Stochastic Local Search for POMDP Controllers

Darius Braziunas
Department of Computer Science
University of Toronto
Toronto, ON M5S 3H5
darius@cs.toronto.edu

Craig Boutilier
Department of Computer Science
University of Toronto
Toronto, ON M5S 3H5
cebly@cs.toronto.edu

Abstract

The search for finite-state controllers for partially observable Markov decision processes (POMDPs) is often based on approaches like gradient ascent, attractive because of their relatively low computational cost. In this paper, we illustrate a basic problem with gradient-based methods applied to POMDPs, where the sequential nature of the decision problem is at issue, and propose a new stochastic local search method as an alternative. The heuristics used in our procedure mimic the sequential reasoning inherent in optimal dynamic programming (DP) approaches. We show that our algorithm consistently finds higher quality controllers than gradient ascent, and is competitive with (and, for some problems, superior to) other state-of-the-art controller and DP-based algorithms on large-scale POMDPs.

1 Introduction

Partially observable Markov decision processes (POMDPs) provide a natural model for sequential decision making under uncertainty. Unfortunately, the application of POMDPs remains limited due to the intractability of current solution algorithms, especially those that use dynamic programming (DP) to construct (approximately) optimal value functions [16; 5]. One method for dealing with this bottleneck is to restrict the space of policies being considered, and devise techniques that search directly in that space. *Finite-state controllers (FSCs)* are the policy representation of choice in such work, providing a compromise between the requirement that action choices depend on certain aspects of observable history and the ability to easily control the complexity of policy space being searched.

While optimal FSCs can be constructed if no restrictions are placed on their structure [8], it is more usual to impose some structure that one hopes admits a good parameterization, and search through that restricted space. Among various techniques that search through a restricted policy space [12; 10; 18], *gradient ascent (GA)* has proven to be especially attractive because of its computational properties. GA has been applied to both the offline solution of POMDPs whose model is known [12] as well as the online reinforcement learning setting [13; 1]. We focus on known models.

One difficulty with gradient-based approaches, not surprisingly, is the ease with which they converge to local suboptima. Our experiences have demonstrated that GA, for example, has difficulty in problems where the *precise sequence* of actions taken is important for good performance. This is a common feature of stochastic planning problems to which POMDPs are often applied; they usually have different characteristics from navigational problems on which GA has often been tested. While various restrictions on policy space can be used to encode prior knowledge about a problem's solution [12], such restrictions may be hard to encode naturally, and such knowledge may be hard to come by.

In this paper, we describe an algorithm that searches for good controllers while remaining within the "local search" framework. Since finding an optimal fixed-size FSC is NP-hard [10; 12], we propose a *stochastic local search (SLS)* technique which, like GA, works in the space of FSCs, but uses very different heuristics to evaluate moves. *Belief-based SLS (BBSLS)* incorporates intuitions—used in the DP solution to POMDPs that work in belief-state value function space—that allow moves in different directions than those permitted by gradient-based methods. Specifically, BBSLS considers making moves that would be of high value when executed at some belief state, even though that belief state is not reachable given the current controller; since they do not improve controller value, such moves cannot be considered by GA. A tabu list is used to allow subsequent changes to the controller to adjust to this move. BBSLS is much less computationally intensive than DP methods, and provides a good compromise between full DP and the very restricted form of local search admitted by GA. Our empirical results suggest that our algorithm is competitive with recent state-of-the-art FSC and value-function-based methods such as BPI [15] and PBVI [14; 17], producing smaller, higher quality controllers, often significantly more quickly.

We begin in Section 2 with an overview of POMDPs. In Section 3 we first describe problems with GA that inspired our method, and then detail the BBSLS algorithm. Empirical results are provided in Section 4 that confirm various advantages over other algorithms, and we conclude with discussion of future research directions in Section 5.

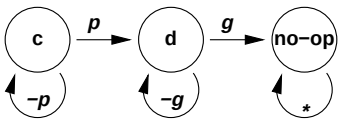

Figure 1: A simple FSC for a planning problem

2 Background and Notation

A POMDP is defined by: a set $\mathcal{S}$ of states; a set $\mathcal{A}$ of actions; a set $\mathcal{Z}$ of observations; a transition function T, where $T(s, a, s')$ denotes the probability $Pr(s'|s, a)$ of transitioning to state s' when action a is taken at state s; an observation function Z, where $Z(s, a, z)$ denotes the probability $Pr(z|s, a)$ of making observation z in s after performing a; and a reward function R, where $R(s, a)$ denotes the immediate reward associated with state s and action a. We assume discrete state,[1] action and observation sets, and we focus on discounted, infinite horizon POMDPs with discount factor $0 \leq \gamma < 1$. When considering FSCs for POMDPs, we will assume an initial *belief state* b_0 denoting the initial distribution over states (where $b_0(s)$ is the probability of the initial state being s).

