Incremental Policy Iteration with Guaranteed Escape from Local Optima in POMDP Planning

Marek Grzes and Pascal Poupart
Cheriton School of Computer Science, University of Waterloo
200 University Avenue West, Waterloo, Ontario N2L 3G1, Canada
{mgrzes, ppoupart}@cs.uwaterloo.ca

ABSTRACT

Partially observable Markov decision processes (POMDPs) provide a natural framework to design applications that continuously make decisions based on noisy sensor measurements. The recent proliferation of smart phones and other wearable devices leads to new applications where, unfortunately, energy efficiency becomes an issue. To circumvent energy requirements, finite-state controllers can be applied because they are computationally inexpensive to execute. Additionally, when multi-agent POMDPs (e.g. Dec-POMDPs or I-POMDPs) are taken into account, finite-state controllers become one of the most important policy representations. Online methods scale the best; however, they are energy demanding. Thus methods to optimize finite-state controllers are necessary. In this paper, we present a new, efficient approach to bounded policy interaction (BPI). BPI keeps the size of the controller small which is a desirable property for applications, especially on small devices. However, finding an optimal or near optimal finite-state controller of a bounded size poses a challenging combinatorial optimization problem. Exhaustive search methods clearly do not scale to larger problems, whereas local search methods are subject to local optima. Our new approach solves all of the common benchmarks on which local search methods fail, yet it scales to large problems.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

Keywords

Planning under Uncertainty; POMDP; Policy Iteration; Finite State Controller

1. INTRODUCTION

We propose a new approach to solving partially observable Markov decision processes (POMDPs) when the policy—that is, the solution to a POMDP—is represented in the form of a finite-state controller (FSC) [11]. Our approach deals with the combinatorially challenging problem of finding an optimal or near-optimal controller of a small size [22]. It distinguishes itself from previous work by guaranteeing improvements until a globally optimal solution is found while scaling gracefully by incrementally growing the size of the controller one node at a time.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

Several factors can be identified to motivate research on finite-state controllers and the need for small controllers specifically. On one hand, research on efficient POMDP algorithms has advanced significantly in the past decade. The most widely known and esteemed approaches are based on point-based value iteration [20, 29], refinement of lower and upper bounds on the optimal value function [25, 14, 19], and online planning [24]. On the other hand, in several modern applications of POMDPs—especially those executed on mobile phones or wearable devices [18, 10]—a POMDP policy has to be energy efficient. Finite state controllers are both energy efficient and easy to implement on small devices, especially when a hardware implementation is needed. They generalize easily to multi-agent POMDPs such as decentralized POMDPs (Dec-POMDPs) [4] or interactive POMDPs (I-POMDPs) [7]. For instance, Bounded Policy Iteration (BPI) [22] has been successfully extended to both Dec-POMDPs [1] and I-POMDPs [27, 28]. Hence, any advances in single-agent POMDP planning using finite-state controllers could be useful for the multi-agent setting.

Early work on POMDPs considered policy iteration with an explicit finite-state controller representation [26, 11], and then subsequent research has shown many improvements [22, 2, 3, 17, 10]; however, point-based approaches are still considered by practitioners to be more robust in many settings, even though current research has shown that finite-state controller algorithms can learn smaller policies [23]. Also, when point-based algorithms are used, the resulting policy can be compiled into a finite-state controller [9]. There exists, however, an important need for robust policy iteration algorithms, which would allow for the efficient optimization of finite-state controllers of better quality and smaller size. To that effect, this paper contributes a new policy iteration algorithm that outperforms existing controller techniques in terms of robustness and scalability while being competitive with a state-of-the-art point-based technique. This is achieved by a novel composition of efficient heuristics to improve existing nodes and create new nodes in conjunction with a new technique that provably escapes local optima. Furthermore, improvements in finite-state controller search can be potentially useful for point-based algorithms (e.g., [25, 14, 19]) since the lower bound computation in those algorithms could be replaced by a finite-state controller which would avoid many challenges associated with maintaining large sets of belief points.

2. BACKGROUND

Partially observable Markov decision processes (POMDPs) are formally defined by a tuple $\langle S, A, O, T, Z, R, \gamma \rangle$ where S is the set of states s, A is the set of actions a, O is the set of observations o, $T(s, a, s') = \Pr(s'|s, a)$ defines the transition probabilities, $Z(o, a, s') = \Pr(o|a, s')$ defines the observation probabilities, $R(s, a)$ defines the reward function, and $0 < \gamma < 1$ is the

discount factor. The goal is to find a policy $\pi : H \rightarrow A$ that maps histories $h \in H$ of previous actions and observations to the next action. Since the length of histories grows with time and the number of histories grows exponentially with time, there is a need for a bounded representation. One option is to use belief states (i.e., distributions over states) which can be thought as a sufficient statistic that encodes the same information as histories [13]. Alternatively, we can restrict policies to finite state controllers, which define a compact mapping from histories to actions [11].

A finite state controller (FSC) consists of a set of nodes labeled with actions and edges labeled with observations. An FSC is parametrized by an action mapping ϕ, which assigns an action to each node (i.e., $\phi : N \rightarrow A$) and a node mapping ψ, which indicates which node each edge maps to (i.e., $\psi : N \times O \rightarrow N$). The policy encoded by a controller is executed by performing the action associated with each node traversed and by following the edge associated with each observation received. This execution requires a negligible amount of computation at each step to update the current node and lookup the corresponding action. Hence, it is ideal for applications with severe computation or energy constraints. The value $V_n^\pi(s)$ of starting a controller $\pi = \langle \phi, \psi \rangle$ in node n at state s is an $|S|$–dimensional vector computed as follows:

$$V_n^\pi(s) = R(s, \phi(n)) + \gamma \sum_{s',o} \Pr(s', o|s, \phi(n)) V^\pi_{\psi(n,o)}(s') \; \forall s, n \quad (1)$$

These $|S|$–dimensional vectors are often called α-*vectors*. Without loss of generality, we assume that the policy of a controller always starts in the first node n_1. Hence the value of a controller at initial belief b_0 is $V^\pi(b_0) = \sum_s b_0(s) V^\pi_{n_1}(s)$. Solving a POMDP using a controller search method consists of finding $\pi = \langle \phi, \psi \rangle$ which maximizes $V^\pi(b_0)$.

While finite-horizon POMDPs (i.e. when the policy is planned for a finite number of time steps) enjoy piecewise-linear value functions in the form of a finite set of α-vectors, it is not always the case in infinite horizon POMDPs; however, the existing literature has shown that policies with limited memory can still be near-optimal [22]. In general, the optimal value function V^* satisfies Bellman's equation:

$$V^*(b) = \max_a R(b, a) + \gamma \sum_o P(o|b, a) \; V^*(b^a_o) \quad (2)$$

where $b^a_o(s') \propto \sum_s b(s) \Pr(s'|s, a) \Pr(o|a, s')$ is computed using Bayes' rule and represents the belief reached after action a has been executed in belief b and observation o was observed. When applied to compute successive approximations, it is known as a dynamic programming update rule. Since b is continuous, exact solutions are not possible in general, and approximations, e.g., finite-state controllers, are required.

