A Sampling Hyperbelief Optimization Technique for Stochastic Systems

James C. Davidson[1] and Seth A. Hutchinson[1]

Beckman Institute, University of Illinois, {jcdavdsn, seth}@illinois.edu

Abstract: In this paper we propose an anytime algorithm for determining nearly optimal policies for total cost and finite time horizon partially observed Markov decision processes (POMDPs) using a sampling-based approach. The proposed technique, sampling hyperbelief optimization technique (SHOT), attempts to exploit the notion that small changes in a policy have little impact on the quality of the solution except at a small set of critical points. The result is a technique to represent POMDPs independent of the initial conditions and the particular cost function, so the initial conditions and the cost function may vary without having to reperform the majority of the computational analysis.

POMDPs, optimal control, hierarchical, belief roadmap planner (BRP), stochastic motion roadmaps (SMR), sampling methods, hyper-particle filtering

1 Introduction

Uncertainty plays a dramatic role not only on the quality of the optimal solution of POMDP system, but also on the computational complexity of finding the optimal solution, with a worst case running time that is exponential in the length of the time horizon for the exact solution. However, given the importance of finding optimal or nearly optimal solutions for systems subject to uncertainty, numerous researchers have developed approaches to approximate POMDP systems to overcome this limitation (refer to [1, Ch. 15 & 16] for a survey of such approaches). The majority of these methods are for discounted, infinite-horizon problems. Moreover, many of these techniques must reperform all the computational effort when the objective function changes.

A central theme of almost all approximation techniques is to reduce the set of possibilities to be evaluated, whether simplifying the representation of the belief or by the simplifying the cost function. Drawing on insights offered in [2] about why belief sampling techniques (such as [3–8]) are so effective, we develop an alternative method that is inspired by sampling-based methods

(e.g., [9]). In [10], we introduce the notion of hyperfiltering, which evolves forward into future stages the probability function over the belief, or the hyperbelief. We refer to the space of probability functions over the belief as the hyperbelief space. Interestingly, the evolution of a system over the hyperbelief space is deterministic. Thus, we can find the optimal plan in the hyperbelief space using an approach derived from standard search techniques.

To do so, we abstract the problem into a two-level planner. At the high level, a set of points in the hyperbelief space is randomly generated. Edge weights between each ordered pair of samples are then generated using the lower level planner. The lower level planner commissions local, greedy feedback controllers to predict the evolution of the robot from one sampled hyperbelief point to another. Because of the stochastic, partially observed nature of the problem, hyperfiltering is used to estimate the future hyperbelief, under a given policy, from one stage to the next. Instead of requiring that each hyperbelief sample reach the target hyperbelief sample, the distance between the hyperbelief sample and the target sample is included with the edge information between each pair of samples.

At the cost of completeness, this graph representation reduces the set of possibilities to be explored from a continuum in an infinite number of dimensions to a finite set. Moreover, because the evolution is deterministic in the hyperbelief space, optimizing over the graph can be performed in worst case quadratic time complexity in the number of vertices in a complete graph using standard graph optimization algorithms–without the exponential explosion when planning in the belief space. Because the local policy may not attain the exact position of the target hyperbelief sample, a refinement algorithm is used to incrementally identify a possibly better policy and then simulate this policy to determine the actual cost and, thus, if the newly proposed policy is better. Once the initial graph is generated, a refinement algorithm is used, whereby the bound on the optimal value decreases with every iteration and will find the best solution for a given graph in a finite amount of time.

The presented method is motivated by the problem of minimizing the uncertainty of a robot at a goal position, wherein the robot is subject to state-and-control dependent uncertainty in the process and observation models. This implies not only minimizing the expected distance to the goal, but also the uncertainty in the representation of the probability function. Such an optimization function is often the implicit, if not explicit, goal of many robotic problems. SHOT, however, is much more general and is applicable to any hyperbelief dependent objective function.

The proposed method will be developed in Section 4. However, first related research (Section 2) is explored and background concepts (Section 3) are introduced. Examples are provided in Section 5, and we conclude with some final remarks and comments in Section 6.

2 Related research

Finding optimal policies for partially observable systems is intractable, with a best known computational time complexity that is exponential in both the time horizon and the number of observations [11]. Thus, many researchers have focused on finding efficient approximation methods (such as [3–8]). Each of these approximation techniques approximates the POMDP by reducing the the set possibilities that are evaluated to a subset of the complete set of possibilities required to find an exact solution–whether a limited subset of sample beliefs are evaluated or a limited subset of policies are searched.

