to parallelize our search. Since the particles do not diverge during search, we can better utilize GPU parallelization as we will always have K particles to expand in parallel. Our EXPAND algorithm contains additional calculation of $p(z|s',a)$ for each state observation pair (Line 5 in Alg. 4). For this we simply add an additional GPU kernel during node expansion which parallelizes $|A|KC$ operations. We do not currently parallelize the weight computation (Lines 15-16 in Algorithm 4) because latency of copying weights to and from GPU memory is higher than doing the inner product computation. If the number of sampled scenarios are so large that the computation time exceeds memory latency then we can parallelize this part too.

Algorithm 1: BUILDDESPOTALPHA(b_0)

1 Sample randomly a set $\mathbf{\Phi}_{b_0}$ of K scenarios from the current belief b_0. Assign weight of $1/K$ to each particle.
2 Create a new DESPOT $\mathcal{D}$ with a single node b_0 as the root. Initialize *AD* $\leftarrow \{\}$, $\zeta(b_0) \leftarrow \emptyset$. Compute α^{def}, $\overline{\alpha}^{def}$, $\overline{\alpha}^{pp}$ for each particle in $\mathbf{\Phi}_{b_0}$.
3 *AD*$(\zeta(b_0)) \leftarrow < \mathbf{\Phi}_{b_0}, \emptyset, \emptyset, \emptyset, \alpha^{def}, \overline{\alpha}^{def}, \overline{\alpha}^{pp} >$.
4 $\alpha_{b_0} \leftarrow \alpha^{def}$.
5 Compute lower, upper bound ($\underline{V}(b_0)$, $\overline{V}(b_0)$) by dot product of α^{def} and $\overline{\alpha}^{def}$ with particle weights.
6 $\epsilon(b_0) \leftarrow \overline{V}(b_0) - \underline{V}(b_0)$.
7 **while** $\epsilon(b_0) > \epsilon_0$ *and the total running time is* $< T_{\max}$ **do**
8 $\quad b \leftarrow$ EXPLORE($\mathcal{D}, b_0$).
9 $\quad$ BACKUP($\mathcal{D}, b$).
10 $\quad \epsilon(b_0) \leftarrow \overline{V}(b_0) - \underline{V}(b_0)$.
11 **end**
12 **return** $\arg\max_{a \in A} \alpha^a_{b_0}$

Algorithm 2: EXPLORE($\mathcal{D}, b$)

1 **while** $depth(b) \leq D$ *and* $WEU(b) > 0$ **do**
2 $\quad$ EXPAND(b, $\mathcal{D}$) if b is a leaf node in $\mathcal{D}$.
3 $\quad a^* \leftarrow \arg\max_{a \in A} \overline{Q}(b,a)$.
4 $\quad z^* \leftarrow \arg\max_{z \in Z_{b,a^*}} WEU(\tau(b,a^*,z))$.
5 $\quad b \leftarrow \tau(b,a^*,z^*)$.
6 **end**
7 **return** b.

IV. EXPERIMENTS

To evaluate (HyP-)DESPOT-α, we run it on the following problems with large observation spaces in simulation: 1) TigerI 2) Danger Tag 3) RockSample 4) Navigation in partially known map 5) Multi-agent rock sample 6) Car driving among pedestrians. We compare the results of DESPOT-α with the state-of-the art online POMDP solver DESPOT for all these problems. We choose DESPOT as a baseline because only DESPOT has a parallel version available which can scale to very large problems and compute policy for time-critical task like car driving in just 0.1 seconds.

We compare HyP-DESPOT-α with HyP-DESPOT for last 3 problems as they are all large scale problems, with large state or action space along with large observation space. For all the experiments, we set $C = 10$ for (HyP-)DESPOT-α. Planning time for all the problems is fixed at 1 sec except for car driving problem for which it is 0.1 sec. From the results in Table I and II, we can see that DESPOT-α and HyP-DESPOT-α perform much better as compared to DESPOT and HyP-DESPOT.