Let H be the set of all finite *observable histories* (i.e., all finite sequences of action-observation pairs). A *stochastic policy* $\pi : H \mapsto \Delta(\mathcal{A})$ associates a distribution over actions with each history. A deterministic policy associates a single action with each $h \in H$. The *value* of policy π at state s is the expected sum of discounted rewards obtained by executing π starting at state s:

$$V_\pi(s) = E(\sum_{t=0}^{\infty} \gamma^t R^t | \pi, s).$$

The expected value of π is $b_0 \cdot V_\pi$ (viewing V_π as an $|\mathcal{S}|$-vector).

The representation of a policy π can take several forms. An indirect way of mapping histories into actions is to map *belief states* into actions: an agent's belief state comprises a sufficient statistic summarizing all relevant aspects of its history. An advantage of this approach is that instead of an infinite collection of discrete histories, one maps a continuous space $\Delta(\mathcal{S})$ into $\mathcal{A}$. Indeed, this mapping has nice properties and finite-horizon approximations of the optimal value function are piecewise linear and convex [16]. Unfortunately, methods that produce such representations (e.g., the Witness algorithm [4]) are often intractable, though approximation techniques like PBVI [14] can be used.

An FSC offers a different way of representing a policy π. A stochastic FSC $\langle \mathcal{N}, \psi, \eta \rangle$ comprises: $\mathcal{N}$, a finite set of *nodes*; ψ, an *action selection* function, where $\psi(n, a)$ denotes the probability with which $a \in \mathcal{A}$ is selected when the FSC is at node $n \in \mathcal{N}$; and η, a *transition function*, where $\eta(n, z, n')$ denotes the probability with which the FSC moves to node n' given that it was at node n after making observation $z \in \mathcal{Z}$. A *deterministic* FSC has deterministic action selection and node transition functions. While an FSC of a fixed size cannot generally represent an optimal policy, it offers a concise representation that can be exploited computationally. Figure 1 illustrates a 3-node, deterministic FSC. For each node n, ψ dictates which action to take (stochastically) and, conditioned on the observation, η prescribes which node to move to next. We sometimes refer to η as an *observation strategy*. Later, we will also use the notion of a deterministic *conditional plan* $\sigma = \langle a, \nu \rangle$, where $a \in \mathcal{A}$ is an action to execute, and $\nu : \mathcal{Z} \mapsto \mathcal{N}$ is a deterministic observation strategy.

The original POMDP and an FSC π induce a Markov chain whose states $\langle s, n \rangle$ are drawn from the cross-product $\mathcal{S} \times \mathcal{N}$. The value $V_\pi(s, n)$ of such a policy at $\langle s, n \rangle$ is:

$$V_\pi(s, n) = \sum_a \psi(n, a) R(s, a) + \gamma \sum_{a,z,s',n'} \psi(n, a) T(s, a, s') Z(s', a, z) \eta(n, z, n') V_\pi(s', n').$$

V_π can be computed by solving this linear system. The value of π given b is $V_\pi(b, n) = \sum_s V_\pi(s, n) b(s)$, and for an initial b_0, the best initial node n_0 can be determined readily.

One can find good or optimal FSCs in several different ways. Hansen's policy iteration [8] uses DP to produce a sequence of (generally, increasingly large) FSCs that converge to optimality. *Bounded policy iteration (BPI)* [15] uses basic DP together with techniques for bounding the size of the FSC, resulting in an approximation method that scales much better than policy iteration. Various search techniques have been proposed as well that do not ensure optimality, but often yield good results in practice without the computational cost of policy iteration. Among these gradient ascent has proven very popular. We refer to Meuleau *et al.* [12] for details of a typical formulation; in that work, an FSC of fixed size is assumed and the gradient of $V_\pi(b_0, n_0)$ w.r.t. the policy parameters $\eta(n, z, n')$ and $\psi(n, a)$ is derived. Standard GA is then used to navigate through the space of bounded FSCs until a local optimum is found. Computational results on maze-like problems [12] and continuous navigation-style problems [1] suggest that GA can tackle problems that are generally considered unsolvable using exact DP techniques.