Many approximation approaches are sampling-based. Sampling-based methods have become one of the dominant methods for planning in the robotics community (refer to [12, Ch. 7] for an overview and survey of sampling-based techniques). As described in [9], probabilistic sampling-based methods generate a series of samples in the configuration space and simple planners are used to link the samples (or vertices) together. In this way a roadmap (or graph) of the samples is created. The majority of sampling-based methods focus on finding feasible, but not necessarily optimal solutions. However, in [13], Kim et al. develop a technique to determine the optimal solution over the roadmap.

The concept of stochastic uncertainty in the process model was first introduced into sampling-methods in [14]. This method created a discrete approximation of a continuous space by creating a roadmap. Then, probabilities of transitions from one node to another are assigned. By abstracting the problem this way, Apaydin et al. are able to reduce a continuous Markov decision process system to a finite Markov decision process. This method focuses on determining feasible solutions, while recently in [15] the concept of optimizing over the roadmap was introduced via the stochastic motion roadmap, whereby Alterovitz et al. sample a set of points in the configuration space to generate the roadmap. Next, for each node in the roadmap they generate a random set of resulting states for a given action. They then associate an edge weight between the node and any other node according to the number of times, out of the total, that the given action resulted in reaching the other node. Expanding to POMDP's, Prentice and Roy in [16] generate a sampling based approach for linear Gaussian systems. In their approach, a set of mean samples are generated corresponding to points in the configuration space. Next, a traditional probabilistic roadmap is used to generate a roadmap of the system (without consideration of the uncertainty). They generate the transfer function to the belief samples and generate their respective covariances. Next they use a standard graph search (e.g., A^*) to search the graph in a forward manner, while generating the best estimate of the actual covariance as the search progresses.

3 Background

3.1 POMDP formulation

As the most general model of stochastic systems, POMDPs incorporate the possibility of incomplete and uncertain knowledge when mapping states to observations. Such a representation enables the modeling and analysis of systems where sensing is limited and imperfect.

POMDPs include at least the following components: the state space $\mathcal{X}$ representing the finite set of states of the world; the finite set of control actions $\mathcal{U}$ that can be executed; the transition probability function $p_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1}}$representing the likelihood of the system being in one state x_{k-1} and transferring into another state x_k at stage k given the applied action u_{k-1} at stage $k-1$; the set of all possible observations $\mathcal{Y}$; the observation probability function: $p_{\mathbf{y_k}|\mathbf{x}_k}$ describing the likelihood of a particular observation y_k occurring given the system is in a specified state x_k; and the cost function $c(\cdot)$, which defines the objective to be optimized for the POMDP.

3.2 Hyperfiltering

Hyperfiltering is a method for systems modeled by POMDPs to propagate the estimate of the belief and its uncertainty forward into future stages for unseen observations and unactualized control inputs. By choosing the probability function over the beliefs, hyperfiltering is able to sequentially evaluate the estimate of the system and its uncertainty forward from one stage to the next. Moreover, by adopting the complete representation of the uncertainty, instead of a limited number of statistics of the belief, a more accurate representation of the evolution of the system is obtained.

The evolution of the probability function $p(x_k|I_k)$ at stage k, also known as the belief b_k, can be determined given the previous belief b_{k-1} and an applied control action $u_k \in \mathcal{U}$ via the belief transition function:

$$b_k = B(b_{k-1}, u_{k-1}, y_k),$$

where the belief transition function describes the Bayesian filtering over all states $x \in \mathcal{X}$, or

$$\begin{aligned} &B(b_k, u_k, y_{k+1})(x_{k+1}) \\ &= \frac{p(y_{k+1}|x_{k+1}) \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k) b_k(x_k)}{\sum_{x_{k+1} \in \mathcal{X}} p(y_{k+1}|x_{k+1}) \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k) b_k(x_k)}. \end{aligned} \tag{1}$$

The notation $B(\cdot)(x_{k+1})$ is adopted to represent the resulting functional evaluated at a specific state x_{k+1}. The belief at each stage k resides in the belief space $\mathcal{P}_b$, which is the space of all possible beliefs. For discrete state POMDPs, the belief space is represented as an $|\mathcal{X}|-1$ dimensional simplex $\Delta^{|\mathcal{X}|-1}$, where $|\mathcal{X}|$ is the number of states in the state space.

When predicting future behavior, the observations are unknown and stochastic in nature. The future belief therefore becomes a random variable defined by the stochastic process:

$$\mathbf{b}_{k+1} = B(\mathbf{b}_k, u_k, \mathbf{y}_{k+1}). \tag{2}$$

The evolution from one stage to the next via the stochastic process (2) generates a random variable; thus, a representation of the probability function over the belief is needed to proceed.

The hyperbelief is a probability function over the belief space at each stage. The initial hyperbelief β_1 at stage $k = 1$ is given; for $k > 1$, the hyperbelief is defined as

$$\beta_k \triangleq p_{b_k|\beta_1,\pi}.$$

Each β_k is contained in the *hyperbelief space* $\mathcal{P}_\beta$.

