The Permutable POMDP: Fast Solutions to POMDPs for Preference Elicitation

Finale Doshi
CSAIL MIT
32 Vassar Street
Cambridge, MA 02139
finale@mit.edu

Nicholas Roy
CSAIL MIT
32 Vassar Street
Cambridge, MA 02139
nickroy@mit.edu

ABSTRACT

The ability for an agent to reason under uncertainty is crucial for many planning applications, since an agent rarely has access to complete, error-free information about its environment. Partially Observable Markov Decision Processes (POMDPs) are a desirable framework in these planning domains because the resulting policies allow the agent to reason about its own uncertainty. In domains with hidden state and noisy observations, POMDPs optimally trade between actions that increase an agent's knowledge and actions that increase an agent's reward.

Unfortunately, for many real world problems, even approximating good POMDP solutions is computationally intractable without leveraging structure in the problem domain. We show that the structure of many preference elicitation problems—in which the agent must discover some hidden preference or desire from another (usually human) agent—allows the POMDP solution to be solved with exponentially fewer belief points than standard point-based approximations while retaining the quality of the solution.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Markov Processes

General Terms

preference elicitation, planning, uncertainty

Keywords

decision-making under uncertainty

1. INTRODUCTION

In almost all real-world domains, automated agents must reason and plan with only incomplete, noisy information about their environments. Partially Observable Markov Decision Processes (POMDPs) are a desirable framework for many planning domains because POMDP policies allow agents to reason about their own uncertainty. In stochastic environments with hidden state and noisy observations, POMDPs optimally trade between actions that increase an agent's knowledge and actions that increase an agent's immediate reward. Applied to a variety of traditional AI problems, the POMDP framework has also found applications in cooperative multi-agent domains such as preference elicitation and dialog management. In these applications, POMDPs allow the agent to trade between asking the user for information about his or her intent and performing a useful action. Specific examples include:

- Determining a user's desired flight number in an automated telephone booking system (3-2000 slot values) [13].
- Determining reputations in electronic markets [7].
- Determining a user's desired destination on a robotic wheelchair (10 locations) [3].

One common feature of all these scenarios is that the agent must discover some piece of hidden information through a series of questions with potentially noisy or ambiguous answers. Once sufficiently certain, the agent must "submit" its response to another system or authority. The agent is penalized for the number of questions it asks—effectively, the time it takes to come to a decision—and for submitting a state estimate that does not match the true state.

A POMDP solver finds the optimal policy through a deep search through all of the scenarios that the agent may encounter in the future and chooses an action that maximizes the reward it expects to receive. Unfortunately, solving POMDPs is often difficult at best—even with approximate solution methods, POMDPs for realistic problems may take hours or days to solve, if they can be solved at all. In some senses, such time requirements may be immaterial since executing a policy is fast once the solution has been computed [2], but long computation times inhibit the agent's ability to tune its policy as more data is received[1].

Several works have suggested heuristics for POMDPs that have the structure of preference elicitation and dialog management problems. For example, Boutilier [2] uses quadratic approximators and Gaussians to approximate a complicated POMDP's value function. Williams and Young [13] create a mini-POMDP from the larger POMDP that contains only two states: the most likely state and a state corresponding to all other possible options.

The work in Williams and Young [13] alludes to common feature of preference elicitation scenarios. In many cases, the agent's optimal action *type* does not depend on the true hidden state, but only the agent's uncertainty in the true state. For example, if a media-playing agent is only 60% sure of a user's media request, the optimal action may be to ask the user for clarification, rather than potentially playing the wrong music. The clarification action is also likely to be the optimal action in all scenarios in which the system is uncertain about the media, regardless of the tune itself. However, the summary POMDP (and other similar approximation algorithms [8]) cannot take advantage of actions that gather specific information, such as disambiguating between pairs of similar states.

We present a principled approach for solving POMDP models

[1]We focus here on planning in a known domain rather than learning about new domains, but tractable planning is also a prerequisite for most learning algorithms.

