An Approximate Algorithm for Solving Oracular POMDPs

Nicholas Armstrong-Crews
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
narmstro@cs.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
veloso@cs.cmu.edu

Abstract— **We propose a new approximate algorithm, LA-JIV (Lookahead J-MDP Information Value), to solve Oracular Partially Observable Markov Decision Problems (OPOMDPs), a special type of POMDP that rather than standard observations includes an "oracle" that can be consulted for full state information at a fixed cost. We previously introduced JIV (J-MDP Information Value) to solve OPOMDPs, an heuristic algorithm that utilizes the solution of the underlying MDP and weighs the value of consulting the oracle against the value of taking a state-modifying action. While efficient, JIV will rarely find the *optimal* solution. In this paper, we extend JIV to include lookahead, thereby permitting arbitrarily small deviation from the optimal policy's long-term expected reward at the cost of added computation time. The depth of the lookahead is a parameter that governs this tradeoff; by iteratively increasing this depth, we provide an anytime algorithm that yields an ever-improving solution. LA-JIV leverages the OPOMDP framework's unique characteristics to outperform general-purpose approximate POMDP solvers; in fact, we prove that LA-JIV is a poly-time approximation scheme (PTAS) with respect to the size of the state and observation spaces, thereby showing rigorously that OPOMDPs are "easier" than POMDPs. Finally, we substantiate our theoretical results via an empirical analysis of a benchmark OPOMDP instance.**

I. INTRODUCTION

Partially Observable Markov Decision Problems (POMDPs) comprise a useful theoretical tool for robots making decisions in a time-varying stochastic process with hidden state. POMDPs can represent a wide array of real-world problems, providing elegant solution techniques with strong guarantees on optimality. Unfortunately, solving POMDPs optimally is a computationally intractable task in general [1] [2]. In contrast, Markov Decision Processes (MDPs) can be solved in poly-time [3]; but the trade-off is representational power. MDPs assume the robot always knows the system's state, which is much less common in real-world problems.

We have previously introduced the Oracular POMDP framework [4], a framework "between" POMDPs and MDPs that captures the benefits of both and in some situations works better than either; see Figure I. The *oracle* is a special type of observation that gives perfect state information, but consulting the oracle uses up an action and incurs a penalty to reward. Additionally, the OPOMDP assumes that no other observations exist besides the oracle, allowing the actions to be partitioned into those that acquire information and those that have some effect on the world.

Besides being interesting from a scientific perspective, OPOMDPs demonstrate real-world significance. It is a natural sensing modality to ask a knowledgeable source when no other information is available. Examples of such scenarios include: covert operations, in which using an active sensor might give away the robot's position; medical diagnosis, in which doctors use expensive or invasive procedure to reveal the presence or severity of a disease; expert systems, in which the robot consults a virtually omniscient source; and human-computer dialog systems, in which the machine must ask the human for information about his or her intentions.

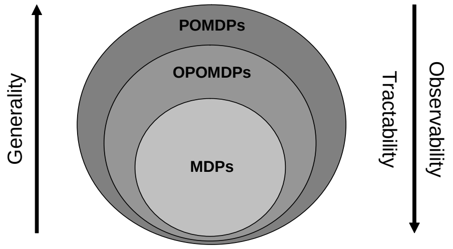


Fig. 1. Oracular POMDPs are "between" MDPs and POMDPs in terms of generality, observability, and tractability.

Prior examples of frameworks between POMDPs and MDPs include: unobservable MDPs [2], which are POMDPs that produce no observations; even-odd POMDPs [5], which acquire perfect state information at every other timestep; and "MDPs with observation costs", which have no observations and an oracle action [6]. The lattermost is quite similar to our framework, but differs in two important ways: it requires that consulting the oracle is optimal after some finite period of time; and it defines a policy only for belief states with perfect information. The first requirement precludes unobservable MDPs, as well as our "Wizard's Curse" example problem (specified later, in Section V); while the second requirement precludes any POMDP in which the robot begins with uncertainty (e.g., the "kidnapped robot" problem). Our framework encompasses all these cases.

Furthermore, some authors have modified an MDP's state space [7] and/or reward function [8] to handle uncertainty; however, this practice is insufficient in general [4]. In an unrelated vein, there is work that refers to an "oracle" that gives perfect state information, but only during a prior model-learning phase rather than at execution time [9].

In [4], we proposed an algorithm to solve OPOMDPs, called JIV (J-MDP Information Value), that extends Q-MDP [10] to the OPOMDP framework. JIV achieves high reward with provably small computation time; however, it is an heuristic algorithm, meaning that it will typically fall short of the optimal solution.

To address JIV's shortcomings, in this work we present Lookahead J-MDP Information Value (LA-JIV), an approximate, anytime algorithm that converges to the optimal value function and thus the optimal policy. LA-JIV is a confluence of ideas from JIV [4], Heuristic Search Value Iteration (HSVI) [11], and policy iteration for MDPs with observation costs [6]. Like HSVI, it uses a forward search via heuristic, employing both upper and lower bounds on the value function; however, LA-JIV's search method, backup operator, initial bounds, and control policy are all novel. As in JIV, these novelties rely primarily on the OPOMDP's partitionable action set (strictly information-gathering actions and strictly domain-level actions) and the importance of "pure beliefs" at the corners of the simplex.