3 A Stochastic Local Search Technique

We now describe a new approach for solving POMDPs approximately using FSCs, based on the use of stochastic local search and a specific heuristic, that circumvents some of the difficulties of GA. We begin with a simple example that illustrates a common type of local optimum to which GA falls prey, and describe intuitions that would allow a local search technique to break out of such local optima. We then formalize these intuitions within our algorithm.

3.1 A Motivating Example

Consider a simple planning problem in which the optimal solution consists of performing action c until the precondition p for action d is observed, then performing d until goal g is observed, and finally terminating (perhaps repeating some no-op).[2] Suppose further that: actions c and d are very costly, but

[1] Below, we will generalize our approach to continuous state POMDPs through the use of sampling.

[2] We assume the actions are stochastic and observations noisy. For instance, when c is performed, it is not guaranteed to make p

the reward associated with the goal g more than compensates for their expected costs; d only achieves g (with reasonable probability) if p is true; p is only made true by c; all other actions (at any state) have costs/rewards that are small relative to the costs/rewards of c, d, and g.

The optimal policy for this POMDP can be represented using the simple, deterministic 3-node FSC shown in Figure 1. If the action space is large enough, a random instantiation of this FSC is very unlikely to be optimal.[3] Suppose we attempt to solve this problem using GA, starting from some initial FSC, and suppose no node selects action c or d with significant probability. In this case, GA has no hope of finding the optimal FSC. Since the probability of c being executed is small, the probability of p being true at any belief state reachable using the current FSC is small; hence, increasing the probability of d at any node will decrease controller value, preventing GA from moving in that direction. Similarly, since d is unlikely to be executed, the value of increasing the probability of c at any node is negative, preventing GA from moving in that direction. Indeed, the nature of this problem is such that GA will be forced to move *away* from the optimal FSC. The sequential nature of the problem, and the fact that optimal actions are *undesirable* unless their counterparts are in place, make the landscape very hard to navigate using (even stochastic) GA.

How could one avoid the difficulties GA faces on POMDPs of this type within the local search framework? Intuitively, action d would be considered useful at a belief state in which precondition p held. Unfortunately, since c is never executed, such a belief state is unreachable given the current FSC. However, it is easy to verify that action d is good at some belief state in the context of the current controller. More precisely, a conditional plan $\langle d, \nu \rangle$ installed at node n—where ν transitions to a terminal node if g is observed, and back to n otherwise—would have high value in any belief state where p is sufficiently probable. As we will see, identifying the usefulness of this plan at *some* belief state b is straightforward, requiring the solution of a simple linear program (LP). Our local search procedure will consider adjustments to the FSC of this type: if a plan has high value at some belief state b, even if it cannot be realized by the current controller, we will consider (stochastically) making that move (or adjusting the FSC parameters in that direction).

Of course, if we make this move by adjusting the parameters at node n toward plan $\langle d, \nu \rangle$, we decrease the value of the FSC. Should we subsequently resort to moving in a direction that improves FSC value, we would naturally want to "undo" this move. Hence moves of this type will be held on a *tabu list* [7] for some period of time. This allows the algorithm a chance to "catch up" to the move. Specifically, since the plan at node n has high value at belief states near b, by holding this node fixed, we give the FSC a chance to find a policy for the rest of the FSC that will induce this region of belief space at node n. In this example, by holding n fixed, the plan $\langle c, \nu' \rangle$ at node n'—where ν' transitions to node n if p is observed, and back to n' otherwise—will now look attractive (indeed, with n fixed, GA would move in this direction). In a sense, this process simulates the reasoning inherent in value or policy iteration over belief space.

In the next sections, we make these intuitions more precise.

3.2 Algorithm Structure

Our *belief-based SLS algorithm (BBSLS)* stochastically adjusts the parameters of the fixed-size FSC at each iteration based on one of two criteria. The *moves* it makes consist of "installing" a deterministic conditional plan $\langle a, \nu \rangle$ at a node n: when such a move is made, the parameters of probabilistic FSC functions ψ and η at node n are adjusted in the direction of the plan $\langle a, \nu \rangle$ to make action a and transitions dictated by ν more likely. At each step, our SLS algorithm performs one or more *local* moves, followed by a sequence of *global* moves.