It was shown in [22] that stochastic finite-state controllers (FSCs) that allow stochastic edges and stochastic actions can yield higher value than deterministic FSCs introduced above subject to the same number of nodes. In stochastic FSCs, the notation for the observation strategy becomes $\psi(n, o, n') = P(n'|n, o)$, defining a distribution over successor nodes n' given an $\langle n, o \rangle$ pair. Similarly, $\phi(n, a) = P(a|n)$ is the probability of an action a given a node n. During controller execution, actions and next nodes are sampled from their distributions associated with the current node.

3. RELATED ALGORITHMS

We first review policy iteration for fully observable Markov decision processes (MDPs). After initializing the policy, π, arbitrarily, the algorithm iterates between (1) *policy evaluation* and (2) *policy improvement*. The first step uses a simplified version of Eq. 1, whereas the second step tries to improve the policy at every state, s, by computing a Bellman update for state s that uses the current values of all next states s':

$$\pi(s) = \arg\max_a \big(R(s, a) + \gamma \sum_{s'} T(s, a, s') V^\pi(s') \big). \quad (3)$$

The process stops when π cannot be improved any more; at this point, π is optimal. Policy iteration is, however, not as straightforward in the case of POMDPs because the policy improvement step is more complicated due to partial observability. In MDPs each state corresponds to a node, but in POMDPs the states are unobserved and therefore a vector of values for all states is associated with each node.

Initially, policy improvement in POMDPs was implemented by adding new nodes to the controller using dynamic programming (DP) and removing dominated nodes [12]. Having a solution to Eq. 1, which is the result of policy evaluation (in the form of N α-vectors), the dynamic programming update produces new α-vectors and their nodes. Dynamic programming for POMDPs has been the subject of extensive research; the goal was to compute new, useful α-vectors efficiently using Eq. 2 and exploiting the fact that the value function is piecewise-linear and convex [13]. Various methods were introduced for this purpose with incremental pruning being the fastest known method for exact (DP) updates [5]. However, up to $|A||N|^{|O|}$ nodes may be produced by each DP backup, so the method quickly becomes intractable. In policy iteration, after new nodes have been added to the controller, pointwise dominated nodes can be removed. A node is pointwise dominated when its value is less than the value of some other node at all belief states; conversely, a node is not dominated when there exists a belief—called a *witness belief*—at which the value of the node is higher than the value of all other nodes. Inward edges leading to a removed node are redirected to the dominating node.

Instead of adding a potentially intractable number of nodes and then removing pointwise dominated nodes, bounded policy iteration (BPI) [22] directly improves existing nodes without adding new nodes unless a local optimum is reached. First, BPI considers stochastic nodes, which often allow controllers of equal or higher value to be represented by fewer nodes. In particular, nodes jointly dominated by several nodes can be removed by using stochastic edges. Second, the authors of BPI observed that the quality of a FSC can often be improved by 'rewiring' edges of existing nodes, which keeps the size of the controller constant. This approach improves existing nodes individually and can be implemented with the algorithm in Fig. 1. This optimization procedure guarantees that the new node is better at all beliefs than the node n whenever $\delta > 0$. Interestingly, this approach has a very close relationship to policy improvement in MDPs. The procedure in Fig. 1 tries to improve the current node n by doing DP backup(s) from $V^\pi_{n'}$ (see similarity to Eq. 3 where s are nodes). It is one DP backup when the new node is deterministic, and a convex combination of DP backups when the new node is stochastic; we will refer to this first step of policy iteration in POMDPs as *node improvement*. This ob-

$$\begin{array}{ll} \text{max:} & \delta \\ \text{s.t.} & \forall_s, V_n(s) + \delta \leq \sum_a [c_a R(s, a) + \\ & \qquad \gamma \sum_{s',o} P(s'|s, a) P(o|s', a) \sum_{n_o} c_{a,n_o} V_{n_o}(s')] \\ & \sum_a c_a = 1; \sum_{n_o} c_{a,n_o} = c_a; \forall_a, c_a \geq 0; \forall_{a,o}, c_{a,n_o} \geq 0 \end{array}$$

Figure 1: The BPI node improvement linear program for node n where $c_a = P(a|n)$ and $c_{a,n_o} = P(n_o|a, n)P(a|n)$. Decision variables are δ, c_a, and c_{a,n_o}.

$$
\begin{array}{ll}
\text{max:} & \sum_{n,s} o(n,s)\delta_{n,s} \\
\text{s.t.} & \forall_s, V_n(s) + \delta_{n,s} \leq \epsilon + \sum_a [c_a R(s,a) + \\
& \qquad \gamma \sum_{s',o} P(s'|s,a) P(o|s',a) \sum_{n_o} c_{a,n_o} V_{n_o}(s')] \\
& \sum_a c_a = 1; \sum_{n_o} c_{a,n_o} = c_a; \forall_a, c_a \geq 0; \forall_{a,o}, c_{a,n_o} \geq 0 \\
& \delta_{n,s} \geq 0;
\end{array}
$$

Figure 2: The biased BPI node improvement linear program for node n where $c_a = P(a|n)$ and $c_{a,n_o} = P(n_o|a,n)P(a|n)$. Occupancy frequency $o(n,s)$ can be thought as a scaled belief where improvement is maximized. Decision variables are $\delta_{n,s}$, c_a, and c_{a,n_o}; ϵ is a parameter explained in text.

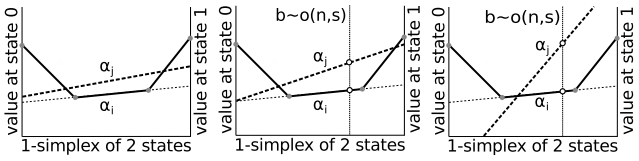

Figure 3: Alpha vectors α_j for improved nodes.

servation is useful for understanding Sec. 4 and the challenges of policy iteration in POMDPs.

When no node can be improved, BPI will try to escape local optima using a special procedure that replaces the dynamic programming update used in earlier algorithms. NB: the escape step is not required in MDPs since policy improvement converges to a global optimum. In BPI, the escape procedure exploits the fact that when the algorithm in Fig. 1 cannot improve node n, the dual solution corresponds to a belief whose value is tangent to the current value function. In order to escape the local optimum, BPI performs a one-step lookahead search (DP backup) from the tangent belief in order to find a new node which is then added to the controller. An improved version of the optimization program in Fig. 1 was introduced in [21] and is shown in Fig. 2. The objective of the optimization program is biased by the occupancy frequency of the current controller. Occupancy frequency, $o(s,n) \in \mathbb{R}$, indicates the expected number of times (discounted by γ^t for t time steps) that state s is reached in node n when executing a particular policy or a finite-state controller [21]. Intuitively, it is useful to focus the node improvement procedure on those states that are more likely to reach the node n. Relevant relationships between occupancy frequency and optimality of the controller are discussed in [23] where forward search from the beliefs proportional to the occupancy frequency of each node was used to escape local optima too.

Overall, policy iteration in POMDPs requires two steps (1) policy evaluation and (2) policy improvement which attempts (2a) node improvement, does (2b) escape when (2a) does not improve any nodes, and (2c) performs pruning to compress the controller and potentially improve its quality.