The organization of this paper is as follows: first we review OPOMDPs and how JIV solves them; followed by introducing LA-JIV and an in-depth analysis of its properties; then we show error bounds, convergence, and computational complexity; and finally we present initial empirical results on a benchmark OPOMDP domain.

II. Oracular POMDPs

We now take a moment to review the three frameworks (MDPs, POMDPs, and OPOMDPs) and specify our notations.

A. MDPs and POMDPs

The Markov Decision Process, or MDP, is a framework for modeling a robot trying to maximize its reward in a time-varying, stochastic process. It asserts the Markov property, that the robot's next state depends only upon its previous state and its action choice. In this paper, we consider the case of discounted reward and infinite horizon. The MDP is specified by a tuple $(\gamma, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, whose elements correspond respectively to the scalar discount factor; the finite set of states; the finite set of actions; the stochastic transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \Pi(\mathcal{S})$; and the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$.

The objective of the robot is to maximize long-term expected reward $E[\sum_0^\infty \gamma^t r_t]$. The typical solution method is to determine a stationary policy $\pi : \mathcal{S} \mapsto \mathcal{A}$, completely specifying the robot's behavior. The optimal policy can be found by solving the dynamic programming equation $J^*(s) \equiv \max_a \mathcal{R}(s,a) + \sum_{s'} \mathcal{T}(s,a,s')J^*(s')$. There are many techniques to solve this equation, such as value iteration and policy iteration.

The Partially Observable Markov Decision Process, or POMDP, is a generalization of the MDP framework that allows for incomplete state knowledge. It adds the concept of observations, which are generated stochastically from states, providing the robot with only limited state information. The POMDP tuple is $(\gamma, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O})$, containing the same elements as the MDP tuple, plus: Ω, the finite set of possible observations; and $\mathcal{O} : \mathcal{S} \times \mathcal{A} \mapsto \Pi(\Omega)$, the observation function. A robot's *belief state* is a discrete probability distribution that summarizes its observation history, action trajectory, and initial knowledge; the belief state lies in the probability simplex $\mathcal{B} \equiv \Pi(\mathcal{S})$. Given the robot's action choices, it can maintain a proper belief state via a Bayesian update at each timestep. The belief state is a sufficient statistic to act optimally, meaning the POMDP is Markovian in belief state. Hence, the POMDP is equivalent to an MDP in the continuous belief space simplex (called the *belief MDP*), and so can be solved by dynamic programming [12]. Unfortunately, solving POMDPs optimally has been proven PSPACE-hard [1]. The literature is rich with various optimal and heuristic solution techniques (introduced individually as well as surveyed [13]), but nonetheless, computation time remains the primary limiting factor for using POMDPs in realistically large domains.

B. OPOMDPs

In the Oracular POMDP, there exists an oracle in lieu of observations.[1] The oracle provides the robot perfect state information, but consulting the oracle incurs an immediate penalty to reward as well as the opportunity cost of foregoing a domain-level (a.k.a. state-altering) action for the current timestep.

One can view OPOMDPs as a special case of POMDPs. Consider a POMDP with one action that produces observations unique to each state, removing all ambiguity, while all other actions produce an uninformative "null" observation; such a POMDP is equivalent to an OPOMDP. Conversely, an OPOMDP with oracle cost zero is equivalent to an MDP whose actions correspond to a domain-level action followed by an oracular consultation. We say, then, that OPOMDPs are strictly "between" POMDPs and MDPs, with respect to both generality and observability.

Formally, the OPOMDP tuple is $(\gamma, \lambda, \mathcal{S}, \mathcal{A}, \tau, \rho)$, where λ is the oracle cost; $\tau : \mathcal{B} \times \mathcal{A} \mapsto \mathcal{B}$ is the transition function, as in the belief MDP; and $\rho : \mathcal{B} \times \mathcal{A} \mapsto \mathcal{R}$ is the expected reward function, also as in the belief MDP. The remaining components are defined as in the MDP. [2] The OPOMDP tuple is instantiated from its underlying MDP: $\mathcal{S} \equiv \mathcal{S}^{\text{MDP}}$, $\mathcal{A} \equiv \mathcal{A}^{\text{MDP}} \cup \{o\}$, and

$$\tau(b,a)|_{s'} \equiv \begin{cases} \sum_s b(s)\mathcal{T}^{\text{MDP}}(s,a,s') & a \in \mathcal{A}^{\text{MDP}} \\ I(s' = s_{\text{oracle}}) & a = o \end{cases}$$

$$\rho(b,a) \equiv \begin{cases} \sum_s b(s)\mathcal{R}^{\text{MDP}}(s,a) & a \in \mathcal{A}^{\text{MDP}} \\ \sum_s b(s)\mathcal{R}^{\text{MDP}}(s,\text{NO\_OP}) & a = o \end{cases}$$

Note that s' indexes the vector-valued output of τ, $I(\cdot)$ is the indicator function, and s_{oracle} is the true state told by the oracle. We also use the following miscellaneous notations throughout the paper:

[1] In future work, we will extend the framework to include both oracles and imperfect observations.