Local moves are designed to capture the basic intuitions described above, allowing BBSLS to break out of the types of local optima to which GA often falls prey. Intuitively, DP-inspired moves are considered, allowing policy choices to be made at unreachable belief states. By holding these moves on a tabu list, the rest of the controller is given a chance to "adjust" to these moves, by making these belief states reachable in order to attain higher value. Global moves correspond to direct stochastic hill-climbing, and are designed to increase controller value immediately, often taking advantage of earlier local moves. Thus, generally, in the local phase, we optimize for good, but potentially unreachable, belief states; in the global stage, we greedily improve the FSC value and make some of the belief regions considered in the local phase reachable.[4] Since the set of local and global moves is enormous, we need good ways of focusing on potentially useful moves. In each case, different techniques will be used. We now describe both phases in detail.

3.3 Local Moves

We first develop a heuristic for evaluating conditional plans. Let π be a fixed FSC. The Q-function Q^{σ}_{π} for a (deterministic) conditional plan $\sigma = \langle a, \nu \rangle$ is:

$$Q^{\sigma}_{\pi}(s) = R(s,a) + \gamma \sum_{\substack{s', n', \\ z \mid \nu(z) = n'}} T(s,a,s') Z(s',a,z) V_{\pi}(s',n').$$

Intuitively, this is the expected value of performing a at s, moving to the controller node dictated by the resulting observation, then executing the controller from that point on. We define $Q^{\sigma}_{\pi}(b) = \sum_s Q^{\sigma}_{\pi}(s) b(s)$ for any belief state b.

Our aim is to rank plans σ as possible moves in controller space according to a heuristic function $h(\sigma)$ that reflects *potential* value at belief states that might not be reachable from b_0 in the current FSC. Let Σ_l be the set of plans from which we select local moves. For each $\sigma \in \Sigma_l$, we find a belief state

true; and detecting that p is true is also noisy. Intuitively, the optimal policy would choose to execute d only if p is believed with sufficiently high probability; we assume that a single observation of p makes this so.

[3] To keep things simple, we focus on a small two-step sequence; for longer sequences, typical of planning problems, the odds of a random FSC including any significant subsequence are negligible.

[4] BBSLS can be viewed as a form of *iterated local search* [9].

b^σ such that the *difference* between the Q-value of σ and the value of any current controller node is maximal. Define

$$\delta_\pi^\sigma = \max_b [Q_\pi^\sigma(b) - \max_n V_\pi(b, n)],$$

and let b^σ be a belief state that maximizes this expression. The heuristic value of σ is then simply δ_π^σ, i.e., the maximal possible improvement over the current controller value (achieved at "witness" belief state b^σ). δ_π^σ can be computed by solving an LP with $|\mathcal{S}| + 1$ variables and $|\mathcal{N}|$ constraints.

With this heuristic in hand, local moves can be chosen. We first note that any plan σ has the same Q-function Q_π^σ regardless of the node at which it is installed. Because of this we break local move choice into two stages: plan evaluation (which plan σ to install) and node selection (at which node to install the plan). The set of all conditional plans is generally too large to evaluate—we cannot compute $h(\sigma)$ for each $\sigma \in |\mathcal{A}||\mathcal{N}|^{|\mathcal{Z}|}$ for any but the smallest problems. Therefore, we restrict the set Σ_l by using the Witness algorithm [4] to incrementally generate a (sub)set of *useful* plans that would improve controller value if we were to increase its size. We evaluate only such plans, and stochastically choose some σ using a distribution that gives greater weight to plans with greater h-values. The distribution we use in our experiments is straightforward: since all δ_π^σ are positive, we simply normalize them to sum to one, and sample from the resulting probability distribution.

We place one other restriction on the choice of plan: no σ is chosen whose witness belief state b^σ is "near" the witness belief state of an existing node. If a new plan is chosen that has high value at some belief state near another for which a previously selected plan has high value, installing the new plan at a node will "waste" controller capacity by duplicating the function of the earlier plan. For this reason, we maintain a *belief tabu list* containing the witness belief states of the most recently selected plans. The distance between two belief states can be defined in a variety of ways. For our experiments, we used the belief discretization technique of Geffner and Bonet [6], but more suitable measures warrant further research.[5]

Finally, the selected σ is *installed* (see above) at some node n which is not on the *node tabu list*. Among non-tabu nodes, a node n is randomly selected which is either unreachable from n_0 or leads to the greatest increase in value $b_0 \cdot V_\pi$. The latter choice directly increases controller value, while the former exploits unused controller capacity. The witness for σ is associated with n, and n is added to the tabu list.