The belief transition probability function $p(b_{k+1}|b_k, u_k)$, represents the probability of the outcome b_{k+1} of the stochastic process $B(b_k, u_k, \mathbf{y}_{k+1})$ given b_k and the applied control input u_k. Since both π and b_k are known, the probability function over the observations can be inferred. The function that transfers a hyperbelief $\beta_k \in \mathcal{P}_\beta$ into the hyperbelief $\beta_{k+1} \in \mathcal{P}_\beta$ given a policy $\pi \in \mathit{\Pi}$ is denoted as the hyperbelief transition function Υ, such that $\Upsilon : \mathcal{P}_\beta \times \mathit{\Pi} \to \mathcal{P}_\beta$, where $\mathit{\Pi}$ is the set of all information feedback policies. The hyperbelief transition function is represented as

$$\beta_{k+1} = \Upsilon(\beta_k, \pi)$$

where, for each $b_{k+1} \in \mathcal{P}_b$,

$$\Upsilon(\beta_k, \pi)(b_{k+1}) \triangleq \int_{b_k \in \mathcal{P}_b} p\left(b_{k+1}|b_k, \pi\left(b_k\right)\right) \beta_k(b_k) db_k.$$

The notation $\Upsilon(\cdot)(b_{k+1})$ is adopted to represent the resulting function evaluated at a specific belief $b_{k+1} \in \mathcal{P}_b$.

Due to the high nonlinearity of the hyperbelief transition function, we will approximate the hyperbelief transition function using the hyper-particle filter. When the belief transition probability function is chosen as the importance sampling function, the computational complexity of hyper-particle filtering is $O(Knq)$, where K is the desired time-horizon, n is the number samples representing the probability function over the belief space, and q is the number of samples representing a belief. Refer to [10] for a more thorough explanation of both hyperfiltering and the hyper-particle filtering approximation method.

4 Methodology

Our proposed method determines a nearly optimal policy for POMDP systems by approaching the problem from a two-tier, hierarchical approach, whereby

a graph is where the vertices represent sampled hyperbeliefs and the edges represent paths, which are generated by using local (closed-loop) policies, between ordered pairs of the hyperbelief samples. Before we outline this process in more detail, we will introduce the class of cost functions and policies that we consider in this approach.

4.1 Cost Function and Policy representation

As previously described, on of the beneficial aspects of hyperfiltering is that the evolution of the hyper-particle from one *future* stage to the next is deterministic, similar to how the evolution of the belief from one *previous* stage to the next is deterministic. Because the evolution of the system in the hyperbelief space is deterministic, the method we present performs a deterministic evaluation of the performance of the system. Thus, unlike typical approaches where the value function is the expected cost over the set of beliefs, we define the value function as a total sum of costs relative to the hyperbelief. We therefore can optimize the hyperbelief cost functions of the form $c : \mathcal{P}_\beta \times \Pi \rightarrow \mathbb{R}$, (where Π is the set of all feedback policies) and terminal cost $c_K : \mathcal{P}_\beta \rightarrow \mathbb{R}$. The value for a given initial hyperbelief β_0 is defined as

$$V(\beta_0) = \min_{\pi \in \Pi} \sum_{k=0}^{K-1} c(\beta_k, \pi) + c_K(\beta_K).$$

The formulation of this cost function is more general than state and action cost function (i.e., $c(x_k, x_{k-1}, u_k)$, which may depend on the states x_k and x_{k-1} and control action u_k) that is used in most POMDP optimization approaches. The generality of this cost function enables a much richer class of probability functions to be represented. For instance, a cost function that depends on the hyperbelief itself may penalize the possibility of certain beliefs being in distant from one another. Such a cost function could be used to minimize the amount of uncertainty present in the system as it evolves.

In this hierarchical approach, we restrict the lower level to be a local, greedy policy within some class of local feedback policies Π_l. A digraph $G =< N, E >$ is constructed, where each vertex is associated with hyperbelief sample that is both the source and target for a set of greedy policies. The upper level policy $\gamma(\cdot)$ selects the local policy based which edge in G is being traversed and on the current stage to produce a closed-loop policy, such that $\gamma : \mathcal{P}_\beta \rightarrow E$. The complete policy is a switching-based controller (see [17] for an overview of hybrid/switching-based methods), whereby the local policy being executed is determined based on some switching surface. In our formulation the switching surface is defined as the inflection point of the derivative of the distance measurement from the current hyperbelief to the target hyperbelief as specified by the sink of the current edge. At the point that the derivative becomes positive, the controller switches to the next edge specified, whereby the sink in the previous edge is the source in the new edge.