[2] We assume that the NO_OP domain-level action occurs simultaneously with the oracle action; it is just a self-transition in the underlying MDP.

- a *pure belief* is a belief with perfect state information ($b_s^{\text{pure}} = 1$ at s and 0 elsewhere);
- overbars and underbars indicate upper and lower bounds, respectively ($\bar{\text{J}}$, $\underline{\text{J}}$), while double bars ($\bar{\underline{\text{J}}}$) indicate the expression applies to both bounds;
- a "hat" indicates an estimate, while an asterisk indicates the optimal value — e.g., $\widehat{J}$ estimates J^*;
- we refer to the *bound width* at belief b by $\psi(b) \equiv \bar{\text{J}}(b) - \underline{\text{J}}(b)$; and
- Γ represents a vector set whose max specifies a value function; i.e., $J(b) = \max_{\alpha \in \Gamma} b \cdot \alpha$.

III. The LA-JIV Approximate Algorithm

First, we describe our previous heuristic algorithm, JIV, and its benefits and drawbacks; then we introduce LA-JIV, our approximate algorithm, that addresses the shortcomings of JIV. Note that a POMDP solver is said to be *approximate* if it produces sequence of value functions that converge to within ϵ of the optimal value function, given enough time; otherwise, it is said to be *heuristic*.

A. JIV (J-MDP Information Value)

J-MDP Information Value, or JIV, was our initial proposed heuristic algorithm for solving OPOMDPs [4]. It retrofits the Q-MDP heuristic for the OPOMDP, allowing it to overcome Q-MDP's main flaw and select information-gathering actions (i.e., consult the oracle). Meanwhile, it remains computable in poly-time. The resulting bound on the value function is an upper bound, since it uses the optimistic Q-MDP value function estimate.

$$\widehat{Q}^{\text{ JIV}}(b,a) = \rho(b,a) + \begin{cases} \gamma \widehat{J}^{\text{ Q-MDP}}(b) & a \neq o \\ -\lambda + \gamma\, b \cdot J^{\text{MDP}} & a = o \end{cases}$$

$$\widehat{J}^{\text{ JIV}}(b) = \max_a \widehat{Q}^{\text{ JIV}}(b,a)$$

$$\widehat{J}^{\text{ Q-MDP}}(b) = \max_{a \neq o} \left(\rho(b,a) + \gamma\, \tau(b,a) \cdot J^{\text{MDP}} \right)$$

JIV is efficient and empirically shows good performance in accumulated reward; however, it is not without certain drawbacks: 1) it is a greedy heuristic algorithm, meaning that it does not converge to the optimal value function; 2) it provides no (non-trivial) bounds on regret; and 3) it is not guaranteed to perform better than Q-MDP on all OPOMDPs. These shortcomings motivated the development of LA-JIV, which indeed surmounts them all (at the cost of added computation time). Note that LA-JIV subsumes JIV, in that it uses JIV for its initial bounds as described in Section III-C.

B. LA-JIV (Lookahead J-MDP Information Value)

LA-JIV is a forward heuristic search that utilizes both upper and lower bounds to focus the search and to prune the search tree. In this respect, it is similar to general POMDP solvers; but it achieves better performance by capitalizing on several key insights that pertain specifically to OPOMDPs:

- we can achieve a good initial upper bound using JIV;
- we can achieve a fairly good policy, and thus a fairly good initial lower bound, using extant hybrid MDP/POMDP techniques;
- since the oracle always produces a pure belief, we use the estimated value of those pure beliefs *much* more often; and
- the branching factor for consulting the oracle is $|\mathcal{S}|$, while it is only $|\mathcal{A}|$ for examining domain-level actions (typically, $|\mathcal{S}|$ is much larger than $|\mathcal{A}|$).

In the following subsections, we describe how LA-JIV takes advantage of each of these OPOMDP characteristics.

C. Initial Bounds

To get an upper bound over the entire belief space, we simply use JIV, which gives us a convex upper bound defined by the max of the hyperplanes $\widehat{Q}^{\text{ JIV}}$. It is a valid upper bound because it uses the optimistic Q-MDP value function estimate.

Our lower bound springs from the even-odd POMDP [5], a simple MDP/POMDP hybrid in which the robot receives full state information at every even timestep, but that knowledge is corrupted during the odd timestep. The key insight of [5] was that any even-odd POMDP is equivalent to a fully observable MDP with a discount factor γ^2 and an action set consisting of the cross product of $\mathcal{A}$ and $\mathcal{O}$. They call this MDP a 2MDP.