4. NODE IMPROVEMENT

We start with an example in Fig. 3a to show what happens when the optimization procedure in Fig. 1 finds a better node. The example is for a two-state POMDP, and the node that corresponds to the vector α_i is being examined for improvement. If the improvement is possible, the α-vector of an improved node could look like α_j. The improved vector α_j has to dominate the old vector α_i at the entire belief simplex because the optimization procedure in Fig. 1 tries to maximize the smallest improvement over all corners of the belief simplex. The biased version of this procedure, shown in Fig. 2, tries to maximize the improvement at a belief point proportional to the occupancy frequency $o(n,s)$ of the current node. By default, the procedure from Fig. 2 when used with $\epsilon = 0$ tries to maximize the improvement at $o(n,s)$ subject to a constraint that there is no belief in the entire belief simplex where the value would become lower (see Fig. 3b). This restriction often prohibits the algorithm from finding a new node. To circumvent this limitation, the parameter ϵ can be set to a value higher than 0 in Fig.2 ($\epsilon = (\max_{s,a} R(s,a) - \min_{s,a} R(s,a))/400(1-\gamma)$ was recommended in [21]). This way, a new node with some improvement at $o(n,s)$ can be found at the cost of a reduction in value at some other beliefs. This approach does not guarantee improvement of the quality of the controller any more, but it can find useful nodes in many cases. Low values for ϵ ensure that the potential loss in the quality of the controller is low.

The above indicates that the existing literature did not solve the problem of node improvement whereas a robust and valid node improvement procedure is crucial for preventing the growth of the controller and for effective policy iteration. We observe that constraining the loss at some beliefs (using $\epsilon = 0$ in Fig. 2) is the main reason why many good nodes cannot be found by the BPI optimization procedures. Therefore, our approach examines an extreme case when we allow the value at some of the belief points to have arbitrarily large loss. With that, we can gain a much higher improvement at beliefs we are interested in, e.g., at $o(n,s)$. This is illustrated in Fig. 3c which demonstrates a new vector α_j which has to go down very far on the left hand-side of the figure in order to have high improvement at a belief proportional to $o(n,s)$. In order to facilitate such flexibility, it is sufficient to remove the parameter ϵ from the linear program in Fig. 2 and to allow $\delta_{n,s}$ to take any values in $\mathbb{R}$. We will call this optimization model, derived from Fig. 2, an LP-based *unconstrained node improvement* (LP stands for linear programming) in the remainder of the paper. Furthermore, this naturally leads to a conjecture that the existence of the parameter ϵ and the non-negativity constraints on variables $\delta_{n,s}$ are the primary reasons why the original formulations in Fig. 2 (as well as Fig. 1) arrive at stochastic nodes. Those extra constraints prohibit deterministic nodes because deterministic nodes would often require substantially reduced values at some beliefs. With that observation, we can remove the linear program, and an improved deterministic node can be sought directly; for that, it is sufficient to consider the $o(n,s)$ vector of the node as a belief state for which a new node can be constructed by a one-step lookahead search (i.e. one Bellman backup) from $o(n,s)$ (which corresponds to policy improvement in MDPs using Eq. 3). We name this procedure DP-based *unconstrained node improvement* (DP stands for dynamic programming); its effectiveness is demonstrated in Sec. 7.

So far, we have shown how and why better nodes can be found using unconstrained node improvement, and we conjecture that deterministic nodes are sufficient. However, it is well known that if one uses $o(n,s)$ as a belief state for constructing improved nodes, the new node may reduce the quality of the entire controller even if an individual node could be improved at $o(n,s)$. This property is formally discussed in Thm. 2 in [23], and it explains why only a small reduction (small ϵ) or no reduction ($\epsilon = 0$) at some beliefs were allowed in BPI [22]. We propose a simple solution to this problem.

Our approach is to apply the unconstrained node improvement in order to compute the node with the best possible improvement at $o(n,s)$, and also to verify whether the overall quality of the controller is improved. We temporarily replace the old node by the new node and compute $V_{n_1}(b_0)$. If an improvement is found, the new node is kept; if not, we put the old node back into the controller. Exploiting the monotonicity of Bellman backups [16], in some situations, it would be sufficient to perform a single backup instead

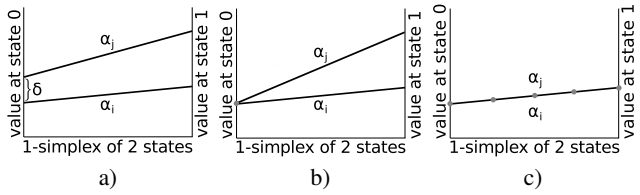

Figure 4: Possible node improvements computed using Fig. 1; α_i corresponds to an old node and α_j to a new node.

of a full evaluation to determine if the new node improves the controller. Additionally, even if the new node is not useful at this stage, it may be useful at the escape stage, which is discussed in detail in Sec. 5; hence, the time to compute the new node is not wasted. Our procedure for node improvement is summarized in Alg. 1.

Algorithm 1: IMPROVENODE: A new method for improving individual nodes of a finite-state controller.

Data: FSC - finite-state controller, n_{old} - node to improve
Result: $true$ when FSC was improved
compute n_{new} by LP or DP unconstrained node improvement
swap(FSC, n_{old}, n_{new})
evaluate(FSC) ; /* solve Eq. 1 */
if *FSC improved at b_0 in node n_1* **then**
  return $true$
else
  swap(FSC, n_{new}, n_{old})
  add(node-set, n_{new}) ; /* store n_{new} for escape */
  return $false$

5. PROVABLE ESCAPE METHODS

Algorithm 1 is expected to find better nodes than the original BPI procedures and in some situations it can find improved nodes even when BPI is not able to find any. As a result, doing bounded policy iteration with our node improvement method will lead to better controllers of a given size. However, there is still a need to escape local optima (i.e., when a controller is subotimal and the node improvement techique cannot improve any of the nodes). An escape technique was proposed in BPI [22] that adds nodes based on a one step lookahead from the tangent belief obtained from the dual solution to the linear program in Fig. 1. Let us investigate this method in detail. Three types of solutions (Fig. 4) can be computed using Fig. 1.

1. In Fig. 4a, the new node is better than the old node at the entire belief simplex. The new node can be used, and there is no local optimum.

2. Fig. 4b shows the situation where the LP objective δ is zero (the smallest improvement is zero). The new node is still useful if there exists a corner in the belief simplex where the improvement is higher than zero (for example, state 1 in this case). The dual solution returns a corner (corner 0 in this example) as a tangent belief for escape. Either the new node or the tangent belief could be used to improve the controller.

3. The objective δ is 0, indicating that α_j is identical to α_i since no improved node was found (Fig. 4c). At this point, BPI would use the dual solution as a belief point from which the escape can be performed by a one step look ahead. An inspection of Fig. 4c shows, however, that there is an infinite number of dual solutions; any point in the belief simplex can be the dual solution. Therefore, the dual solution depends on the solver, and it does not have to be a belief that is useful for escaping the local optimum. Our experiments with the CPLEX solver showed that the same dual solution is always returned; however, the solution may not be useful for escape. In many cases, a corner of the belief simplex is returned by CPLEX, which is usually a useful belief for escaping local optima at the beginning of policy iteration, but we know from Cheng's algorithm that corners of the belief simplex are not sufficient [6].