Cite as: The Permutable POMDP: Fast Solutions to POMDPs for Preference Elicitation , Finale Doshi, Nicholas Roy , *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

that have the structure typically found in preference elicitation problems. Our approach depends crucially on the fact that for certain POMDP structures, the agent's optimal action *type* depends only on its uncertainty over states and not about which particular states the agent is confused. For example, it may not matter that the confusion is between "play CD" and "play TV" or between "make a coffee" and "make a copy"; the optimal action may be to try confirming the more likely request with the user. Under these conditions, we can modify existing solution algorithms for computing the policy without adding approximation error. We formalize these conditions in the "Permutable POMDP" and show that near-optimal point-based value function approximations to the permutable POMDP can be calculated using exponentially fewer belief points than standard POMDPs.

We detail the POMDP model and point-based approximate solution techniques in Section 2. Section 3 describes the specialized structure present in many preference elicitation POMDPs. We formalize that structure and its implications to solve for the agent's optimal policy in Section 4. Section 5 presents our results on a variety of problems. We present our conclusions in Section 6.

2. POMDP OVERVIEW

In this section, we provide background on the POMDP model and the point-based solution techniques that we will optimize in Section 4. A POMDP consists of the n-tuple $\{S,A,O,T,\Omega,R,\gamma\}$. S, A, and O are sets of states, actions, and observations. In the preference elicitation context, the states represent a property that is hidden from the agent. The observations are noisy measurements of this property—either from instruments or queries to the user. For example, in the dialog management setting, observations may correspond to noisy outputs from a voice recognition system. Actions include queries to gather information and a final action to submit the most likely state estimate to another system. We require that the state, action, and observation sets are discrete and finite.

The remaining components of the POMDP tuple describe how the world behaves. The transition function $T(s'|s,a)$ gives the probability $P(s'|s,a)$ of transitioning from state s to s' if taking action a. The observation function $\Omega(o|s,a)$ gives the probability $P(o|s,a)$ of seeing observation o from state s after taking action a. The reward $R(s,a)$ specifies the immediate reward for taking action a in state s. The reward function allows the system designer to specify what the "right" actions are in different states and the relatively penalties for information gathering actions. Finally, the discount factor $\gamma \in [0,1]$ weighs how much the agent values future rewards to current rewards.

Since the POMDP state is hidden, the agent cannot choose its actions based on knowing the true state. Instead, the agent must base its actions on the sequence of interactions that it has had with the user. Keeping an entire history of interactions can get quite cumbersome, but fortunately, a distribution over possible user states—known as a belief—is a sufficient statistic for the history. Suppose that the agent takes action a while in belief b and observes o as a result. It can then update its belief using Bayes rule:

$$b_n(s) = \eta\Omega(o|s',a)\sum_{s\in S} T(s'|s,a)b_{n-1}(s) \tag{1}$$

where η is the normalizing constant $\Omega(o|b,a)$. The agent's goal is to find a policy mapping the set of beliefs B to actions A to maximize the expected reward $E[\sum_n \gamma^n R(s_n,a_n)]$.

We use a value function to represent our policy. Let the value function $V_\pi(b)$ represent the expected reward if we start with belief b and act according to policy π. The optimal value function is known to be piecewise-linear and convex [11], so we represent V with the vectors $\Gamma = \{\alpha_i\}$; $V(b) = \max_i(\alpha_i \cdot b)$. The optimal value function is unique and satisfies the Bellman equation:

$$\begin{aligned} V(b) &= \max_{a\in A} Q(b,a), \\ Q(b,a) &= R(b,a) + \gamma \sum_{b'\in B} T(b'|b,a)V(b'), \\ &= R(b,a) + \gamma \sum_{o\in O} \Omega(o|b,a)V(b_a^o), \end{aligned} \tag{2}$$

where $Q(b,a)$ represents the expected reward for starting in belief b, performing action a, and then acting optimally. The last line follows if we note that there are only $|O|$ beliefs that we can transition to after taking action a in belief b (one for to each observation). The belief b_a^o is the belief that results from taking action a and observing o from belief b (equation 1); $\Omega(o|b,a)$ is the probability of observation o after taking action a in belief b ($\sum_{s\in S}\Omega(o|s,a)b(s)$).