The set of all possible switching policies is denoted as Γ. We therefore can represent the value function for a given system as

$$V(\beta_0) \approx \min_{\gamma \in \Gamma} \sum_{k=0}^{K-1} c(\beta_k, \pi_{\gamma(\beta_k)}) + c_K(\beta_k).$$

When we derive the optimal solution *over the graph*, we will have to take into consideration the fact that each initial hyperbelief sample may not be able to reach each target hyperbelief sample. To address this issue we will bound, from above and below, the cost based in the distance from terminal hyperbelief to the target hyperbelief sample. Thus, we assume that in addition to the cost function $c(\cdot)$, both a function giving an upper-bound $\bar{c}(\cdot)$, and a lower-bound function $\underline{c}(\cdot)$ are provided so that $\underline{c}(\beta, \pi) \leq c(\beta, \pi) \leq \bar{c}(\beta, \pi),\ \forall \beta \in \mathcal{P}_\beta, \pi \in \Pi$. One possibility is to select Liptshitz continuous cost functions, whereby there exists a linear bounds between any two points, such that an upper and lower linear bound can be established for the cost based on the distance.

Algorithmically, SHOT proceeds in two steps: 1) generating the digraph $\mathcal{G}$ and 2) optimizing over the digraph $\mathcal{G}$ (or hyperbelief space roadmap) and then refining the results. Both of these steps will be described in more detail in the following sections.

4.2 Generating the digraph

The first step begins by generating the vertices for the digraph $G =< N, E >$ (or roadmap), whereby each vertex i corresponds to a randomly sampled hyperbelief β^i. Next a complete graph is instantiated by generating all the edge information between each ordered vertex pair, which corresponds to a starting sample hyperbelief and a target sample hyperbelief. Planning between each starting and target hyperbelief sample is then performed to generate bounds on the cost and distance from initial hyperbelief sample and terminal hyperbelief sample.

Generating the vertices: sampling a random hyperbelief set

One of the stated attributes of the proposed method is the capability to generate an abstracted representation of the structure of the hyperbelief space that is independent of the cost function. This way, as with sampling-based methods in general, the majority of work can be performed offline. With relatively little computation expense, the nearly optimal policy for a variety of objectives can be determined based on the current status of the robot as well as changing the initial conditions such as the starting hyperbelief.

To achieve this, we generate a set vertices N, whereby each vertex in the vertex set is associated with a randomly sampled hyperbelief. For notational convenience we will label the vertices with the same label as the hyperbelief samples to minimize confusion.

Sampling is a difficult problem and an enormous amount of research has gone into sampling for PRM methods [12, Ch. 7]. The difficulties of effectively sampling the hyperbelief space could be exacerbated by sampling in the hyperbelief space because the hyperbelief space is infinite dimensional. However, if it is possible to define a metric on the belief space, we can impose a metric on the hyperbelief space so that no two hyperbeliefs are infinitely far apart, which enables a gradient found to perform the local search.

For example, if various measures of the uncertainty between beliefs such as the Jensen-Shannon divergence [18], earth mover's distance [19], or the like (see [18] for a catalog of probability distance functions) are defined over the belief space, then we can impose the probability metric over the hyperbelief space. The *probability metric* $PM(\cdot)$ between two hyperbeliefs is defined as the expected distance between the two hyperbeliefs, or

$$PM(\beta^i \| \beta^j) = \int_{b^i, b^j \in \mathcal{P}_b} d(b^i, b^j) p_{\beta^i, \beta^j}(b^i, b^j) db^i db^j.$$

The distance between any two hyperbeliefs, using the probability metric, is bounded by the maximum distance between any two beliefs in the belief space. Thus, any two hyperbeliefs are only a finite distance apart. Alternatively, if the configuration space has a defined metric, such as the Euclidean distance, then we can use this metric to impose a metric in the belief space and thus on the hyperbelief space.

An illustration of the process of generating belief samples to create a hyperbelief sample, which represents a vertex in the graph G, is illustrated by Figure 1, where a total of 9 hyperbeliefs are sampled. As is the case with most

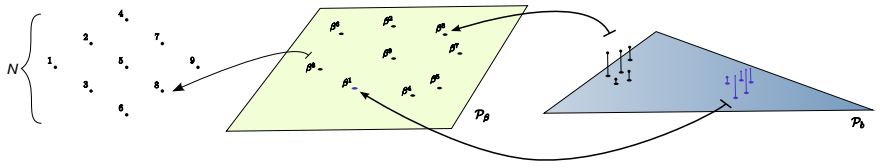


Fig. 1. Random set, N ,of hyperbelief samples in the hyperbelief space $\mathcal{P}_\beta$ and corresponding vertices

sampling-based methods, we anticipate that the sampling function will play a crucial role in the convergence properties of our method. One reason the choice of random sampling functions is crucial relates to the fact that a hyperbelief sample may not be reachable and so the distance attained by the local planner may be too great to be of use (i.e. the bounding function provides an excessively large cost). However, we will demonstrate later in Section 5 that even a naive sampling method performs well for the examples provided. Regardless, any number of alternative sampling techniques may be applicable from sampling-based research, such as the techniques described in [20,21]. Choosing more effective sampling functions is a future direction of this research.