The analysis of Fig. 4 shows that a one step lookahead from the tangent beliefs as done by BPI is not sufficient to guarantee the escape of a local optimum. With this observation, we are back to methods that existed before BPI was proposed if we want to have a method that is guaranteed to escape local optima.

One option is to use methods developed for planning in finite horizon POMDPs where incremental pruning was shown to be the most efficient method [5]. However, those methods do an exact DP update, produce many new nodes, and do not scale to large problems. Another option is based on heuristic search; the forward search can be performed using upper bounds for exploration—an approach discussed by Hansen [12] and utilized in point-based planners with heuristic search [25, 14, 19]. Relying on a duality gap is a standard technique in artificial intelligence and optimization, and assuming that the heuristic search is implemented efficiently, the method is expected to be robust. In our paper, however, we attempt to push the understanding of escape methods in POMDP planning further, so that more can be achieved before one has to apply heuristic search for many steps because many nodes may need to be added in one escape step. In what follows (including all our empirical results), we investigate principled methods to escape local optima in policy iteration where we do not allow more than one step of lookahead search (a challenge that was attempted in BPI as well).

Since BPI is not guaranteed to escape local optima, and exact DP is intractable, a new efficient, but provable, method is missing. Before we introduce our approach, we review some useful concepts. The naïve way to compute the exact DP update for POMDPs is to enumerate all possible alpha vectors. We can compute the set $\Gamma^{a,o}$ of vectors $V_n^{a,o}(s)$ for each $\langle a, o\rangle$ pair by applying a DP update:

$$\Gamma^{a,o} \leftarrow V_n^{a,o}(s) = \frac{R^a(s)}{|O|} + \gamma \sum_{s' \in S} P(o|s', a) P(s'|a, s) V_n^{\pi}(s'), \forall_n \quad (4)$$

Having $\Gamma^{a,o}$, the cross-sum can be computed for every action:

$$\Gamma^a = \Gamma^{a,o_1} \oplus \Gamma^{a,o_2} \oplus ... \oplus \Gamma^{a,o_{|O|}} \quad (5)$$

The last step takes the union of Γ^a and sets:

$$V = \cup_{a \in A} \Gamma^a \quad (6)$$

which computes the set of new α-vectors V. The set V is guaranteed to provide the best possible improvement over the entire belief simplex. In bounded policy iteration, it is sufficient to add one node (in other words, one new α-vector) which may escape a local optimum. We propose a quadratic optimization program with linear constraints that is guaranteed to find a node with the best improvement possible over the entire belief simplex (see Fig. 5). We can also show that the search can be restricted to deterministic nodes.

THEOREM 1. *There always exists an optimal solution to the quadratic problem shown in Fig. 5 that is integral $\forall_{n',a,o} P(n', a|o)$, i.e., there exists an optimal solution that corresponds to a deterministic node.*

$$
\begin{array}{ll}
\text{max:} & \sum_{a,n',o,s} w(s)P(n',a|o)V_{n'}^{a,o}(s) - \beta \\
\text{s.t.} & \sum_s w(s) = 1; \sum_{n',a} P(n',a|o) = 1; \\
& \forall_{a,o_1,o_2} \sum_{n_1} P(n_1,a|o_1) = \sum_{n_2} P(n_2,a|o_2) \\
& \forall_n \beta \geq \sum_s w(s) V_n^\pi(s); \\
& \forall_s w(s) \in R; \forall_{n',a,o} P(n',a|o) \in [0,1]
\end{array}
$$

Figure 5: A quadratic optimization program to search for a new node that provides maximal improvement at the entire belief simplex; the belief w (witness belief) is the belief at which the improvement happens. Decision variables are the witness belief w, the current value β at belief w, and node parameters $P(n',a|o)$ which, when interpreted as probabilities, correspond to $P(n'|o)P(a)$.

PROOF. First, we will show by contradiction that if we pick a particular w, then no stochastic node can be better than the best deterministic node computed for w. When w is fixed, β is also fixed (i.e., constant) and therefore the objective can be simplified to $\sum_{a,n',o} P(n',a|o) \sum_s w(s) V_{n'}^{a,o}(s)$. Suppose that the optimal solution $P^*(n',a|o)$ is stochastic and achieves strictly higher value than any deterministic node $n = \langle \phi, \psi \rangle$ at w:

$$
\sum_{a,n',o} P^*(n',a|o) \sum_s w(s) V_{n'}^{a,o}(s) \qquad (7)
$$
$$
> \sum_{s,o} w(s) V_{\psi(o,n)}^{\phi(n),o}(s) \ \forall n
$$

Since any stochastic solution can be rewritten as a convex combination of deterministic nodes, let p_n^* be the probability of each deterministic node n corresponding to $P^*(n',a|o)$. We can then rewrite the objective as follows:

$$
\sum_{a,n',o} P^*(n',a|o) \sum_s w(s) V_{n'}^{a,o}(s) \qquad (8)
$$
$$
= \sum_n p_n^* \sum_{s,o} w(s) V_{\psi(o,n)}^{\phi(n),o}(s)
$$

Eq. 8 contradicts the assumption in Eq. 7 since a convex combination of terms cannot be strictly greater than each individual term. Since this argument applies to all w's, this shows that there must exist an optimal deterministic solution that corresponds to a deterministic node. □

The theorem shows that stochastic nodes do not have any advantage over deterministic nodes in escaping a local optimum in policy iteration and is consistent with a known fact that deterministic policies are sufficient for solving POMDPs assuming that the size of the controller is not bounded.

We seek exact methods for solving Fig. 1. Fortunately, the quadratic term $w(s)P(n',a|o)$ is formed by a multiplication where at least one of the variables (e.g., $P(n',a|o)$) is in the range of $[0,1]$. Therefore, a special case of the McCormick relaxation [15] (known as disjunctive programming) can be used to linearize quadratic terms. Note that disjunctive programming is exact (not a relaxation) when one of the factors is restricted to $\{0,1\}$ (instead of $[0,1]$). Hence, we replace $w(s)P(n',a|o)$ by a new variable $y(s,a,o,n')$, restrict $P(n',a|o)$ to $\{0,1\}$ and add McCormick constraints to obtain the mixed-integer linear program in Fig. 6. Since restricting $P(n',a|o)$ to integral values does not change the best improvement that can be achieved according to Thm. 1, the folowing corollary follows.

COROLLARY 1. *Optimal solutions to the mixed-integer linear problem in Fig. 6 are also optimal for the corresponding quadratic program in Fig. 5.*

$$
\begin{array}{ll}
\text{max:} & \sum_{a,n',o,s} y(s,n',a,o)V_{n'}^{a,o}(s) - \beta \\
\text{s.t.} & \sum_s w(s) = 1; \sum_{n',a} P(n',a|o) = 1; \\
& \forall_{a,o_1,o_2} \sum_{n_1} P(n_1,a|o_1) = \sum_{n_2} P(n_2,a|o_2) \\
& \forall_n \beta \geq \sum_s w(s) V_n^\pi(s); \\
& \forall_s w(s) \in R; \forall_{n',a,o} P(n',a|o) \in \{0,1\}; \\
& \forall_{s,a,o,n'} 0 \leq y(s,a,o,n') \leq P(n',a|o); \\
& \forall_{s,a,o,n'} w(s) + P(n',a|o) - 1 \leq y(s,a,o,n') \leq w(s);
\end{array}
$$

Figure 6: The McCormick transformation of the quadratic program in Fig. 5. $y(s,n',a,o)$ is a new decision variable. Note that $P(n',a|o)$ is discrete, which changes the model to a mixed-integer linear program.