Algorithm 3: BACKUP($\mathcal{D}, b$)

for *each node* x *on the path from* b *to the root of* $\mathcal{D}$ **do**
  if x *is not a leaf node in* $\mathcal{D}$ **then**
    Compute *new* α_x, $\underline{V}(x)$ using Eq. 16
    if *new* $\underline{V}(x) > \underline{V}(x)$ **then**
      $\underline{V}(x) \leftarrow new\ \underline{V}(x)$
      $\alpha_x \leftarrow new\ \alpha_x$
    end
    Compute *new* $\overline{V}(x)$, $AD(\zeta(b)).\overline{\alpha}^{pp}$ using Eq. 10. Update $\overline{V}(x)$, $AD(\zeta(b)).\overline{\alpha}^{pp}$ if $>$ *new* value.
  end
  UPDATESIBLING(x,y) for each sibling y of x.
end

Algorithm 4: EXPAND(b, $\mathcal{D}$)

foreach $a \in A$ **do**
  Add a new node (b, a) as child of b in $\mathcal{D}$.
  if $\zeta(b), a \notin AD$ **then**
    Collect P, Z, R by applying action a to each particle in $AD(\zeta(b)).P$. Compute α^{def},$\overline{\alpha}^{def}$, $\overline{\alpha}^{pp}$ using default policy for each particle in P
    $L(z, s') \leftarrow p(z|s', a)$ for each $z \in Z$, $s' \in P$.
    Compute η_{max} using Eq 18 and use it to reweight $L(z, s')$ for each $z \in Z$, $s' \in P$.
    $L(z_{res}, s') \leftarrow l(z_{res}|s', a)$ for each $s' \in P$.
    $Z \leftarrow Z \cup z_{res}$
    $AD(\zeta(b), a) \leftarrow < P, Z, R, L, \alpha^{def}, \overline{\alpha}^{def}, \overline{\alpha}^{pp} >$
  end
  $< P, Z, R, L, \alpha^{def}, \overline{\alpha}^{def}, \overline{\alpha}^{pp} > \leftarrow AD(\zeta(b), a)$
  foreach $z \in Z$ **do**
    Add a new node $\tau(b, a, z)$ as child of b, a in $\mathcal{D}$.
    $\alpha_{\tau(b,a,z)} \leftarrow \alpha^{def}$.
    Compute $w_{\tau(b,a,z)}$ using Eq 9 for each $s' \in P$.
    Compute bounds ($\underline{V}(\tau(b, a, z))$, $\overline{V}(\tau(b, a, z))$) by dot product of α^{def} and $\overline{\alpha}^{def}$ with $w_{\tau(b,a,z)}$.
  end
end

Results also show that HyP-DESPOT-α can leverage GPU parallelization better than HyP-DESPOT as we see a wider performance gap between HyP-DESPOT-α and DESPOT-α as compared to HyP-DESPOT and DESPOT. Next, we describe the test problems and discuss the results.

A. TigerI

In the original tiger problem, a tiger is hidden behind one of the 2 doors. Thus there are 2 states, Tiger Left (TL) and Tiger Right (TR). The robot can either open one of the doors or listen to the tiger. Opening a door with the tiger leads to a high penalty of -100. Opening the other door gives a reward of 10. The Listen action tells whether the tiger is behind the left door (Observation = TL) or the right door (Observation = TR) with some error. Thus there are 2 observations. The optimal policy is to listen before opening the door. The number of times one should listen depends on the accuracy of the listen action.

Algorithm 5: UPDATESIBLING(b_{trial}, b_{sib})

Compute $\underline{V}'(b_{sib})$ as dot product of $w_{b_{sib}}$ and $\alpha_{b_{trial}}$
if $\underline{V}'(b_{sib}) > \underline{V}(b_{sib})$ **then**
  $\underline{V}(b_{sib}) \leftarrow \underline{V}'(b_{sib})$
  $\alpha_{b_{sib}} \leftarrow \alpha_{b_{trial}}$
end
Compute $\overline{V}'(b_{sib})$ using SAWTOOTH Aprroximation with values $\overline{V}(b_{trial})$ and $AD(\zeta(b_{trial})).\overline{\alpha}^{pp}$.
Update $\overline{V}(b_{sib})$ if $\overline{V}'(b_{sib}) < \overline{V}(b_{sib})$.