Generating the edge information: planning between hyperbelief samples

After generating the sample hyperbelief set, which corresponds to the vertices in the graph G, the edge information is then generated. In our approach $\mathcal{G}$ is a complete digraph with an edge for every ordered pair of vertices. Each edge from the vertices represented by β^i to β^j is denoted as $i \to j$.

The edge information comprises the set of intermediary hyperbeliefs from the starting hyperbelief (corresponding to the source vertex) and the final distance from the target hyperbelief (corresponding to the sink vertex). Intermediate hyperbeliefs comprise the set of hyperbeliefs that represent the system as it evolves from the starting hyperbelief sample until the target hyperbelief sample is reached or the greedy planner can no longer make progress. These greedy policies, which are used to plan between hyperbelief samples, are meant to be guided by simple functions that only capture the value landscape locally. To determine the policy for an intermediate hyperbelief, we employ a policy that minimizes the cost of some value function $v_l : \mathcal{P}_\beta \times K \to \mathbb{R}$.

Once the local policy is selected, the next set of intermediate hyperbeliefs is generated. Starting for some initial hyperbelief sample β^i and some target hyperbelief sample β^j, corresponding to some vertex in G, the policy for the initial hyperbelief sample is determined and hyper-particle filtering performed to evolve the system forward one stage into the future to generate the following:

$$\beta_k^{i \to j} = \Upsilon(\beta_{k-1}^i, \pi_{_{i \to j}}),$$

where we denote $\beta_0^{i \to j} = \beta^{i \to j}$. This process repeats, generating a set of intermediate hyperbeliefs until a maximum number of iterations is exceeded or until the greedy policy can no longer make any progress towards the target hyperbelief sample.

This method is then performed for a specified number of neighbors for each vertex in the graph. The result is a digraph with intermediate hyperbeliefs. The distance from the terminal and closest hyperbelief from the target hyperbelief is illustrated in Figure 2. As can be seen in Figure 2, each vertex attempts to reach β^2 and once the greedy policy terminates, each edge has associated with it the distance from the final intermediate hyperbelief to hyperbelief β^2 as well as each of the intermediate hyperbeliefs (which is illustrated for β^4). The algorithm describing the complete graph generation (including both the hyperbelief sampling and the planning) is described in Algorithm 1.

4.3 Optimizing over the Graph

With the vertices and the edge information for the graph $\mathcal{G}$ determined, we can perform graph optimization on the directed graph to determine the optimal

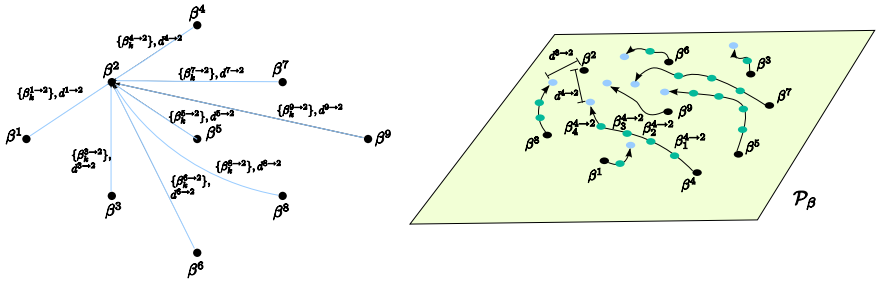


Fig. 2. Graph G and associated edge information directed to vertex β^2

Algorithm 1 Generate hyperbelief sample graph

Input $\mathcal{P}_\beta$: hyperbelief space, n: number of vertices, K: maximum number of iterations, π: greedy policy

Output G: digraph with vertices N, and edges E (includes edge information (e.g. weights))

$N = \{\beta^i\}_{i=1}^n \leftarrow$ randomly generate n samples from $\mathcal{P}_\beta$
for all $i \in N$ **do**
  for all $j \in N$ **do**
    $\beta_0^{i \leftarrow j} \leftarrow \beta^i$
    min_dist $\leftarrow$ distance between $\beta_0^{i \leftarrow j}$ and β^j
    dist $\leftarrow$ -∞
    $k \leftarrow 1$
    while $k \leq K$ and dist $\leq$ min_dist **do**
      $\beta_k^{i \rightarrow j} \leftarrow \Upsilon(\beta_{k-1}^{i \rightarrow j}, \pi)$
      dist $\leftarrow$ distance between $\beta_k^{i \leftarrow j}$ and β^j
      if dist $\leq$ min_dist **then**
        add $\beta_k^{i \leftarrow j}$ to $E^{i \leftarrow j}$ edge information
      end if
      $k \leftarrow k + 1$
    end while
  end for
end for
$G \leftarrow N, E$

choice of edges to traverse. This set of ordered edges defines the higher level policy.