As a result, the mixed-integer linear program (MILP) in Fig. 6 finds a deterministic node and its witness belief, w, such that the improvement at w is the highest possible—the property that would be crucial for planning over the entire belief simplex too. In our case, we are not necessarily interested in *the best* node, because we are optimizing the controller at b_0. Therefore, we can stop the MILP solver as soon as any node that escapes a local optimum is obtained. Such nodes can also be computed using linear programming (LP) relaxation of Fig. 6 by making $P(n',a|o)$ continuous. Next, we introduce a few interesting properties about such a relaxation.

THEOREM 2. *If the belief vector, w, in the solution to an LP-relaxation of the MILP in Fig. 6 is integral, then the solution is optimal for the unrelaxed MILP.*

PROOF. It is sufficient to show that if w is a vector of integers in the solution to the LP-relaxation, then $\forall_{s,n',a,o} y(s,n',a,o) = w(s)P(n',a|o)$ (this is not the case in general, and this is the reason why an LP-relaxation is usually not tight). To this end, we focus on the last two constraints (McCormick constraints) in Fig. 6. For brevity, we do not write all variables. If $w = 0$, then $y = 0$ because $y \leq w$ in the first McCormick constraint; hence, $y = wp$ is satisfied. If $w = 1$, then $y = P$ because $w + P - 1 \leq y \leq P$ gives $P \leq y \leq P$; $y = wp$ is satisfied as well. □

NB: the proof does not rely on the optimal node being deterministic, but Thm. 1 shows that there exists a deterministic node that is optimal indeed. Using the same argument, a corresponding theorem can be proven.

THEOREM 3. *If all $P(n',a|o)$ in the solution to an LP-relaxation of the MILP in Fig. 6 are integral (i.e. the node is deterministic), then the solution is optimal for the unrelaxed MILP.*

Note that this time, w may be fractional.

Theorem 2 has a convenient implication that whenever w is a corner belief of the belief simplex, the solution and the node are optimal. When w is not integer, the objective value is an upper bound, and variables $P(n',a|o)$ may represent a useless node, which is worse than any of the existing nodes (unless all $P(n',a|o)$ are integral as shown in Thm. 3). However, w may still be a useful belief to construct a useful node. Whenever the node computed by the LP-relaxation is not useful, one can create a new deterministic node for w directly. Only if this node does not show a sufficient improvement, the MILP can be attempted. Even though MILP is NP-hard, it can be stopped as soon as a feasible solution is found since the optimal solution is not needed. These steps are summarized in Alg. 2.

The optimization framework described above provides a guaranteed way of escaping local optima, i.e., it will always find a new, useful node whenever such a node exists. Next, we will provide a few heuristic approaches for escape which are cheap to compute

Algorithm 2: MILPESCAPE: Provable escape method using mathematical optimization.

Data: V^{π} - alpha-vectors of the current controller
compute $\Gamma^{a,o}$ using Eq.4
$\langle node_{LP}, w\rangle \leftarrow$ solve an LP-relaxation of Fig. 6
if w *is integral* ***or*** $node_{LP}$ *is deterministic* **then**
  return $node_{LP}$
else
  $\langle node_w, improvement\rangle \leftarrow$ construct a new node for witness w
  if $improvement > threshold$ **then**
    return $node_w$
  else
    return solve MILP in Fig. 6 in an anytime fashion

yet are quite robust. Therefore, it is possible to reduce the number of times the optimization framework needs to be used.

The node improvement algorithm (LP- or DP-based) examined in Sec. 4 computes a DP backup for a given node, n, at its current belief, b_n, obtained from the occupancy frequency, $o(n, s)$. Values of beliefs b' reachable from b_n are estimated using the current alpha vectors, V^{π}. The first, heuristic escape method that we consider constructs new nodes by computing DP backups for beliefs $b' = T(b_n, a, o)$. For every node, n, the algorithm computes b' for all (a, o) and then computes their DP backups. If the new value at b' is better than the value obtained from V^{π}, the new node can improve the quality of the controller. All new nodes are ranked by improvement at their b', and the node that has the highest improvement is selected for addition to the controller. Two versions of the procedure are considered. In the first one, named *on-policy lookahead*, only the actions that have non-zero probability of being used in their node n are used to compute b'; we assume that it is more advantageous to add nodes for beliefs, b', that are reachable by the current controller. The second version, named *off-policy lookahead*, uses the remaining actions to compute b'. The use of occupancy frequency and forward search for escape was considered in [23] where the algorithm constructs a search tree up to an arbitrary depth and expands all actions at every step.

Nodes computed in Sec. 4 are rejected when they do not improve the quality of the controller. However, they can still be useful for escape because they improve the value at their beliefs, b_n. When such a node is added to a controller, one can view it as a process of splitting the node n because the old node, n, is left in the controller whereas the new node that improves the value at b_n is added as well; this leads to two different nodes that have the same b_n (in other words, the original node had two reachable beliefs which required different edges) which justifies why the process can be named node splitting. To this end, our method introduced in Sec. 4 stores all rejected nodes of the current iteration, and when necessary, it uses the node with the highest improvement at its b_n for escape. The concept of node splitting was investigated in [23]; however, the connection between the original node and the new node was unclear.

Even though, the corners of the belief simplex do not guarantee escape [6], our third heuristic approach computes new nodes for corners of the belief simplex, and the node that has the highest improvement at its corner is selected for addition.

6. POLICY ITERATION

The ideas introduced in this paper are combined into a policy iteration algorithm (see Alg. 3). Traditionally, controller optimization algorithms are initialized with a random controller, which helps escaping poor solutions. Here, we take a more challenging

Algorithm 3: IPI(-LP): Incremental Policy Iteration for POMDPs.

Data: $POMDP$
Result: FSC for $POMDP$
1 $FSC.N \leftarrow \{n_1\}$; /* the first node */
2 $FSC.N \leftarrow FSC.N \cup \{n_2\}$; /* the second node */
3 **while** $impr = true$ **do**
4 Policy evaluation using Eq. 1
5 **for** $n \in FSC.N$ **do**
6 $impr \leftarrow$IMPROVENODE(FSC, n); /* DP or LP */
7 **if** $\neg impr$ **then** /* escape is required */
8 $impr \leftarrow$ONPOLICYLH(FSC)
9 **if** $\neg impr$ **then**
10 $impr \leftarrow$BESTOF(OFFPOLICYLH, SPLIT, CORNER)
11 **if** $\neg impr$ **then**
12 $impr \leftarrow$MILPESCAPE(FSC)
13 Prune dominated nodes