For the version with a large observation space, we discretize a continuous observation z that is between 0 and 1 into N levels. To demonstrate the problem with information gathering when observation space is large, we define 2 listen actions: $L1$ and $L2$. $L1$ is less accurate and has reward of -1. $L2$ is more accurate but has reward of -1.2. We define:

$$p(z|s,a) = \begin{cases} \frac{g(z,s)}{\mathcal{N}(s,a)}, & \text{if } z \in [u(a), u(a)+\epsilon) \cup [l(a)-\epsilon, l(a)) \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

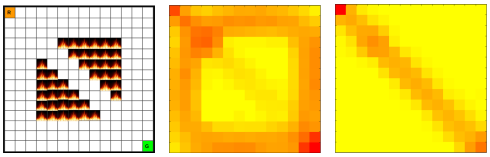
where $\mathcal{N}(s, a)$ is a normalizing constant. We set $g(z, s) = z$ for $s = TR$ and $g(z, s) = 1 - z$ for $s = TL$, i.e. if the tiger is on the right, larger values of z have larger probability, and if the tiger is on the left, smaller values of z have larger probability. The effect of different actions is controlled by the support of observations that have non-zero probability; more accurate action $L2$ allows non-zero probability for observations near 1 and 0, i.e. $u(L2) = 0.9$ and $l(L2) = 0.1$, while less accurate action $L1$ allows non-zero probability near 0.5, i.e. $u(L1) = 0.65$ and $l(L1) = 0.35$. For experiments, we set $\epsilon = 0.1$, $N = 100000$. The observation space size for each action is $2\epsilon N = 20000$. For DESPOT, $K = 500$ gives best results. For DESPOT-α, we set $K = 100$.

In DESPOT search tree, $L1$ and $L2$ will produce exactly same child belief nodes. Since reward for $L2$ is less than $L1$, value of $L2$ will always be lower than $L1$. Thus DESPOT will never choose $L2$, a better information gathering action, leading to a lower reward.

B. Danger Tag

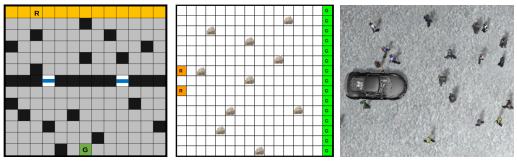
Danger Tag is a modified version of Laser Tag problem described in [19]. The robot has to tag the opponent but does not know its location. It can only estimate its position based on distance estimates in 8 directions from the laser sensor. Thus the observation is a set of 8 integers. For a 14×14 grid, the observation space is $\sim 10^8$. The robot can move in 8 directions in a map which contains some danger cells (See Fig. 2a). If it steps on danger cells, it gets a high penalty of -100, -1 otherwise. With a probability of 0.3, it can move either clockwise or anti-clockwise of the intended direction. It gets a reward of 100 on tagging the opponent correctly.

In this setting, the diagonal path is sub-optimal because of high probability of stepping on a danger cell. Fig. 2c shows that DESPOT still takes the diagonal path. This is because optimizing for each individual particle makes DESPOT think



(a) Danger Tag (b) DESPOT-α (c) DESPOT

Fig. 2: b) and c) show the frequency of robot positions during 500 runs. Red means higher frequency.



(a) (b) (c)

Fig. 3: Problems for evaluation of HyP-DESPOT-α. a) Navigation in a partially known map b) Multi-agent rocksample c) Car Driving among pedestrians

that it will be able to find a diagonal path without stepping onto danger cells, giving a high value to the diagonal path. When there are multiple particles in the belief node as is the case with DESPOT-α, using the diagonal path for one particle would result in some other particle stepping onto danger cells. This decreases the value of the diagonal path. See Fig. 2b.

C. Navigation in partially known map

This is similar to navigation problem in [4]. The robot has to navigate in a 13×13 map (See Figure 3a) with fixed (black cells) and unknown (grey cells) obstacles to reach the goal through one of the gates. The robot has uncertainty about its own position and the unknown obstacles in the map. It has 9 actions, one for movement in each direction and one STAY action where the robot does not move and only observes neighbouring cells. After each action, the robot can observe 16 surrounding cells (2 cells in each direction). The observation tells the robot whether the cell is occupied or not with 0.2 noise. Thus the observation space is 16 dimensional with a size of 2^{16}. STAY action has a penalty of 0.2 while move actions have a penalty of 0.1. Colliding with an obstacle results in a penalty of -1. Reaching the goal gives a reward of 20. This problem has a very large state space of 169×2^{124}.