Intuitively, in the local stage, we find plans that have high value at some belief states, even though they might not be reachable in the current FSC. Since the LP that computes $h(\sigma)$ also returns the witness b^σ, we record this as well. This information allows us to rule out other subsequent moves that duplicate the effect of σ (which would waste controller capacity). It is also important to exploit unreachable nodes, since this helps avoid unused capacity; moves at unreachable nodes, since they have no impact on FSC value, would never be considered by GA (or in our global stage). Once a useful σ has been installed at an unreachable node, there is incentive at subsequent iterations to link to this node.

[5]Given an integer resolution $r > 0$ (we set $r = |\mathcal{S}|$ in our experiments), the probabilities $b(s)$ are discretized into r levels. Two belief states b, b' are close if their discretized representation is the same: $round(b(s) \cdot r)/r = round(b'(s) \cdot r)/r, \forall s \in \mathcal{S}$.

3.4 Global Moves

In the global stage, we select moves that increase the overall controller value with respect to the initial belief state b_0. As with local moves, we consider a subset of deterministic plans Σ_g as possible candidates. Each $\sigma \in \Sigma_g$ is evaluated based on its improvement in FSC value $b_0 \cdot V_\pi$ (much like GA) and the moves are chosen with probability related to their level of improvement. Since the objective of a global move is to increase controller value, we build the set of possible plans for consideration as follows. First, we simulate the controller to obtain a sample of *reachable* belief states and nodes.[6] For each belief state, we compute the best plan for that specific belief state (this can be done in time linear in controller size), and calculate the value of the controller that would result if the plan was installed at the corresponding node. We repeat as long as controller value increases.

The global stage is essentially a form of stochastic hill-climbing. While local moves instantiate nodes with plans that are useful for *some* belief states, we ultimately care about FSC value (w.r.t. b_0). Therefore, global moves are used to increase value. Such moves often link to the nodes instantiated with potentially useful plans in the local stage. In a sense, the global stage is used to verify the usefulness of moves proposed in the local stage.

3.5 The BBSLS Algorithm

The BBSLS algorithm is summarized in Table 1. At each iteration, the BBSLS algorithm executes one or more local moves, and a sequence of global moves. The parameters of the algorithm can be chosen to reflect problem structure. If there are many strong local basins of attraction, the number of local moves l could be increased to facilitate an escape from local suboptima. Different approaches can be used to generate the set of candidate local moves Σ_l (the witness method we use, described above, appears to work well).

The condition C we use to terminate the execution of global moves is lack of improvement in controller value (as discussed above), though other conditions may be useful (e.g., a fixed number of steps). The parameters k and m in the global stage could be tuned based on the number of FSC nodes, POMDP states, and the discount factor. However, the default parameter values described in the next section proved to be adequate for our experiments. Finally, we have encoded the specific method of sampled reachable belief states within the algorithm as the method of constructing Σ_g; again, other ways to restrict the space of candidate moves could prove useful in specific settings.

BBSLS is an anytime algorithm. Empirically, because the global stage is essentially greedy optimization over reachable belief states, BBSLS quickly achieves (and exceeds) the performance of GA. The local stage runs in time polynomial in $|\mathcal{N}|, |\mathcal{S}|, |\mathcal{A}|, |\mathcal{Z}|$, and $|\Sigma_l|$, and the global stage

[6]This is done using k runs of the controller (starting at b_0, n_0, randomly sampling transitions and observations) of m steps each.