approach and our algorithm always starts from an empty controller; this way our algorithm does not introduce any randomization. The first node (Ln. 1) is the one that maximizes $V_{n_1}(b_0)$ whereas the second node (Ln. 2) can also be easily constructed because all its edges go to n_1—such a node would be found using our MILP, but we can construct it directly. After computing the first two nodes, the algorithm starts its main 'while' loop where the first step is policy evaluation (Ln. 4). Afterwards, all nodes are examined (Ln. 6), and the escape block (Ln. 7–12) is entered only when none of the nodes can be improved. *The order of the escape methods is crucial.* Intuitively, we first want to add new nodes that improve the nodes of the current policy and consequently improve $V(b_0)$. Therefore, the on-policy lookahead method is justified. The second group of methods is executed only when no new node can be constructed using on-policy lookahead. They are put together in one group because none of them guarantee that a node will be found. If there is no improvement, the method of a last resort is the MILP optimization procedure which is guaranteed to find an improvement (as long as one exists), yet it facilitates anytime execution. The methods executed in Ln. 10–12 seek improvement anywhere in the belief simplex, but they still guarantee escape because in subsequent iterations their nodes will become useful for $V(b_0)$ after incoming edges that introduce reachability from b_0 have been added. The escape step adds at most one node to keep the size of the controller small; therefore, pruning of dominated nodes can be left for the last step (Ln. 13). We prune nodes that are unreachable from n_1. The algorithm is named IPI when used with DP-based node improvement in Ln. 6 and IPI-LP when used with LP-based node improvement.

In our algorithm, one extension was introduced into the way the new nodes are added to the controller during escape. We observe that under specific conditions the new node can be merged into an existing node, and the size of the controller does not have to grow. This can happen when the two nodes differ only by those edges that have zero probability (with respect to a node's witness beliefs) in at least one of the nodes. In our implementation, such nodes are also merged during controller pruning in Ln. 13 in Alg. 3.

7. EXPERIMENTS

Algorithm 3 is compared against the state-of-the-art for controller optimization: bounded policy iteration (BPI) with escape [22], quadratically constrained linear programming for Moore (QCLP) [1] and Mealy automata (QCLP-Mealy) [3], expectation maximiza-

tion (EM) [23], and branch-and-bound (B&B) with isomorph pruning [10]. For reference, the results with the point-based solver SARSOP [14] are also provided (NB: SARSOP is not a policy iteration algorithm). Since IPI was outperforming all the other FSC search algorithms, and it is was the only FSC algorithm that provided consistent improvement, we stopped it whenever more nodes would not lead to a significant value increase. In this way, the controller size of IPI(-LP) was used as a reference point for other methods. For SARSOP, we report both the result that has the same or closest running time to IPI(-LP) and the result that has the same or closest number of nodes. All algorithms were evaluated on a suite of standard POMDP problems from Cassandra's repository[1] and a few problems from the recent literature [10].

Results in Tab. 1–2 are remarkable because, for the first time, policy iteration can solve a number of common benchmarks (chainOfChains3, cheese-taxi, hhepis6obs_woNoise, and tiger.95) which were notoriously challenging for local search methods. IPI could solve them by starting from an empty controller without relying on randomized initialization. While B&B is guaranteed to find the best controller of a given size since it does an exhaustive search, our method scales much better, especially for problems exceeding 10^4 states. IPI also computes controllers that are much smaller than those computed by SARSOP, e.g., on elevators_inst_pomdp_1 and baseball the number of nodes is reduced by 3 orders of magnitude. On several domains, controllers computed by IPI were not only the smallest, but also with the highest value even when compared to large policies computed by SARSOP on lacasa2a, lacasa4.batt, hallway2, 4x5x2.95, aloha.10, and underwaterNav. Overall, for the first time, an FSC technique compares well to SARSOP, which is one of the best point-based value iteration techniques. Additionally, our methods consistently outperform alternative controller optimization methods: they achieve higher value on 13/17 problems, the same value and controller size on 3/17 problems, and the same value with smaller controller size on 1/17 problems when compared to existing controller techniques.

Almost identical results for IPI and IPI-LP show that it does not matter if node improvement is implemented using a linear program or a DP backup in our *unconstrained node improvement*, and therefore deterministic nodes are sufficient. A detailed investigation has shown that even if the LP-based unconstrained node improvement is used, the nodes it finds are all deterministic in all results reported in Tab. 1 and 2; recall that in Sec. 4, we explained that constraints present in Fig. 1 frequently lead to stochastic nodes in BPI. An important consequence of that fact is not only the reduced computational complexity of the node improvement procedure (a one-step lookahead search instead of solving a linear program is sufficient), but the fact that good, limited memory controllers can be constructed from deterministic nodes only. The differences in the performance of IPI and IPI-LP can result from different ways both methods break ties or deal with zero-probability edges.

Table 3 shows which escape methods were used by the IPI-LP solver in our evaluation. The ratios (vertical bars) indicate that the node improvement step led to the most frequent improvements on almost all domains; the significance of our insights into the node improvement step of policy iteration for POMDPs becomes evident. Intuitively, after a new node has been added during escape, the controller can be improved several times using our simple, yet powerful node improvement procedure. On-policy lookeahead was the most frequent escape method. The other methods were less frequent; however, their importance is critical to escape local optima. On the current suite of domains, the MILP-based escape method