The Bellman equation may be solved iteratively. Suppose that we have a representation of the value function as a collection of vectors $\Gamma_n = \{\alpha_i\}$. First, we generate intermediate sets $\Gamma^{a,*}$ and $\Gamma^{a,o}$ for all action, observation pairs:

$$\Gamma^{a,*} = \{\alpha | \alpha(s) = R(s,a)\} \tag{3}$$

$$\Gamma^{a,o} = \{\alpha | \alpha(s) = \gamma \sum_{s'\in S} T(s'|s,a)\Omega(o|s',a)\alpha'(s')\},\ \forall \alpha' \in \Gamma_n \tag{4}$$

Next, we generate the set of Q-functions Γ^a which is a cross sum that includes a vector from $\Gamma^{a,*}$ a vector from $\Gamma^{a,o}$ for each observation o:

$$\Gamma^a = \Gamma^{a,*} \oplus \Gamma^{a,o_1} \oplus \Gamma^{a,o_2} \ldots \tag{5}$$

Finally, the new value function is the union of all the Q-functions:

$$\Gamma_{n+1} = \bigcup_{a\in A} \Gamma^a \tag{6}$$

Each iteration, or *backup*, brings the value function closer to its optimal value [4]. Once the value function has been computed, it is used to choose actions. After each observation, we update the belief using equation 1 and then choose the next action using $\arg\max_{a\in A} Q(b,a)$ with $Q(b,a)$ given in equation 2.

The cross sum in the exact solution can cause the number of α-vectors to grow exponentially in each backup iteration. The optimal solution may consist of an infinite number of α-vectors, so even if we prune away α-vectors completely dominated by other α-vectors, we cannot prevent the computation of an ever-growing number of cross sums as the number of backups increases.

Several algorithms sidestep the issue of an exponentially increasing number of α-vectors by backing up the value function only at select beliefs [6, 12, 10, 9]. Each belief can have only one associated α-vector, so the the size of the value function is capped by the number of beliefs that we wish to consider. We will describe our approach in the context of PBVI [6] because it is one of the simplest point-based POMDP approximations; however, we stress that our approach—which can be thought of as exponentially reducing the size of the belief space—can be applied to any point-based value function approximation technique.

When using a point-based approach, the $\Gamma^{a,*}$ and $\Gamma^{a,o}$ sets are computed as before, but we only compute the Q-function for each belief in our belief set:

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{o\in O} \arg\max_{\alpha\in\Gamma^{a,o}} (\alpha \cdot b) \tag{7}$$

Finally, we retain only the members of the Γ_b^a set that are maximal for some belief in our set:

$$\Gamma_{n+1} = \arg\max_{\Gamma_b^a, \forall a\in A} (\Gamma_b^a \cdot b), \forall b \in B. \tag{8}$$

For later comparison, Table 1 summarizes the point-based value iteration algorithm [6]. The quality of point-based approximations often depends on whether certain supporting beliefs are present in the belief set or depend on how densely we sample beliefs from the space of reachable beliefs. In some problems, we may be able to represent near-optimal policies with only a few supporting belief points. However, if we need a high quality approximation, or if the problem simply has a large number of states, actions, and observations, we may require a large number of belief points to find an adequate approximation. In these situations, even a point-based approximation may be computationally intractable.

Table 1: Point-Based Value Iteration.

1. Sample a set of initial beliefs.
2. Begin point-based value iteration loop:
   - Compute $\Gamma^{a,*}$ and $\Gamma^{a,o}$ sets (equation 4).
   - Compute Γ_b^a sets (equation 7).
   - Compute new value function (equation 8).
   - Add beliefs to the belief set (based on various heuristics or information criteria).

3. PREFERENCE ELICITATION POMDPS

In the context of preference elicitation—which we interpret broadly to mean identifying a user's preference, intent, or command—POMDPs tend to have a very specific structure. The hidden state often represents some property, such as a desired destination or product, that does not change during the course of the interaction. The agent's goal is to discover this hidden state and submit it to another system that will satisfy the user's desire ([13], [3]).