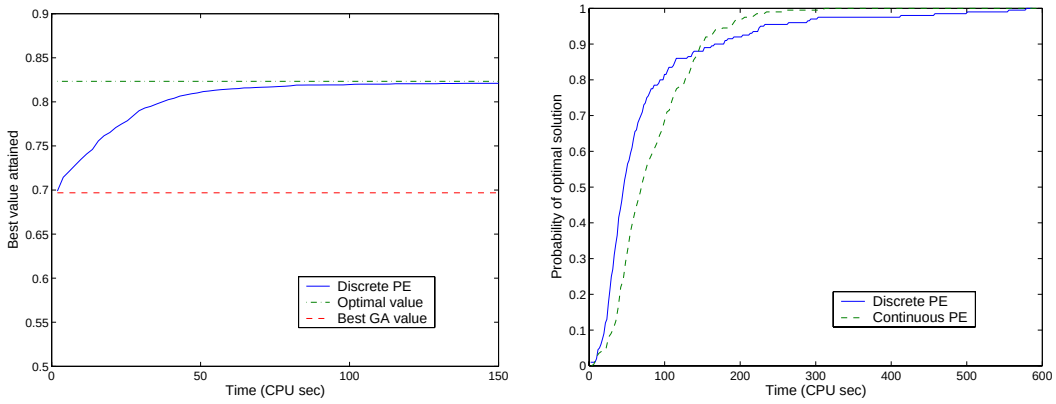

Figure 2: (a) Anytime performance of discrete PE; (b) RTDs for discrete (2.0 s/iter) and continuous (3.2 s/iter) PE.

While some search termination criterion is not met:

- Perform l local moves:
  - create a set of candidate condition plans Σ_l;
  - *sample* a conditional plan $\sigma \in \Sigma_l$ according to the Q-value heuristic $h(\sigma)$ (plans with higher h-values are given greater weight in the sampling distribution) while ensuring that no node in the FSC already has a witness belief state b^σ;
  - randomly choose a non-tabu node n which is either not reachable from n_0 (with probability p_l), or which leads to the highest increase in the FSC value when instantiated with the plan σ (with probability $1 - p_l$);
  - execute the local move $\langle n, \sigma \rangle$, add the node n to the move tabu list and the witness belief state b^σ to the witness belief list.
- Perform global moves (until condition C met):
  - run k policy execution simulations for m steps, starting from the initial node n_0 and belief state b_0, and record the belief states and nodes that were reached;
  - create the set of conditional plans that are optimal at the belief states visited during the previous step, and calculate the value (w.r.t. to the initial belief state b_0) of the controller that would result if a conditional plan from the set was installed at the corresponding node;
  - choose a move that leads to the highest increase in controller value. Make the selected move, and remove the witness belief state ascribed to the associated node.

Table 1: The BBSLS algorithm.

in time polynomial in $|\mathcal{N}|, |\mathcal{S}|, |\mathcal{A}|, |\mathcal{Z}|$, and $|\Sigma_g|$. Since $|\Sigma_l|, |\Sigma_g|, |\mathcal{N}|$ are controllable parameters, we can usually achieve a very good trade-off between solution quality and computation time.

4 Empirical Results

The following experiments illuminate various aspects of BBSLS and compare its performance to GA, PBVI, and BPI on examples drawn from the research literature.[7] Default algorithm parameters were: tabu list size equal to $|\mathcal{N}|/2$, $l = 1$, $p_l = 0.5$, $|\Sigma_l| = 10$; $k = 2$, $m = |\mathcal{N}|/2$. We used deterministic controllers on all problems except for Heaven/Hell. Generally, better performance can be achieved if parameters are tuned to specific problems.

The need for sequential policy structure is clearly evident in the small preference elicitation problem (PE) described in [2]. The objective is to optimally balance the cost of queries and the gain provided by the elicited utility information with respect to the quality of the final decision. We refer to [2] for a specification of the problem.

We tackle two variants of this problem. In the first, we discretize state space (possible utility functions) to six states and the number of actions (queries) to 14. In the second, the state space remains continuous. In both cases, an optimal FSC has 12 nodes. Good performance requires that a precise sequence of actions (queries) be executed before making a final decision. However, a default decision exists as a safe alternative for any belief state. Since asking queries before making a decision is initially costly, GA always converges to a safe, but suboptimal alternative. On the other hand, BBSLS is designed to avoid such local optima; for this small problem, it always finds the best global solution (which can be computed analytically). Figure 2 (a) plots the average value of the best 22-node FSC found by BBSLS (200 trials) and compares it to

[7] BBSLS was implemented in Matlab and run on Xeon 2.4GHz computers; linear programs were solved using CPLEX 7.1; GA used the quasi-Newton method with BFGS update and quadcubic line search from Matlab's Optimization Toolbox.

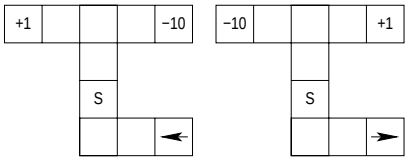

Figure 3: Heaven/Hell problem

the best GA value (measured on 10,000 random initial FSCs, each trial taking 3 seconds on average).