[1]http://www.cassandra.org/pomdp/examples/index.shtml

POMDP	solver	value $V(b_0)$	#nodes	time [s]
chainOfChains3	SARSOP	157	10	0.1
$\|S\|$=10, $\|A\|$=4	B&B	157	10	1.7
$\|O\|$=1, $\gamma = 0.95$	EM	0.17 $\pm$ 0.06	10	6.9
UB=157	QCLP	0 $\pm$ 0	10	0.2
	QCLP-Mealy	33.1 $\pm$ 2.3	10	0.1
	BPI	25.7 $\pm$ 0.77	10	4.3
	IPI	157	10	0.6
	IPI-LP	157	10	1.5
cheese-taxi	SARSOP	2.48	163	0.1
$\|S\|$=34, $\|A\|$=7	SARSOP	-6.38	40	0.0
$\|O\|$=10, $\gamma = 0.95$	B&B	-19.9	10	24h
UB=2.48	EM	-17.2 $\pm$ 2.05	12	64.5
	QCLP	-17.77 $\pm$ 2.22	12	124.6
	QCLP-Mealy	-12.5 $\pm$ 4.74	12	2779.4
	BPI	-10.46 $\pm$ 0.97	12	75.1
	IPI	2.48	11	3.6
	IPI-LP	2.48	12	8.1
lacasa2a	SARSOP	6715.67	53	135.5
$\|S\|$=320, $\|A\|$=4	SARSOP	6706.5	12	6.7
$\|O\|$=12, $\gamma = 0.95$	B&B	6710.0	3	493.8
UB=6717	EM	6707.9 $\pm$ 0.0	7	1708.8
	QCLP	6687.5 $\pm$ 12.85	7	5880.3
	QCLP-Mealy	6714.6 $\pm$ 0.0	2	5497.0
	BPI	6707.0 $\pm$ 0.21	7	450.4
	IPI	6715.70	7	61.6
	IPI-LP	6715.70	7	134.2
lacasa3.batt	SARSOP	293.5	103	345.2
$\|S\|$=1920, $\|A\|$=6	SARSOP	293.2	34	120.8
$\|O\|$=36, $\gamma = 0.95$	B&B	287.0	5	24h
UB=294.6	EM	293.1 $\pm$ 0.02	8	46873.6
	QCLP	n.a.	n.a.	n.a.
	QCLP-Mealy	n.a.	n.a.	n.a.
	BPI	293.3 $\pm$ 0.10	13	6168.3
	IPI	293.5	13	4758.7
	IPI-LP	293.5	12	6017.2
lacasa4.batt	SARSOP	291.46	12866	27979.1
$\|S\|$=2880, $\|A\|$=6	SARSOP	290.9	53	521.4
$\|O\|$=72, $\gamma = 0.95$	B&B	285.0	10	24h
UB= 292.6	EM	290.5 $\pm$ 0.01	5	35895.5
	QCLP	n.a.	n.a.	n.a.
	QCLP-Mealy	n.a.	n.a.	n.a.
	BPI	291.38 $\pm$ 0.08	14	19078.7
	IPI	291.48	9	9414.2
	IPI-LP	291.46	14	27370.1
hallway	SARSOP	1.00	630	1708.7
$\|S\|$=60, $\|A\|$=5	SARSOP	0.15	49	0.4
$\|O\|$=21, $\gamma = 0.95$	B&B	0.18	8	24h
UB=1.19	EM	0.95	40	n.a.
	QCLP	0.94 $\pm$ 0.00	40	5025.2
	QCLP-Mealy	0.90 $\pm$ 0.01	10	4425.4
	BPI	0.95 $\pm$ 0.03	40	88.8
	IPI	0.99	40	289.5
	IPI-LP	0.99	40	1711.0
hallway2	SARSOP	0.41	697	5046.6
$\|S\|$=60, $\|A\|$=5	SARSOP	0.11	50	0.4
$\|O\|$=21, $\gamma = 0.95$	B&B	0.11	8	24h
UB=0.88	EM	0.43	40	n.a.
	QCLP	0.42 $\pm$ 0.00	40	9133.2
	QCLP-Mealy	0.38 $\pm$ 0.00	10	7902.0
	BPI	0.41 $\pm$ 0.02	40	95.9
	IPI	0.42	40	899.8
	IPI-LP	0.43	40	5048.3
hhepis6obs_woNoise	SARSOP	8.64	1775	1.7
$\|S\|$=20, $\|A\|$=4	SARSOP	0.0	13	0.0
$\|O\|$=6, $\gamma = 0.99$	B&B	8.64	8	4.5
UB=8.64	EM	0.0 $\pm$ 0.0	7	9.7
	QCLP	0.82 $\pm$ 0.0	7	3.4
	QCLP-Mealy	0.81 $\pm$ 0.0	7	7.0
	BPI	0.0 $\pm$ 0.0	7	2.7
	IPI	8.64	7	8.5
	IPI-LP	8.64	7	9.6

Table 1: Results: comparison of the value $V(b_0)$, the number of nodes (#nodes), and the time to obtain a controller. Column POMDP contains details of all domains that are solved using solvers listed in column solver. 'n.a.' means that QCLP did not complete on `http://www.neos-server.org`, or that BPI and EM running on our machine did not produce results within 24 hours using 8GB of RAM.

POMDP	solver	value $V(b_0)$	#nodes	time [s]
4x5x2.95	SARSOP	2.02	509	6.7
$\|S\|$=39, $\|A\|$=4	SARSOP	1.14	30	0.1
$\|O\|$=4, $\gamma = 0.95$	B&B	2.02	5	639.9
UB=2.08	EM	1.46 $\pm$ 0.06	12	83.1
	QCLP	1.99 $\pm$ 0.05	12	13.4
	QCLP-Mealy	1.99 $\pm$ 0.04	12	19.0
	BPI	1.68 $\pm$ 0.19	12	26.7
	IPI	2.08	12	5.9
	IPI-LP	2.08	10	6.7
aloha.10	SARSOP	535.1	70	381.8
$\|S\|$=30, $\|A\|$=9	SARSOP	535.0	39	120.2
$\|O\|$=3, $\gamma = 0.999$	B&B	529.0	10	24h
UB= 544.2	EM	533.2 $\pm$ 0.42	38	1257.3
	QCLP	533.9 $\pm$ 0.57	38	541.6
	QCLP-Mealy	532.3 $\pm$ 2.05	38	1999.6
	BPI	524.6 $\pm$ 1.20	38	3129.9
	IPI	537.7	38	230.3
	IPI-LP	537.7	38	381.8
baseball	SARSOP	0.6412	123	1.35
$\|S\|$=7681, $\|A\|$=6	SARSOP	0.636	6	0.3
$\|O\|$=9, $\gamma = 0.999$	B&B	0.636	5	24h
UB=0.641	EM	0.636 $\pm$ 0.0	2	48656.0
	QCLP	n.a.	n.a.	n.a.
	QCLP-Mealy	n.a.	n.a.	n.a.
	BPI	0.636 $\pm$ 0.0	6	187.1
	IPI	0.6412	6	12.8
	IPI-LP	0.6412	6	60.7
elevators_inst_pomdp_1	SARSOP	-44.32	26701	4520.9
$\|S\|$=8192, $\|A\|$=5	SARSOP	-149.2	52	108.9
$\|O\|$=32, $\gamma = 0.99$	B&B	-149.0	10	24h
UB=-44.3	EM	n.a.	n.a.	n.a.
	QCLP	n.a.	n.a.	n.a.
	QCLP-Mealy	n.a.	n.a.	n.a.
	BPI	n.a.	n.a.	n.a.
	IPI	-45.41	10	3654.9
	IPI-LP	-44.32	11	13328.5
machine	SARSOP	63.17	78	96.5
$\|S\|$=256, $\|A\|$=4	SARSOP	35.73	42	1.0
$\|O\|$=16, $\gamma = 0.99$	B&B	62.6	6	52100.0
UB=66.4	EM	62.45 $\pm$ 0.09	9	780.7
	QCLP	62.78 $\pm$ 0.04	9	2859.2
	QCLP-Mealy	53.69 $\pm$ 9.21	9	17011.2
	BPI	53.27 $\pm$ 1.04	9	115.3
	IPI	63.04	9	48.9
	IPI-LP	63.04	9	96.5
tagAvoid	SARSOP	-6.11	9113	6614.4
$\|S\|$=870, $\|A\|$=5	SARSOP	-13.7	49	0.6
$\|O\|$=30, $\gamma = 0.95$	B&B	-19.9	9	24h
UB=-3.46	EM	-6.81 $\pm$ 0.12	9	19295.0
	QCLP	-19.99 $\pm$ 0.0	9	5506.9
	QCLP-Mealy	-9.46 $\pm$ 0.81	2	15798.0
	BPI	-8.46 $\pm$ 0.45	9	788.4
	IPI	-6.22	9	5888.0
	IPI-LP	-6.29	9	6613.9
tiger.95	SARSOP	19.3	5	0.1
$\|S\|$=2, $\|A\|$=3	B&B	19.3	5	1.4
$\|O\|$=2, $\gamma = 0.95$	EM	6.91 $\pm$ 2.48	5	0.2
UB=19.3	QCLP	-6.3 $\pm$ 3.79	5	0.7
	QCLP-Mealy	9.81 $\pm$ 5.23	5	0.03
	BPI	-20.2 $\pm$ 0.12	5	0.1
	IPI	19.3	5	0.7
	IPI-LP	19.3	5	0.9
underwaterNav	SARSOP	746.9	38860	34495.5
$\|S\|$=2653, $\|A\|$=6	SARSOP	681.2	54	0.5
$\|O\|$=103, $\gamma = 0.95$	B&B	747.0	10	24h
UB=754.0	EM	749.9 $\pm$ 0.00	6	34933.9
	QCLP	n.a.	n.a.	n.a.
	QCLP-Mealy	n.a.	n.a.	n.a.
	BPI	749.6 $\pm$ 0.08	53	30992.2
	IPI	749.5	3	22.6
	IPI-LP	750.0	53	33840.0
rockSample-7_8	SARSOP	21.66	32727	87671.5
$\|S\|$=12545, $\|A\|$=13	SARSOP	15.34	70	0.9
$\|O\|$=2, $\gamma = 0.95$	B&B	11.9	10	24h
UB=24.2	EM	n.a.	n.a.	n.a.
	QCLP	n.a.	n.a.	n.a.
	QCLP-Mealy	n.a.	n.a.	n.a.
	BPI	10.7 $\pm$ 0.40	40	21174.1
	IPI	17.9	40	88747.0
	IPI-LP	14.96	21	87749.0