The relationship between OPOMDPs and even-odd POMDPs is apparent: if we restrict an OPOMDP's set of policies to those that consult the oracle every other timestep, we have an even-odd POMDP (and its corresponding 2MDP). Note that this is *not* a reduction from the OPOMDP to the 2MDP; we are restricting the search space of solutions for the OPOMDP to those that solve the (embedded) even-odd POMDP, and it so happens that the even-odd POMDP is reducible to the 2MDP.

We require one slight modification to the 2MDP in [5], that the 2MDP's reward function must be reduced by the oracle cost (once discounted), since in our case, the information does not come freely: $\mathcal{R}^{\text{2MDP}}(s,a) \equiv \rho(b_s^{\text{pure}}, a) + \gamma\,(\rho(\tau(b,a),o) - \lambda)$. Otherwise, the elements in the 2MDP tuple are the same as those in the MDP.

Solving the 2MDP gives us an estimate of the original OPOMDP's value function at the pure beliefs. To estimate the value function over the rest of the belief space, we can perform a single backup using the oracle action; thereafter, the even-odd assumption is met and the value for executing π^{2MDP} is exact. We call this value function estimate Q-2MDP, due to its similarity to Q-MDP. Formally:

$$J^{\text{ Q-2MDP}}(b) = \begin{cases} J^{\text{2MDP}}(s) & \text{if } \exists s \ni b(s) = 1 \\ \rho(b,o) - \lambda + \gamma\, b \cdot J^{\text{2MDP}} \end{cases}$$

An important and complementary difference from the JIV upper bound, this approximation is a *lower* bound on the OPOMDP value function. It uses the exact solution to the even-odd POMDP, which provides merely one of many candidate policies for the OPOMDP. Since the OPOMDP value function is a max over the value of these policies, it is clear that Q-2MDP provides a lower bound. In most general POMDP solvers, the lower bound is initialized with the much more conservative bounds of 1) assuming the robot is in the worst state and stays there forever, or 2) choosing a policy which takes the same action forever and evaluating its value [11]. We can see intuitively that the lower the oracle cost, the more closely the optimal policy (and therefore value function) will resemble that of Q-2MDP.

Figure 2(a) depicts the initial bounds.

D. Looking Ahead

Several prior works have employed the technique of forward exploration (or *lookahead*) from a known initial belief (HSVI [11], PBVI [14], RTDP [15]) to avoid unreachable portions of the belief space, quite common in real-world POMDPs. Moreover, with approximate algorithms, an exploration of finite depth is sufficient to ensure the error is less than a given threshhold; as a result, the set of beliefs to consider is reduced from the $|S|$-dimensional simplex to a finite set.

Of course, lookahead comes at a price: the tree of possible beliefs grows exponentially with branching factor $|\mathcal{A}||\mathcal{O}|$, and to solve the POMDP, we might potentially have to visit each node in the tree. The runtime would then be $O(\exp(|\mathcal{A}||\mathcal{O}|))$.

Luckily, with OPOMDPs, the price of lookahead is somewhat lessened. As discussed in [4], an OPOMDP's action set is factorizable into information-gathering actions and domain-level actions. As a result, we can factorize the exponential dependencies of lookahead so that the algorithm's runtime scales as $O(\exp(|\mathcal{A}|) + \exp(|\mathcal{S}|))$. Additionally, we know beforehand several very important beliefs: the "pure" beliefs at the corners of the simplex, corresponding to certainty of being in a given state. Due to the oracle action, these are always reachable in a single timestep from any other belief. Caching these beliefs allows us to further reduce the complexity to $O(|\mathcal{S}|\exp(|\mathcal{A}|))$. A final benefit of concentrating on pure beliefs is that these beliefs are totally sparse (only a single entry), and typically will transition to very sparse beliefs as well. Computationally, it is much faster to compute updates and transitions on sparse beliefs.

Hansen's policy iteration for MDPs with observation costs (hereafter referred to as "pure belief policy iteration," or PBPI) takes advantage of pure beliefs as well, but to an extreme: it maintains the value function *only* at those beliefs. As a result, values for impure beliefs encountered in the search do not reflect the work done to update them at earlier iterations. In contrast, LA-JIV maintains vector-based bounds, which are updated during the search, to fully utilize prior work. Maintaining upper and lower bounds also allows more pruning. Finally, the heart of LA-JIV is value iteration, while PBPI is an instance of policy iteration.

In LA-JIV, we look ahead from each of the pure beliefs, performing $|\mathcal{S}|$ forward searches. Our search method is iterative deepening A* (or IDA*). Typically, IDA* iteratively loosens limits on the heuristic value $f(n)$; however, LA-JIV limits the search depth instead, since *it* is the determining factor in computation, memory, and discounting. Hence, for a given depth limit T, we perform an A* forward search for each pure belief (to maximum depth T). After completing the search for each pure belief, we increment T. We stop when the bound width for all pure beliefs is less than ϵ (for an anytime version of LA-JIV, let ϵ be zero). See Algorithm 1 and Figure 2.