This problem is different from the other problems because all the actions gather information equally and there is no policy which performs better only under uncertainty. Thus even though the observation space is large, particle divergence does not affect the policy much. To compute the optimal-policy, the key requirement is that the search tree is deep enough to find the goal during search. Thus DESPOT-α performs worse than DESPOT in this problem due to its additional computation overhead of expanding all the particles in nodes during trials. With a 60 step random action default policy, this requires lot of computation. In contrast, DESPOT can quickly search deeper for nodes with just one particle and find the goal. However GPU parallelization alleviates this computation overhead. HyP-DESPOT-α is able to perform even better that HyP-DESPOT because it leverages GPU parallelization better.

1) NavigationI: To see the effect of different information gathering actions, we set the observation noise of the STAY action to 0.002 and move actions to 0.4. DESPOT-α outperforms DESPOT in this setting. See NavigationI in Table I.

D. Rock Sample(RS)

Rock Sample(m,n) is a standard benchmark POMDP problem ([17, 19]). The robot needs to collect good rocks from n rocks which can be either good or bad while moving in a $m \times m$ map (see Fig. 3b). The robot has a sensor to check whether the rock is good or bad. The accuracy of information decreases exponentially with the distance of the rock from the robot. Sampling a good rock gives reward of 10, and sampling a bad rock gives a penalty of -10. The robot can exit the map from the East side and get a reward of 10. The number of actions is $n+5$, n CHECK actions, 4 move actions, 1 SAMPLE action. We make the observation space large in the same way as in Tiger problem with the support of the observations depending on distance of the robot from the rock which is being checked and ϵ as 0.1. We did experiments with 10, 15 and 20 rocks. m is 20 for 20 rocks and 15 for 10 and 15 rocks. We get best results at $K = 50$. DESPOT-α performs better than DESPOT (See Table II) because DESPOT makes the robot do the CHECK actions repeatedly instead of moving as particle divergence makes it think that the CHECK action allows it to identify all the rocks correctly.

1) Multi-Agent Rock Sample(MARS): As described in [4], MARS has 2 agents for collecting rocks. This makes action space size $(n+5)^2$. Thus we have very large $|A| = 225$, 400, 625 for 10, 15 and 20 rocks. Still both DESPOT-α and HyP-DESPOT-α perform better than DESPOT and HyP-DESPOT respectively. See Table I.

We also compare (HyP)-DESPOT-α running on large observation version with (HyP)-DESPOT running on the original version of the problem with only 2 observations for CHECK action ((HyP)-DESPOT-DIS). See Table III. We can say that (HyP)-DESPOT-DIS is using prior information to reduce the observation size to 2. Still, HyP-DESPOT-α which does not use any prior information is as good as HyP-DESPOT-DIS for MARS upto $|A| = 400$. There is only a slight difference at $|A| = 625$. DESPOT-α also performs as well as DESPOT-DIS for the single agent rock sample but worse for MARS. This is expected because of very high computational overhead with large number of actions.

E. Car driving among pedestrians

This problem is same as in [1, 4]. The robot car has to drive along a straight line among pedestrians (See Fig. 3c.) which can be going in two different directions. Speed of the robot is controlled by POMDP planner which can take 3 actions: ACCELERATE, DECELERATE, MAINTAIN. All pedestrians are assumed to be moving with same speed but have Gaussian noise in their heading direction. The car is equipped with LIDAR, which gives it an estimate of the positions of the pedestrians. However the goal of each pedestrian is unknown

TABLE I: Average total discounted reward achieved by (HyP-)DESPOT and (HyP-)DESPOT-α for large scale problems.

	Navigation	*NavigationI*	*MARS(15,10)*	*MARS(15,15)*	*MARS(20,20)*	*Car Driving*
$\|Z\|$	2^{16}	2^{16}	20000^2	20000^2	20000^2	$> 10^{65}$
DESPOT	6.16 ± 0.20	0.45 ± 0.20	17.31 ± 0.36	13.25 ± 0.43	12.52 ± 0.38	-8.52 ± 0.14
DESPOT-α	4.79 ± 0.20	2.08 ± 0.20	24.02 ± 0.31	16.98 ± 0.35	13.44 ± 0.33	-7.96 ± 0.09
HyP-DESPOT	6.88 ± 0.19	2.50 ± 0.22	23.61 ± 0.50	27.66 ± 0.41	20.01 ± 0.51	-7.92 ± 0.06
HyP-DESPOT-α	**7.76 ± 0.16**	**6.21 ± 0.18**	**42.43 ± 0.32**	**52.37 ± 0.31**	**49.18 ± 0.33**	**-6.70 ± 0.04**

TABLE II: Average total discounted reward by DESPOT and DESPOT-α for relatively small scale problems.