To begin, the weight for each edge is generated based on the intermediate hyperbeliefs. The total weight for a given edge is then computed as

$$v^{i \rightarrow j} = \sum_{k=0}^{K^{i \rightarrow j}} c(\beta_k^{i \rightarrow j}, \pi_{i \rightarrow j}) + c_K(\beta^j),$$

where $K^{i\to j}$ is the number of intermediate hyperbeliefs and where $\beta_0^{i\to j}$ is the starting hyperbelief sample. The terminal cost $c_K(\beta^j)$ is determined by performing simulations of each vertex in the graph to minimize the the terminal cost associated with each vertex. In this way, each vertex also has a terminating cost of $c_K(\beta^j)$. To use discounted cost functions, all that need occur is that the discount be applied to the value function for each edge and then as the evaluation of the total value proceeds, the value of each edge is multiplied by the discount factor raised to the total number of stages performed before reaching the said edge.

The value of each edge can be determined on-line for various cost functions without having to re-execute the graph edge algorithm. With the edge weights of the graph determined, we can now perform graph optimization to determine the nearly optimal policy.

Standard graph optimization

Local policies are used to plan between hyperbelief samples, so all that remains is to establish what is the optimal choice of hyperbelief samples to visit. To do this, we optimize the cost over the graph. However, as discussed in Section 4.2, each starting hyperbelief sample may not be able to attain a target hyperbelief sample using the applied greedy policy. To account for the error introduced in the edge weights on the graph, we utilize $\bar{c}(\cdot)$ to generate an upper-bound on the value function. Because, for some i and j in N, $v^{i\to j}$ is a linear sum of the cost of each hyperbelief, and we can derive bounds for each $i, j \in N$ on $\bar{v}^{i\to j}(\cdot)$, which represents the upper-bound and $\underline{\mathrm{v}}^{i\to j}(\cdot)$ as the lower bound on the value function of each node, respectively. Both bounds are a function of the distance between any hyperbelief and the source vertex of the edge. Thus, where $d^{j\to l} = d(\beta^{i\to j}_{K^{i\to j}}, \beta_0^{j\to l})$, the bounds on the value between path $i \to j \to l$, are given as

$$v^{i\to j} + \underline{\mathrm{v}}^{j\to l}(d^{j\to l}) \leq v^{i\to j\to l} \leq v^{i\to j} + \bar{v}^{j\to l}(d^{j\to l}).$$

We derive the graph optimization method based on Dijkstra's algorithm (refer to [22, Ch. 24]). The basic idea is to start with each vertex $i \in N$. Update the minimal value for that vertex as the cost $c_K(\beta^i)$. Then for each edge $i \to j \in E$, evaluate the minimum of upper and bound and the minimum on the lower-bound on the cost from vertex i to adjacent vertex j and update the cost of the edge. We then repeat this step again for the number of vertices. This method takes $O(|N|^2 + |E|)$.

This process is performed to determine both the least upper-bound cost and path, as well as the greatest lower-bound cost and path. We simulate the system using the least upper-bounded path on the graph because the resulting values are just a bound on the actual cost. The result is the actual cost of the specified policy, not a least upper-bound. With the actual result and a bound on the lowest cost for the system, we can provide a possible range that the

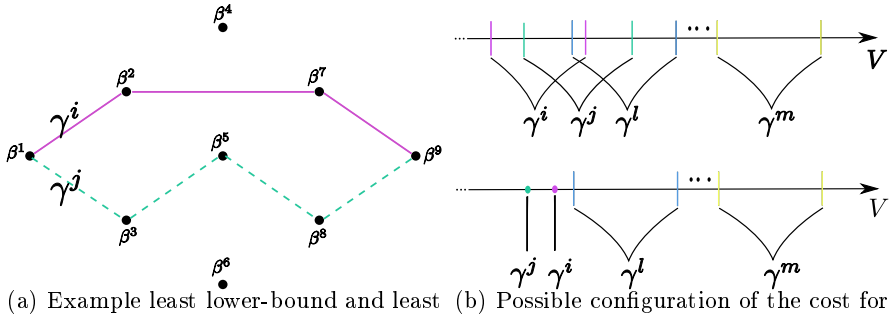
optimal solution may reside, which is the difference between the simulated cost and the greatest lower-bound cost.

One of the deficiencies of this approach is that the resulting policy and total cost are not necessarily optimal for the specified hyperbelief set. What is achieved is a bound on the optimal policy and cost. To overcome this shortcoming we present a refinement algorithm that will find the optimal solution for a given graph in a finite amount of time for the given hyperbelief samples.