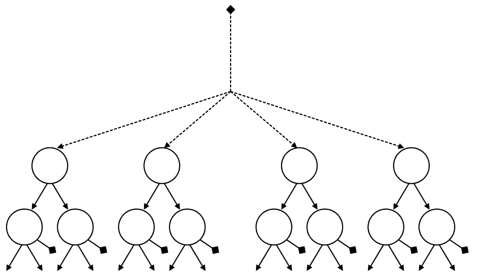


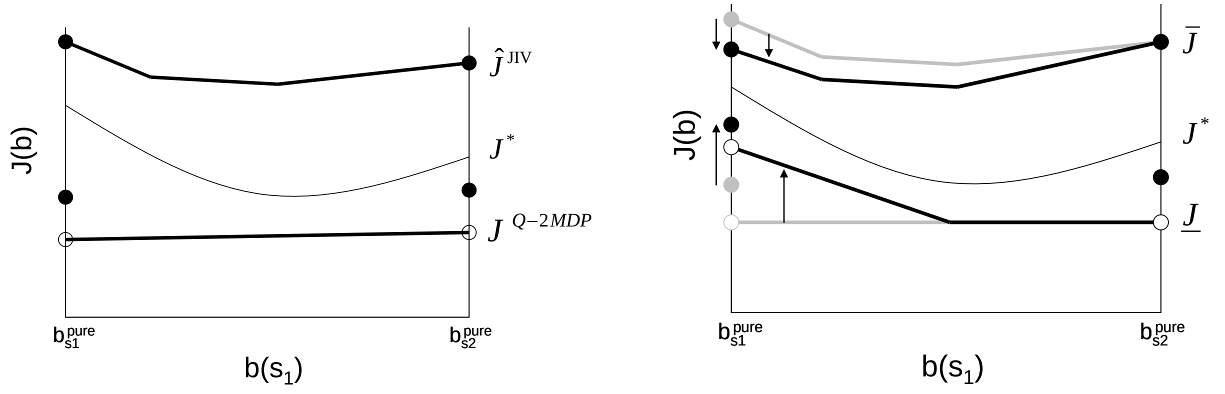
Fig. 3. The LA-JIV search tree structure, where the diamond indicates the oracle action. LA-JIV, in contrast to HSVI, does not expand the tree further after consulting the oracle, because it would be a copy of the whole tree.

Algorithm 1 LA-JIV(ϵ)

```
{Initialize vector-based and point-based bounds}
$\Gamma \leftarrow (\Gamma^{\text{JIV}}, \Gamma^{\text{Q-2MDP}})$
$\forall s,\ \overline{\underline{\mathbf{J}}}_s^{\text{pure}} \leftarrow (\widehat{J}^{\text{JIV}}(b_s^{\text{pure}}), J^{\text{Q-2MDP}}(b_s^{\text{pure}}))$
while $\max_s \psi(\overline{\underline{\mathbf{J}}}_s^{\text{pure}}) > \epsilon$ do
  {IDA*}
  $T \leftarrow 1$
  for all $s \in \mathcal{S}$ do
    $\overline{\underline{\mathbf{J}}}_s^{\text{pure}} \leftarrow$ A*$(s, T)$
  end for
  $T \leftarrow T + 1$
end while
```

The node data structure we maintain in the search is a tuple (b, R, t, a) consisting of: b, the belief point; R, the expected accumulated reward from descending the tree $E[\sum_t r_t]$; the depth t; and the action choice a that led to this node (undefined, for the root). A node corresponds to a choice of t actions from the root.

Our version of A* is shown as Algorithm 2. It maintains a priority queue, popping the node with maximal priority at each iteration and expanding that node. Adopting the common search notation, the priority f of a node n is given by $f(n) = g(n) + h(n)$, where $g(n)$ is the expected accumulated reward $n.R$ and $h(n)$ is an estimate of the discounted expected future reward. In our case, we are concerned with maximizing value rather than minimizing cost, so an admissible heuristic is one that *overestimates* the future reward. As such, we choose the (discounted) upper bound value of current node: $h(n) = \gamma^{n.t}\overline{\mathbf{J}}(n.b)$. We can prune whenever the depth is too large or the priority (an upper bound on the value for the pure belief at the root) of a node is less than the lower bound of the value of that pure belief. Furthermore, we stop when we pop a node corresponding to the oracle action, since this indicates that we need a better bound on the value of the oracle action,



(a) Initial Bounds

(b) Backup

Fig. 2. Left: initial bounds for a simple two-state, three-action OPOMDP. Right: a backup, both vector-based and point-based, for both upper and lower bounds, on the search rooted at the pure belief for state s_1.

achieved by another iteration of the algorithm to greater depth.