	TigerI	*Danger Tag*	*RS(15,10)*	*RS(15,15)*	*RS(20,20)*
$\|Z\|$	20000	$\sim 10^8$	20000	20000	20000
DESPOT	-3.96 ± 0.24	-21.99 ± 1.87	9.38 ± 0.21	4.80 ± 0.29	3.31 ± 0.11
DESPOT-α	**6.19 ± 0.23**	**-4.48 ± 0.79**	**32.72 ± 0.25**	**40.75 ± 0.41**	**36.43 ± 0.24**

TABLE III: Average total discounted reward by (HyP-)DESPOT-α for large observation (MA)RS and (HyP-)DESPOT for original (MA)RS with 2 observations.

	RS(15,10)	*RS(15,15)*	*RS(20,20)*
DESPOT-DIS	32.25 ± 0.25	39.29 ± 0.41	37.40 ± 0.21
DESPOT-α	32.72 ± 0.25	40.75 ± 0.41	36.43 ± 0.24
	MARS(15,10)	*MARS(15,15)*	*MARS(20,20)*
DESPOT-DIS	41.52 ± 0.25	31.87 ± 0.41	14.96 ± 0.35
DESPOT-α	24.02 ± 0.31	16.98 ± 0.35	13.44 ± 0.33
HyP-DESPOT-DIS	43.07 ± 0.34	51.09 ± 0.43	55.60 ± 0.39
HyP-DESPOT-α	42.43 ± 0.32	52.37 ± 0.31	49.18 ± 0.33

and has to be estimated by using the action-observation history. There is a very high penalty of -1000 on collision with pedestrians. Every step gets a reward of -0.5. Higher car speed and smooth driving is encouraged by giving some additional positive reward. For details, see [1]. The solution for this problem can be directly transferred to the real robot car driving among pedestrians as shown in [1, 4].

For this problem, in the true observation model, the pedestrian position is fully observed and particle weights are updated using a noisy transition model. During DESPOT-α forward search, we assume a deterministic transition model and update the weights using only the observation model. Therefore we cannot use true observation model and instead use a noisy observation model defined as a unit variance multivariate Gaussian with position of pedestrians in s as mean. 12 nearest pedestrians are considered. The belief update after executing a step is done using the true transition and observation model for both DESPOT and DESPOT-α.

The default lower bound policy used by DESPOT is not a fixed action policy and does not provide a valid lower bound for DESPOT-α. Therefore we use a different default policy which does DECELERATE if car speed is non zero and MAINTAIN after that. As car speed is fully observable, all the particles have same car speed in initial belief. Thus car speed summarizes $b_0, \zeta(b)$ and can be used to decide next action. Performance of (HyP-)DESPOT degrades with this default policy. Thus we do not change the default policy for (HyP-)DESPOT. (HyP-)DESPOT-α is able to outperform (HyP-)DESPOT which uses a better default policy, using only a fixed action default policy. See Table I.

TABLE IV: Performance comparisons of (HyP-)DESPOT and (HyP-)DESPOT-α on the autonomous driving task.

	Collision rate	Traveled distance
DESPOT (K=200)	0.001397 ± 0.00011	12.171 ± 0.05
DESPOT-α (K=100)	0.001344 ± 0.00010	14.948 ± 0.03
HyP-DESPOT (K=1000)	0.000431 ±0.00006	11.898 ± 0.07
HyP-DESPOT-α (K=1000)	**0.000291 ± 0.00004**	**15.490 ± 0.03**

HyP-DESPOT-α computes a policy which makes the car cover more distance while having a lower collision rate (number of collisions per meter). See Table IV. DESPOT-α also performs better than DESPOT, although the difference in performance is not substantial because DESPOT-α cannot plan with more than 100 particles in 0.1 sec. With 100 particles many collisions are not sampled during planning, leading to a higher collision rate. Beyond 100 particles, DESPOT is also not able to do many trials. Still it performs best at $K = 200$ as it mostly follows the very good hand designed default policy.

V. Conclusion

We have provided an online POMDP solver which can deal with both large state and observation spaces. Through our experiments, we have shown that our algorithm can generate significantly better policy when the observation space is large and can scale to complex real world problems which require very fast decision making and have complex dynamics. To the best of our knowledge this has not been demonstrated before.