In many cases, it may be reasonable to assume that the rewards associated with the agent's actions depend only on whether the agent acts correctly with respect to the user's desire and not on the particular desire itself. For example, if the user wants the agent to start to play some media, it may not matter if the true task is to start the TV or play a CD; the user will be happy as long as the agent starts the correct system. Likewise, it may be reasonable to assume that the user will be just as upset if the agent starts the TV instead of the CD as if the agent plays a CD instead of the TV.

Similarly, certain symmetries may also exist in the observational model. We assume that there exists some system that converts user input—be it text, voice, or visual—into a prediction of what the user desires and a confidence score. A reasonable model for such a recognition system is that the agent is most likely to receive an output that matches the user's true desire: for example, if the user wants to hear some music, then the recognition system is most likely to output an indicator for "CD" We can simplify further and assume that neither (1) hearing any other goal by mistake nor (2) the quality of the recognition depends on the goal state.

Next, we will show that if the preference elicitation POMDP has, or can be reasonably approximated to have, the qualities described above, then its solution can be computed with exponentially fewer supporting belief points without any loss in the quality of the point-based value approximation.

4. THE PERMUTABLE POMDP

The core concept exploited by the Permutable POMDP is that preference elicitation states are often symmetric with regard to the transition, reward, and observation functions, and therefore interchangeable with regard to the policy. This symmetry implies that the POMDP model is "flat": the correct type of action depends not on the particular states in question, but only on the distribution over the states. For example, if the dialog manager's goal is to determine what media to play, then the optimal policy might be to ask for clarification if unsure—regardless of what media is in question—but submit the most likely location to the media player if that request is sufficiently more likely than any other. This flatness assumption is clearly a simplification, since in reality some requests may be harder to recognize than others, and some pairs of requests may be more likely to be confused with each other than other pairs. The user may also be more forgiving of mistakes in some situations than others. However, since the optimal policy is often fairly robust to small model variations, the symmetric model may be a reasonable approximation in many real-world situations.

The key insight of our algorithm is that in a permutable POMDP, if some vector α is part of the value function, then all permutations of α must also be part of the value function. This insight follows from the observation that only the distribution over states—and not the particular state—matters when choosing an action. If a particular α-vector optimizes the value function for a particular belief, then a permutation of that α-vector will optimize the value function for a permutation of that belief. We will therefore realize considerable computational savings by representing (and backing up) only one permutation of each α-vector.

To define the notion of a "permutation", recall that in order to express the value function and beliefs as vectors, an explicit ordering on the state space S always required: that is, some state is the first field in the vector, some other state is the second field in the vector, etc. Given an explicit ordering on the state space S, we define a permutation operator π, $\pi : s \rightarrow s'$, that maps one ordering of states to another. This permutation may be applied to the order of probabilities in a belief $b(s) \rightarrow b_\pi = b(\pi(s))$ or the values in an α-vector $\alpha(s) \rightarrow \alpha_\pi = \alpha(\pi(s))$.

In general, we cannot blindly re-order states and expect the policy to be unchanged. However, if the reward, action and observation spaces have certain properties, then the policy is in fact invariant to permutations, that is, $V(b(s)) = V(b(\pi(s)))\forall\pi, b$. For example, suppose that the action a "Start TV" has reward r if the user's desire is s_1, watch TV. If we now re-order the states such that s_1' corresponds to hear music, then we require there be some other action a' (in this case, "Start CD") with the same reward r. Similarly, if the observation "thank-you" is most likely to be seen given (s_1, a), then it should also be the most likely observation given (s_1' , a').

Intuitively, we can see that the preference elicitation model from section 3 has the necessary symmetries; we now formally define a broader set of sufficient conditions for the permutable POMDP. First, let $\pi_{\pi_s,a}(o)$ to be a permutation on the observations, parameterized by an action a and a state permutation π_s.