Refinement

The quality of the bounding functions potentially play a critical role in producing a suitable solution in the previously outlined optimization technique. We present a refinement algorithm to mitigate the effects of the bounding function and also to guarantee the optimal solution will be found if enough time is given.

The process initiates just as the method without refinement. Then the optimization technique described above is performed to obtain both the least upper-bounded policy and the greatest lower-bounded paths. The first upper bounded path is depicted in Figure 3 as γ^i. Then the least upper-bounded



(a) Example least lower-bound and least upper-bound optimal switching policy

(b) Possible configuration of the cost for every possible policy

Fig. 3. Example least lower-bound and least upper-bound optimal switching policy

optimal path is simulated to achieve the actual value for the specified policy. In Figure 3a, policy γ^j is the second least upper-bounded path. After simulation the actual cost of γ^i is achieved. Then the refinement begins. The resulting cost is compared to the second to least upper-bounded cost. If the lower bound of second to lowest cost is above the actual cost of the already simulated cost, then the simulated cost is optimal and the method terminates. However, if the actual cost resides within the bound of this second to least upper-bound path, then the second to least upper-bounded path is simulated. The value of the resulting simulation is then compared to the previous lowest cost. If the newly simulated cost is the lowest then it is selected as the minimum.

This is in case in Figure 3b, where after simulation of γ^i, it is determined that the resulting cost is between the bounds of γ^j. Thus γ^j is simulated and it is discovered that γ^j achieves a lower cost than γ^i. The process then repeats with the third least upper-bounded path being checked. This process continues until a maximum number of iterations (less than or equal to the number of possible paths in the graph) or minimum threshold on the bounds of the cost is met. At any stage of this process, the bounds on the cost are evaluated as the minimum actual simulated cost to the least lower-bounded cost of the remaining (non-simulated) paths. The resulting minimum path is then the switching order for the switching policy. This method is described in more detail in Algorithm 2.

5 Results

To verify the SHOT technique, we applied it to several of the benchmark problems found in the literature: namely, $4x4$ and hallway2. Both of these methods are discounted infinite horizon problems. To simulate the system we adapted k to represent each stage of execution. Just as with generating a bound for the cost, we generate a bound on the number of iterations that elapse while traversing across edges of the graph. This way we can propagate the discount from one event to the next via the number of iterations that elapse between events.

For the presented examples, we generate upper bounds experimentally by measuring the sensitivity of the cost associated with the starting distance of each neighboring edge. For example, given edge $i \rightarrow j$, we determine the relationship of the cost for $v^{l \rightarrow i}$ relative to the distance of edge l to i for each l such that edge $l \rightarrow i$ exists. We then generate a piecewise linear, Lipshitz upper and lower bound for the cost. In a similar manner, bounds for the distance from one edge to the next were determined as well as the bounds on the number of elapsed iterations.

To generate the vertices of the graph we sample points in the configuration space and then push them through the transition and observation probability functions to generate hyperbelief samples. Undoubtedly this method of sampling is not ideal. In the future, we wish to implement a more intelligent sampling schemes.

The limited results produced so far seem to verify that that this method is comparable to other POMDP approximation methods. For the $4x4$ example with average of 18 hyperbeliefs, SHOT achieved an average cost of 3.703 with no variance in the 10 iterations run. The method HSVI2 presented in [6] achieved a reward of 3.75 for this example. With the he hallway2 example similar relative performance was achieved: 0.416, which is on par with the the expected reward 0.48 achieved by SARSOP [23]. The hallway2 example averaged 127.6 hyperbelief samples.

Algorithm 2 Refinement method

optimal_cost: the optimal cost found so far, optimal_policy: the current best policy, verified_policies: the list of polices evaluated so far, G: the hyperbelief event graph, initial_event: the initial hyperbelief, k: refine iteration
optimal_cost, optimal_policy,G, verified_policies
//Initial values generated optimal_policy←verified_policies ← Dijkstra(G, initial_event)
check_path ← verified_policies.end
while check_path is not empty **do**
  i ← Remove and return lowest cost vertex from check path
  for all neighbors j of i **do**
    value_list(j) ← $v^i.end + \bar{v}$(i,j)
  end for
  {value, index} ← find k'th lowest value in value_list
  append v^i ← value
  append $E \leftarrow i \leftarrow j$
end while
append verified_policies ← policy generated from initial_event starting with the the k'th lowest value
new_cost ← simulate system with new_policy
if new_cost < verified_cost **then**
  optimal_cost ← new_cost
  optimal_policy ← new_policy
end if
least_bound ←optimal_cost - least lower bound cost //which is generated by searching the graph and determining $\underline{v}$ and comparing against the verified_policies
if $k <$ maximum refine iterations and least_bound $\geq$ threshold **then**
  $k \leftarrow k + 1$
  {optimal_cost, optimal_policy, G, verified_policies} ← refine(optimal_cost, optimal_policy, cost_graph, verified_policies, G, initial_event, k)
end if
optimal_cost, optimal_policy, G , verified_policies

One point to note is that each of these problems comprise a set of states the once entered results in the system being uniformly distributed over the state space at the next iteration. This reset mechanism is pivotal in finding the optimal solution and if none of the hyperbelief sample are able to place some probability over this state, then the resulting policy fails to achieve the target and receives an inferior reward. This sensitivity is one of the motivating factors of behind the SHOT approach.