VI. Future Work

Apart from sub-optimal policy computation due to particle divergence, another main issue in planning with very large observations like images, 3D maps etc. is the slow generation of observations which makes forward search prohibitively slow. DESPOT-α can possibly solve this issue as it requires only $p(z|s, a)$ instead of the actual observation during planning. In future we would like to explore if we can speed up forward search using this property. Also we would like to obtain theoretical performance bounds for our algorithm.

Acknowledgments

We would like to thank PanPan Cai for the insights and tips for GPU parallelization. This work is supported by MOE AcRF grant MOE2016-T2-2-068.

REFERENCES

[1] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee. Intention-aware online pomdp planning for autonomous driving in a crowd. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 454–460, May 2015.

[2] Haoyu Bai, David Hsu, and Wee Sun Lee. Integrated perception and planning in the continuous space: A POMDP approach. *I. J. Robotics Res.*, 33(9):1288–1302, 2014. doi: 10.1177/0278364914528255. URL https://doi.org/10.1177/0278364914528255.

[3] Dimitri P. Bertsekas and David A. Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, April 1999. ISSN 1381-1231. doi: 10.1023/A:1009634810396. URL http://dx.doi.org/10.1023/A:1009634810396.

[4] Panpan Cai, Yuanfu Luo, David Hsu, and Wee Sun Lee. Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. *CoRR*, abs/1802.06215, 2018. URL http://arxiv.org/abs/1802.06215.

[5] Edwin K. P. Chong, Robert Givan, and Hyeong Soo Chang. A framework for simulation-based network control via hindsight optimization. In *CDC 2000*, 2000.

[6] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *CoRR*, abs/1106.0234, 2011. URL http://arxiv.org/abs/1106.0234.

[7] Jesse Hoey and Pascal Poupart. Solving pomdps with continuous or large discrete observation spaces. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1332–1338, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1642293.1642505.

[8] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, May 1998. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X. URL http://dx.doi.org/10.1016/S0004-3702(98)00023-X.

[9] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1001–1007, Cambridge, MA, USA, 1999. MIT Press. URL http://dl.acm.org/citation.cfm?id=3009657.3009798.

[10] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2):193–208, Nov 2002. ISSN 1573-0565. doi: 10.1023/A:1017932429737. URL https://doi.org/10.1023/A:1017932429737.

[11] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *In Proc. Robotics: Science and Systems*, 2008.

[12] Andrew Y. Ng and Michael I. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 406–415, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9. URL http://dl.acm.org/citation.cfm?id=647234.719765.

[13] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025 – 1032, August 2003.

[14] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online planning algorithms for pomdps. *The journal of artificial intelligence research*, 32 2:663–704, 2008.

[15] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *NIPS*, 2010.

[16] Richard D Smallwood and Edward J Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations research*, 21(5):1071–1088, 1973.

[17] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 520–527, Arlington, Virginia, United States, 2004. AUAI Press. ISBN 0-9749039-0-6. URL http://dl.acm.org/citation.cfm?id=1036843.1036906.

[18] Trey Smith and Reid Simmons. Point-based pomdp algorithms: Improved analysis and implementation. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 542–549, Arlington, Virginia, United States, 2005. AUAI Press. ISBN 0-9749039-1-4. URL http://dl.acm.org/citation.cfm?id=3020336.3020402.

[19] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1772–1780. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5189-despot-online-pomdp-planning-with-regularization.pdf.

[20] Matthijs T. J. Spaan and Nikos A. Vlassis. Perseus: Randomized point-based value iteration for pomdps. *J. Artif. Intell. Res.*, 24:195–220, 2005.

[21] Zachary Sunberg and Mykel J. Kochenderfer. POMCPOW: an online algorithm for pomdps with continuous state, action, and observation spaces. *CoRR*, abs/1709.06196, 2017. URL http://arxiv.org/abs/1709.06196.

[22] Nevin L. Zhang and Weihong Zhang. Speeding up the convergence of value iteration in partially observable markov decision processes. *J. Artif. Int. Res.*, 14(1):

29–51, February 2001. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622394.1622396.

[23] Zongzhang Zhang, David Hsu, Wee Sun Lee, Zhan Wei Lim, and Aijun Bai. Please: Palm leaf search for pomdps with large observation spaces. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 249–258, 2015. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10491.