Algorithm 2 A*(s, T)

```
OPEN ← $\{b_s^{\text{pure}}, 0, 0, \text{NULL}\}$
while OPEN $\neq \{\}$ AND $\psi(\overline{\underline{J}}_s^{\text{pure}}) > \epsilon$ do
    node ← $\arg\max_{n \in \text{OPEN}} f(n)$
    OPEN ← OPEN$\setminus\{$node$\}$
    BACKUP()
    if $f(\text{node}) \leq \underline{J}_s^{\text{pure}}$ OR node.$a = o$ then
        break
    else
        OPEN ← OPEN $\cup$ EXPAND(node, $s, T$)
    end if
end while
return
```

Algorithm 3 EXPAND(node, s, T)

```
children ← $\{\}$
for all $a \in \mathcal{A}\setminus\{o\}$ do
    $R \leftarrow$ node.$R + \gamma^{\text{node}.t}\rho(\text{node}.b, a)$
    child ← $\{\tau(\text{node}.b, a), R, \text{node}.t + 1, a\}$
    if $f(\text{child}) > \underline{J}_s^{\text{pure}}$ AND child.$t \leq T$ then
        children ← children $\cup$ child
    end if
end for
```

HSVI also uses the value function upper bound as its search heuristic, but it searches the AND/OR tree, in which observations correspond to the branches of an AND node. PBPI uses a lower bound on the value function for a search heuristic, which is an inadmissible heuristic and thus limits its ability to prune.

E. Backing Up

LA-JIV maintains both vector-based bounds, and, at pure belief points, point-based bounds. Since it performs searches rooted at the pure beliefs, it can update the values of those pure beliefs from the search tree. Then, it can use the updated values of the pure beliefs to update the vector-based bounds for the rest of the belief space.

1) Point-based Upper Bounds: Since the search heuristic is admissible, the current node has a greater upper bound than any nodes that will later be encountered in the search. As such, we can update the upper-bounding value of the pure belief at the root whenever we pop a node from the queue. We initialize these bounds by evaluating the initial vector-based upper bounds at the pure beliefs.

2) Vector-based Upper Bounds: Throughout the algorithm, we maintain only $|\mathcal{A}|$ vectors for the upper bound. These are initialized as the vectors from JIV, each representing a single action and thereafter assuming perfect information. The value of perfect information is estimated by J^{MDP}. Then, when we update the upper bounding value of a pure belief, we can recalculate the JIV bounds by replacing J^{MDP} with the updated estimates of the value of perfect information. Hence, we maintain a fairly tight upper bound while avoiding the overhead of HSVI's convex hull calculation for its upper bound.

3) Point-based Lower Bounds: These bounds, existing only at pure beliefs, are initialized by solving the 2MDP; recall that they are slightly larger than the value of the vector-based lower bounds at the pure belief points, since here it is not necessary to consult the oracle before using the 2MDP policy. During the search, at a particular node, we know that we can achieve *at least* the value given by the accumulated reward plus the discounted lower bound on future reward (because the node's descent path, followed by the lower bound policy, is one of many potential policies, over which the value function is a max). If this value is greater than the lower bounding value of the pure belief at the root of the search tree, then we can update that value.

4) Vector-based Lower Bounds: These are initialized to the single vector given by Q-2MDP. Unfortunately, this case is not analogous to the upper bound; when we update the point-based lower bound, we cannot update the vector-based lower bound as a result. Hence, we add vectors to the lower bounding vector set via standard vector backups [11].

```
Algorithm 4 BACKUP(node)
$\bar{\mathrm{J}}_s^{\text{pure}} \leftarrow \min\{f(\text{node}), \bar{\mathrm{J}}_s^{\text{pure}}\}$
$\underline{\mathrm{J}}_s^{\text{pure}} \leftarrow \max\{\text{node}.R + \gamma^{\text{node}.t}\underline{\mathrm{J}}(\text{node}.b), \underline{\mathrm{J}}_s^{\text{pure}}\}$
$\bar{\Gamma} \leftarrow \mathcal{R}(a, \cdot) + \gamma \sum_{s'} \mathcal{T}(\cdot, a, s')\bar{\mathrm{J}}_{s'}^{\text{pure}}$
VECTORBACKUP(node.$b$)
```

A "backup" for LA-JIV consists of performing each of the updates described above; see Figure 2(b) and Algorithm 4 (VECTORBACKUP here is no different from in the literature [11]). Since the vector-based lower bound is the only one that grows in size, it is the only one we need to periodically prune. For efficiency, we only prune when one vector pointwise dominates another; and we only test for pruning when the number of vectors has grown by 50%.

F. Action Selection

Once we have terminated the algorithm and have good value function bounds, we must use them somehow to choose an action (i.e., generate a policy). Using a single step of lookahead, we choose the action that minimizes the upper bound on regret: $\pi(b) \equiv \arg\min_a \left(\left(\max_{a' \neq a} \bar{\mathrm{Q}}(b, a')\right) - \underline{\mathrm{Q}}(b, a)\right)$. Note that this method of policy selection defines a policy at an arbitrary belief point, not just at a pure belief (as in PBPI). Furthermore, this method differs from HSVI and PBVI, which use only the lower bound to induce a policy and which might choose an action with higher regret.

IV. THEORETICAL RESULTS

We now summarize several key theoretical results about OPOMDPs and LA-JIV.