The examples presented were chosen to demonstrate the validity of the SHOT method. However, the presented method was developed with large observation spaces and planning for systems where the number of stages required to achieve a satisfactory result is prohibitive for most current POMDP approximation methods. To more fully address this class of problems and the performance of presented method, a more exhaustive set of experimental results for the SHOT method can be found at [24].

6 Conclusion

A method for finding nearly optimal policies for POMDPs with total cost or finite time horizons was presented. The proposed method (SHOT) was a sampling-based technique using a two-level hierarchical planner, whereby the lower level planner executes local, greedy feedback policies and the higher level planner coordinates the order of hyperbelief samples that are visited. This method attempts to capture the connectivity of the POMDP system, which is independent of the starting hyperbelief (or belief) and the objective function, so that an efficient multi-query technique can be utilized for any initial hyperbelief or objective function for a given POMDP system.

Future research includes evaluating alternate sampling schemes, such as generating event samples from the observation space. Sampling from the observation space has the potential to alleviate some of the issues that arise from attempting to sample from the hyperbelief space directly.

References

1. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2005.
2. D. Hsu, W. S. Lee, and N. Rong, "What makes some pomdp problems easy to approximate?," in *Advances in Neural Information Processing Systems* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), Cambridge, MA: MIT Press, 2008.
3. S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing Systems*, pp. 1064–1070, 2000.
4. J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for pomdps," in *In the International Joint Conference on Artificial Intelligence (IJCAI).*, 2003.
5. T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, (Arlington, Virginia, United States), pp. 520–527, AUAI Press, 2004.
6. T. Smith and R. Simmons, "Point-based pomdp algorithms: Improved analysis and implementation," in *Proc. of the Conference on Uncertainty in Artificial Intelligence*, July 2005.
7. M. T. Spaan and N. Vlassis, "A point-based POMDP algorithm for robot planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (New Orleans, Louisiana), pp. 2399–2404, Apr. 2004.

8. M. T. Spaan and N. Vlassis, "PERSEUS: Randomized point-based value iteration for POMDPs," in *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.
9. L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, June 1996.
10. J. C. Davidson and S. A. Hutchinson, "Hyper-particle filtering for stochastic systems," in *IEEE International Conference on Robotics and Automation*, 2007 (to appear).
11. L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, Jan 1998.
12. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.
13. J. Kim, R. Pearce, and N. Amato, "Extracting optimal paths from roadmaps for motion planning," *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 2, pp. 2424–2429 vol.2, 14-19 Sept. 2003.
14. M. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, J. Latombe, and C. Varm, "Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion," *Journal of Computational Biology*, vol. 10, pp. 257–281, 2003.
15. R. Alterovitz, T. Simeon, and K. Goldberg, "The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty," in *Proceedings of Robotics: Science and Systems*, (Atlanta, GA, USA), June 2007.
16. S. Prentice and N. Roy, "The belief roadmap: Efficient planning in linear pomdps by factoring the covariance," in *Proceedings of the 13th International Symposium of Robotics Research (ISRR)*, (Hiroshima, Japan), November 2007.
17. D. Liberzon, *Switching in Systems and Control*. Boston, MA: Birkhäuser, 2003.
18. A. L. Gibbs and F. E. Su, "On choosing and bounding probability metrics," *International Statistical Review*, vol. 70, pp. 419–435, 2002.
19. E. Levina and P. Bickel, "The earth mover's distance is the mallows distance: some insights from statistics," *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 251–256 vol.2, 2001.
20. C. Nissoux, T. Simeon, and J. Laumond, "Visibility based probabilistic roadmaps," *Advanced Robotics Journal*, vol. 14, no. 6, 2000.
21. M. Morales, S. Rodriguez, and N. M. Amato, "Improving the connectivity of prm roadmaps," in *IEEE International Conference on Robotics and Automation*, pp. 4427–4432, 2003.
22. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2002.
23. H. Kurniawati, D. Hsu, and W. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proc. Robotics: Science and Systems*, 2008.
24. J. Davidson, ""experimental results for the shot technique," 2008. [online]. available: http://robotics.ai.uiuc.edu/ jcdavid/shot."