To solve a continuous POMDP with BBSLS, we *sample* u states (utility functions) at each iteration (we set $u = 20$). We then calculate the observation function and the reward function for the sampled states. Although our work with continuous elicitation is at an early stage, the results are promising—because of added stochasticity, BBSLS finds optimal controllers even faster than in the discrete case. Figure 2 (b) shows *run-time distributions* (RTDs) for both the discrete and continuous versions of the PE problem, that is, plots of the empirical probability of finding an optimal FSC as a function of time.

In the Heaven/Hell (HH) POMDP [1; 6], an agent starts with equal probability in either the "left" or "right" worlds, and, because of partial observability, does not know which. The (left or right) arrow (Figure 3) conveys information about the location of "heaven" (positive reward); if the agent does not observe the arrow, it risks falling into "hell" (negative reward). After visiting heaven or hell, the agent is again placed in either the left or right world. It can observe the top and bottom cells in the center column, as well as non-zero reward locations. In [1; 6], both heaven and hell have symmetric rewards of +1 and -1. This allows GA methods some flexibility, since moving down from the start state (and acting randomly) is no worse than moving up (in expectation); there is no disincentive to adjusting a controller to move upward before the correct policy for the top part of the maze is discovered. Our experiments with 20-node controllers show that on symmetric HH, GA sometimes reaches a local optimum with value 0.8040 (100 trials, each 2.9 min on average). However, the optimal value is 8.641, achievable by an 8-node FSC. Furthermore, we can make this problem practically unsolvable by GA if we increase the hell penalty to -10 (Figure 3). GA always chooses (in 100 trials) the safe alternative (bumping into walls for zero reward), even though the optimal FSC has not changed. Our SLS procedure (stochastic FSC, $l = 2$, 12 trials) finds a near-optimal solution even in the asymmetric case: in 200 iterations (average time of 10.2 seconds per iteration), the average value attained is 6.93, and in 500 iterations an average value of 7.65 is achieved.

Hallway (60 states, 5 actions, 21 observations) and Hallway2 (92 states, 5 actions, 17 observations) are two domains of moderate size [11]. In Hallway, a 10-node FSC attains a value of 0.7 in 442 seconds, and eventually achieves 0.8 in 500 iterations (averaged over 13 trials). The best value reached by a 30-node FSC was 0.95. The best value reached by a 20-node FSC for Hallway2 problem was 0.34. For comparison, Q-MDP finds a policy with a value of 0.3444 for Hallway, and 0.097 for Hallway2 [3]. Our results are not directly comparable to those reported for recent state-of-the-art algorithms PBVI [14] or BPI [15] (due to different testing methods). However, an indirect comparison seems to indicate that BBSLS achieves similar performance on both hallway domains with much smaller controllers.

Finally, we compare the performance of BBSLS to BPI and PBVI on the large Tag domain [14], with 870 states, 5 actions, and 30 observations. Tag is based on the popular game of laser tag, where the goal is to search and tag a moving opponent. On this problem, GA converges to a local suboptimum of -20 (10-node FSC), and cannot be run with controllers having more than 20 nodes because of time and space constraints. PBVI achieves a value of -9.18 in 180,880 seconds with a policy of 1334 linear vectors (roughly comparable to a 1334-node controller); BPI finds a 940-node controller with the same value in 59,772 seconds; and, a modified version of PBVI described in [17] can find a solution with a value of -6.17 in 1670 seconds with 280 vectors. BBSLS attains similar or better performance faster than PBVI and BPI, and with many fewer nodes than any other algorithm. Figure 4 (a) shows results for 10- and 30-node controllers, averaged over 10 trials; Figure 4 (b) plots how the best value found is increasing for a 10-node FSC during a single best trial; it reaches -6.70 after 322 iterations. Average BBSLS performance gives controllers of somewhat better quality than BPI in the same amount of time, but of much smaller size. However, since BPI attempts to optimize policies with respect to all possible belief states, rather than just the initial belief state (like PBVI and BBSLS), this is not too surprising. The more appropriate comparison to PBVI (since both optimize w.r.t. an initial belief state) shows much better performance for BBSLS in terms of "controller size," and comparable performance in terms of policy quality. With respect to execution time, BBSLS is competitive with the PBVI, but modified PBVI finds high quality policies significantly faster. However, the BBSLS controllers are an order of magnitude smaller than the "controller" found by modified PBVI.