- A search to depth T will result in $O(|\mathcal{S}||\mathcal{A}|^T)$ expansions (we can ignore the iterative deepening because the cumulative expansions at lesser depths are dominated by the expansions for depth T).
- Each node expansion adds at most one vector to the lower bound; hence, the maximum number of vectors $|\Gamma|$ in the lower bound is also $O(|\mathcal{S}||\mathcal{A}|^T)$.
- The computational cost of an expansion is $O(|\mathcal{S}||\mathcal{A}||\Gamma|)$, since each child must have its lower-bound value computed by taking the dot product of the belief (size $|\mathcal{S}|$) with each vector (and this lower bound computation dominates the cost of a node expansion).
- Hence, the computational cost of LA-JIV to depth T is $O(|\mathcal{S}|^3|\mathcal{A}|^{2T+1})$. Note that this term is exponential in $|\mathcal{A}|$, but not in $|\mathcal{S}|$ (nor $|\mathcal{O}|$, the number of observations for the equivalent POMDP).

The following results regarding HSVI (see [11]) also hold true for LA-JIV, but at *all* pure beliefs rather than just a single starting belief:

- A search to depth T will bound the error of the pure belief values by the initial bound width discounted T times, or $\epsilon < \gamma^T \max_b(J^{\text{ Q-2MDP}}(b) - J^{\text{ JIV}}(b))$.
- Conversely, the maximum depth required for a given precision ϵ on the value of the pure beliefs is $T \leq \lceil \log_\gamma(\epsilon / \max_b(J^{\text{ Q-2MDP}}(b) - J^{\text{ JIV}}(b))) \rceil$.
- As T increases, the value function bounds will converge to the optimal value function, thus producing the optimal policy.

Since the maximum depth required for a given ϵ does not depend on the size of $\mathcal{S}$ (nor $|\mathcal{O}|$, the number of observations for the equivalent POMDP), we can conclude that the computational complexity of LA-JIV, in the worst case, is polynomial in the size of the state space $|\mathcal{S}|$, but exponential in the number of actions $|\mathcal{A}|$. Since this is true for any ϵ, we have shown that LA-JIV is a poly-time approximation scheme (or PTAS) with respect to $|\mathcal{S}|$; that is, for any required accuracy ϵ, LA-JIV(ϵ) is an algorithm, poly-time in $|\mathcal{S}|$, that approximates the optimal value function to within ϵ error. In essence, this result substatiates our intuition that OPOMDPs are "easier" than POMDPs, whose search tree grows exponentially in $|\mathcal{O}|$ as well (and recall that the equivalent POMDP for an OPOMDP has $|\mathcal{O}| = |\mathcal{S}| + 1$).

V. EMPIRICAL RESULTS

In [4], it was necessary to define a new benchmark domain due to the lack of POMDPs in the literature that meet the Oracular POMDP requirements. In previous work, we introduced the Wizard's Curse domain, a grid-world problem with an oracle, a goal, and several obstacles [4]; however, we omit the precise specification for brevity.

We ran our tests on an Athlon XP 3800+ with 1 GB RAM. We expect our runtimes to be an extremely lowball estimate of LA-JIV's potential performance, since we did not spend time tuning parameters and since the algorithm was implemented in MATLAB. To compare, HSVI's second implementation outperformed its first implementation by up to 100 times [11].

Using scaled versions of this domain, we measured the solve-times using LA-JIV, JIV, HSVI, and PBPI (with added iterative deepening and ϵ termination criterion, to ensure convergence and termination on this problem) and HSVI (version 2). We chose HSVI since it is one of the most scalable general POMDP solvers and has a freely available implementation; we chose PBPI because it is specialized to solve many problems that fit the OPOMDP framework (however, we re-implemented it using MATLAB, as there was no extant available implementation).

Our results regarding computational efficiency are summarized in Table I. We tested the Wizard's Curse problem, as above, and a scaled-up version of the same problem, five times larger in each dimension; with each, we used oracle cost $\lambda = .25$, discount factor $\gamma = .75$, and precision $\epsilon = .001$. The scaled-up version comprised 900 states, taking 54 minutes for LA-JIV to solve and 120 minutes for HSVI2 (in six hours, PBPI did not complete). Also note that HSVI guarantees the ϵ bound from a known starting belief, whereas LA-JIV guarantees it from *any* pure belief. For the same guarantee from HSVI, it would have to be run $|\mathcal{S}|$ times in succession (though it could re-use earlier bounds).

TABLE I
COMPARISON OF EMPIRICAL SOLVE-TIMES

	36 states	900 states
LA-JIV	2.4	3219
JIV	.03	6.5
PBPI	74.5	$>$ 21000
HSVI	23	7200

In Figure 4, we examine the development of the value function approximation as we increase the maximum search depth T, for both LA-JIV and PBPI. For simplicity, we examine only the approximation at b_0. We can see that LA-JIV converges quickly to within $\epsilon = .001$, requiring only four iterations. PBPI, in comparison, required seven iterations. Although each iteration is more expensive in LA-JIV, since we maintain vector-based bounds, Table I shows that the added overhead pays off, overall.