Table 2: Continuation of Tab. 1.

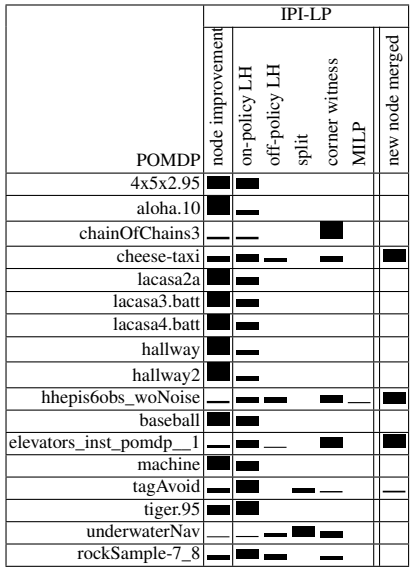

Table 3: Improvements performed by IPI-LP to obtain controllers reported in Tab. 1 and 2. In columns 'node improvement' through 'MILP', the vertical bars represent the ratio of the number of times a particular method was used to the number of times all methods were used. Column 'new node merged' shows the ratio of the new nodes merged into an existing node to all new nodes that were computed during escape.

was needed on the hhepis6obs_inst_woNoise domain only. This domain has a very challenging local optimum, which all the other methods could not escape. This clearly shows the advantage of our provable procedure to escape local optima. Additional testing, when the other escape methods were switched off, showed that MILP-based escape was consistently providing escape whenever all the other methods failed or were not used. Merging new nodes (the last column) was important on some problems too.

8. CONCLUSION AND FUTURE WORK

Even though significant improvements were made, POMDP planning is still challenging. Additionally, numerous practical applications require finite-state controllers, which are even more challenging to compute. On the other hand, policy iteration methods, which compute finite-state controllers, are known to be very efficient in MDPs. In this paper, we bring closer the success of policy iteration in MDPs to POMDPs and show, for the first time, that policy iteration in POMDPs can be both scalable and robust against local optima. Our method is based on—and adds to—fundamental properties of POMDPs, does not rely on randomization, does not rely on upper bounds, and forward search is limited to one-step lookahead from selected beliefs. Certainly, integrating those additional features (especially upper bounds [8]) in future research can yield a much more powerful algorithm. It would also be interesting to extend IPI to multi-agent POMDPs where controllers are often used to represent policies [1, 27, 28].

Acknowledgements

The authors thank Craig Boutilier and Pascal Van Hentenryck for useful discussions. This research was sponsored by NSERC and MITACS.

REFERENCES

[1] C. Amato, D. Bernstein, and S. Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *JAAMAS*, 2009.

[2] C. Amato, D. S. Bernstein, and S. Zilberstein. Solving POMDPs using quadratically constrained linear programs. In *IJCAI*, pages 2418–2424, 2007.

[3] C. Amato, B. Bonet, and S. Zilberstein. Finite-state controllers based on Mealy machines for centralized and decentralized POMDPs. In *AAAI*, 2010.

[4] D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1287–1292, Edinburgh, Scotland, 2005.

[5] A. Cassandra, M. L. Littman, and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, pages 54–61, 1997.

[6] H.-T. Cheng. *Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, University of British Columbia, Vancouver, 1988.

[7] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research (JAIR)*, 24:49–79, 2005.

[8] M. Grześ and P. Poupart. POMDP planning and execution in an augmented space. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2014.

[9] M. Grześ, P. Poupart, and J. Hoey. Controller compilation and compression for resource constrained applications. In *Proceedings of International Conference on Algorithmic Decision Theory (ADT)*, 2013.

[10] M. Grześ, P. Poupart, and J. Hoey. Isomorph-free branch and bound search for finite state controllers. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[11] E. A. Hansen. An improved policy iteration algorithm for partially observable MDPs. In *Proceedings of Advances in Neural Information Processing Systems, 10*, pages 1015–1021. MIT Press, 1997.

[12] E. A. Hansen. Solving POMDPs by searching in policy space. In *Proc. of UAI*, pages 211–219, 1998.

[13] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[14] H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*, 2008.

[15] G. McCormick. Computability of global solutions to factorable nonconvex programs: Part I convex underestimating problems. *Mathematical Programming*, 10, 1976.

[16] H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proc. of ICML*, pages 569–576, 2005.

[17] N. Meuleau, K.-E. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proc. of UAI*, pages 417–426, 1999.

[18] J. Pajarinen and J. Peltonen. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *Advances in Neural Information Processing Systems*, pages 2636–2644, 2011.

[19] K.-E. K. Pascal Poupart and D. Kim. Closing the gap: Improved bounds on optimal POMDP solutions. In *nternational Conference on Automated Planning and Scheduling (ICAPS)*, Freiburg, Germany, 2011.

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

[21] P. Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, University of Toronto, Toronto, Canada, 2005.

[22] P. Poupart and C. Boutilier. Bounded finite state controllers. In *Proc. of NIPS*, 2003.

[23] P. Poupart, T. Lang, and M. Toussaint. Analyzing and escaping local optima in planning as inference for partially observable domains. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, editors, *ECML/PKDD (2)*, volume 6912 of *Lecture Notes in Computer Science*, pages 613–628. Springer, 2011.

[24] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[25] T. Smith and R. G. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[26] E. Sondik. The optimal control of partially observable decision processes over the infinite horizon: Discounted cost. *Operations Research*, 26(2):282–304, 1978.

[27] E. Sonu and P. Doshi. Generalized and bounded policy iteration for finitely-nested interactive POMDPs: Scaling up. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 1039–1048, 2012.

[28] E. Sonu and P. Doshi. Scalable solutions of interactive POMDPs using generalized and bounded policy iteration. *Journal of Autonomous Agents and Multi-Agent Systems*, 2014.

[29] M. T. J. Spaan and N. Vlassis. Perseus: randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24(1):195–220, 2005.