5 Concluding Remarks

Despite its computational attractiveness, GA for FSC search suffers from problems when faced with genuinely "sequential" POMDPs. Our BBSLS algorithm is designed to retain many of the computational properties of GA by remaining within the local search framework, but uses DP-style reasoning heuristically to explore a more appropriate controller neighborhood. Our experiments demonstrate that BBSLS consistently outperforms GA, and is superior in many respects to DP-based algorithms, such as PBVI and BPI, on recent benchmark problems.

There are a number of important directions we are currently pursuing. Refining our heuristics to enable further scaling is key among these. In addition, we are exploring methods that will allow BBSLS to be applied directly to factored POMDP representation (e.g., dynamic Bayes nets) so problems with exponentially larger state spaces can be tackled. Finally, we are investigating the use of BBSLS in practical preference elicitation problems, where continuous state,

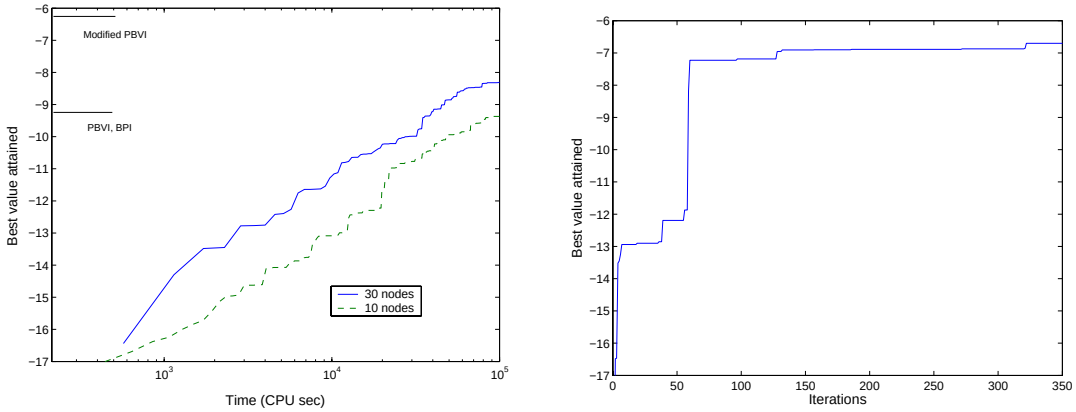

Figure 4: (a) Value vs. time for 10- and 30-node FSCs in Tag domain; (b) A specific trial, 10-node FSC.

action, and observation spaces present challenges to value-based POMDP solution methods.

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council (NSERC) and the Institute for Robotics and Intelligent Systems (IRIS). Thanks to Pascal Poupart for valuable discussions.

References

[1] Douglas Aberdeen and Jonathan Baxter. Scalable internal-state policy-gradient methods for POMDPs. In *Proc. of ML-02*, pages 3–10, 2002.

[2] Craig Boutilier. A POMDP formulation of preference elicitation problems. In *Proc. of AAAI-2002*, pages 239–246, Edmonton, 2002.

[3] Anthony R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Providence, RI, 1998.

[4] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proc. of AAAI-94*, pages 1023–1028, Seattle, 1994.

[5] Anthony R. Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact method for POMDPs. In *Proc. of UAI-97*, pages 54–61, Providence, RI, 1997.

[6] Hector Geffner and Blai Bonet. Solving large POMDPs by real time dynamic programming. In *Working Notes, Fall AAAI Symposium on POMDPs*, 1998.

[7] Fred Glover. Tabu search–part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[8] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Proc. of UAI-98*, pages 211–219, Madison, WI, 1998.

[9] Holger H. Hoos. *Stochastic Local Search—Methods, Models, Applications*. PhD thesis, TU Darmstadt, Darmstadt, Germany, 1998.

[10] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1994. The MIT Press.

[11] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proc. of ML-95*, pages 362–370, Lake Tahoe, 1995.

[12] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proc. of UAI-99*, pages 417–426, Stockholm, 1999.

[13] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Proc. of UAI-99*, pages 427–436, Stockholm, 1999.

[14] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. of IJCAI-03*, pages 1025–1030, Acapulco, 2003.

[15] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16 (NIPS-2003)*, Vancouver, 2003.

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

[17] Matthijs T. J. Spaan and Nikos Vlassis. A point-based POMDP algorithm for robot planning. In *IEEE International Conference on Robotics and Automation*, New Orleans, 2004. to appear.

[18] Marco Wiering and Juergen Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997.