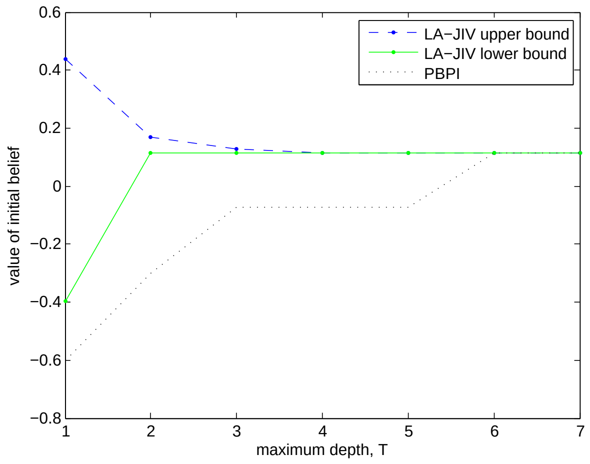


Fig. 4. Comparison of LA-JIV's upper and lower bounds to PBPI's value function estimate, around b_0, as we increase the maximum search depth T on the 36-state Wizard's Curse problem.

We ran the output policy of each algorithm in simulation, but the average accumulated rewards (over 100 trials) for each were within measurement error of one another. This result is not surprising, since LA-JIV is initialized with JIV, which we showed in [4] was within measurement error of HSVI's policy.

Finally, in Figure 5, we examine the effect of oracle cost on the efficiency of LA-JIV. As predicted, lower oracle costs correspond to lower runtime, since the pure beliefs are visited more often and thus our caching is more effective.

VI. CONCLUSION AND FUTURE WORKS

LA-JIV is an efficient, approximate OPOMDP solver that takes advantage of the OPOMDP's factorized action set and the strong connection to the underlying MDP. It uses these characteristics to: achieve better initial bounds; search a tree of reduced complexity; and prune the search tree quickly by maintaining special bounds on the value of pure beliefs. It addresses JIV's main drawback, in that it converges to the optimal solution in the limit. We've also shown that LA-JIV is a poly-time approximation scheme for OPOMDPs. Finally, we've presented preliminary results verifying the efficacy of LA-JIV on the Wizard's Curse benchmark OPOMDP. In future work, we plan to implement LA-JIV in a human/robot team scenario as well as investigate POMDPs with both oracular and imperfect information sources.

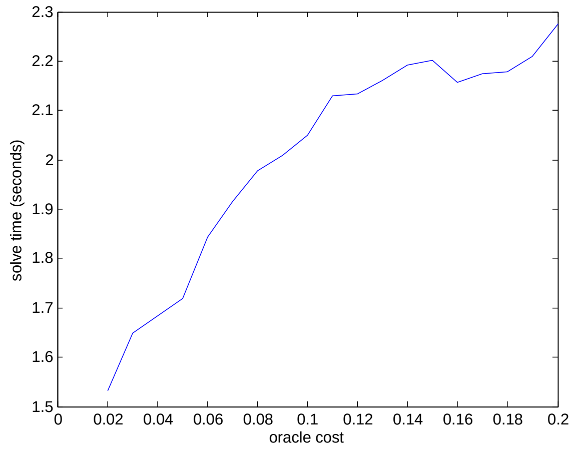


Fig. 5. Solve-time as a function of oracle cost.

Acknowledgments: Myriad thanks to Reid Simmons, Geoff Gordon, and Trey Smith for many helpful discussions.

REFERENCES

[1] C. Papadimitriou and J. Tsisiklis, "The complexity of markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[2] O. Madani, "Complexity results for infinite-horizon markov decision processes," Ph.D. dissertation, University of Washington, 2000.

[3] M. Littman, T. Dean, and L. P. Kaelbling, "On the complexity of solving markov decision problems," in *Proceedings of UAI-95*, 1995.

[4] N. Armstrong-Crews and M. Veloso, "Oracular POMDPs: A very special case," in *Proceedings of ICRA-07*, 2007.

[5] V. B. Zubek and T. Dietterich, "A POMDP approximation algorithm that anticipates the need to observe," in *Proceedings of PRICAI-00*, 2000.

[6] E. Hansen, "Markov decision processes with observation costs, Tech. Rep. UM-CS-1997-001, 1997.

[7] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal navigation – robot motion with uncertainty," in *Proceedings of AAAI-98*, 1998.

[8] F. Melo and M. Ribeiro, "Transition entropy in POMDPs," in *Proceedings of IAS-06*, 2006.

[9] R. Jaulmes, J. Pineau, and D. Precup, "Active learning in POMDPs," in *Proceedings of ECML-05*, 2005.

[10] M. Littman, A. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Proceedings of ICML-95*, 1995.

[11] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proceedings of UAI-04*, 2004.

[12] L. P. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence Journal*, vol. 101, no. 1-2, pp. 99–134, 1998. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(98)00023-X

[13] D. Aberdeen, "A (revised) survey of approximate methods for solving POMDPs," National ICT Australia, Tech. Rep., 2003.

[14] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of IJCAI-03*, 2003.

[15] H. Geffner and B. Bonet, "Solving large POMDPs by real time dynamic programming," in *Proceedings of AAAI-98*, 1998.