THEOREM 1. *If, for every state permutation $\pi_s(s)$ and action a, there exists an action a_{π_s} and observation permutation $\pi_{\pi_s,a}(o)$ such that*

- $R(s,a) = R(\pi_s(s), a_{\pi_s})$
- $\Omega(o|s,a) = \Omega(\pi_{\pi_s,a}(o)|\pi_s(s), a_{\pi_s})$
- $T(s'|s,a) = T(\pi_s(s')|\pi_s(s), a_{\pi_s})$,

then for every vector α in the value function, all permutations of α are also part of the value function.[2]

PROOF. Recall that for the exact POMDP solution, the value function is the union over actions of the α vectors in Γ^a. Given a

[2]We note that while similar, the requirements for the action classes developed here are more restrictive than in first-order MDPs[1].

set of vectors Γ_{i-1}, each vector in the new set Γ_i^a is equal to

$$\alpha_i(s) = R(s,a) + \gamma T(\cdot|s,a)[\sum_{o \in O} \Omega(o|s,a)\alpha_{i-1,o}] \quad (9)$$

for some set of vectors α_{i-1} from Γ_{i-1}. (Indeed, Γ_i^a consists of all vectors α_i that can be produced by various combinations of vectors from the value function, given action a.)

If we apply our sufficient conditions above to equation 9, it follows that

$$\begin{aligned} \alpha_i(\pi_s(s)) &= R(\pi_s(s), a_{\pi_s}) + \gamma T(\cdot|\pi_s(s), a_{\pi_s}) \cdot \quad (10) \\ &\quad [\sum_{o \in O} \Omega(\pi_{\pi_s,a}(o_1)|\pi_s(s), a_{\pi_s})\alpha_{i-1,o}] \end{aligned}$$

for a given state permutation $\pi_s(s)$. Since our conditions must be true for every state permutation, it follows that if α is part of the value function, then all permutations of α are also part of the value function. □

We can now formally demonstrate how the preference elicitation POMDP from section 3 satisfies the considtions above. First, consider a single "general query" action, a_q, e.g., "How can I help you?". In this case, let $a_q = a_{\pi_s}$ for all π. The cost of asking a general query usually does not depend on the user's desire, so $R(s,a_q) = R(\pi_s(s),a_q) = R_{ask}$ satisfies the reward condition of Theorem 1. Furthermore, let there be one observation o_g that uniquely describes each task (i.e., "Turn on the TV."). If we permute the state, we can also permute o_g such that $\pi_{\pi_s,a} = \pi_s$. We satisfy the observation condition by letting $\Omega(o_g|s,a)$ equal p_1 if $o_g = s$ and p_2 if $o_g \neq s$, $p_1 > p_2$. Finally, we can satisfy the transition condition by setting $T(s'|s,a) = \delta(s'-s)$, where $\delta()$ is the Dirac delta function. These conditions essentially say that general queries does not change the user's desire are most likely to result in an input that reflects the desired task.

Secondly, let us consider a "submit" action, in which the dialog system sends a task to the media system. Let the action to submit tune s to the system be $a_g(s)$. We assume that there is one submit action for each task. Then we can let $R(s,a)$ equal r_1 if $a = a_g(s)$, that is, if the agent submits the correct tune, and r_2 otherwise; $r_1 > r_2$. For the observation condition, let there be two observations, o^+ and o^- that represent positive (the agent submitted the action correctly) and negative (the agent submitted an incorrect action) feedback from the user. If $a = a_g(s)$, then we expect to get some positive feedback: $\Omega(o|s,a(s)) = p_3$ if $o = o^+$ and p_4 otherwise, $p_3 > p_4$. If $a \neq a_g(s)$, we expect to get some negative feedback: $\Omega(o|s,a(s)) = p_5$ if $o = o^-$ and p_6 otherwise, $p_5 > p_6$. To satisfy the transition condition, let $T(s'|s,a)$ be uniform if $a = a_g(s)$ and the identity otherwise. Finally, for some state permutation π_s, set $a_{\pi_s} = \pi_s(a)$ and $\pi_{\pi_s,a}(o) = o$ and note that these settings will ensure the symmetry conditions are met.

We are not restricted to these two classes of actions above. For example, many dialog management systems will have a "confirm" type of action, in which the system will ask a question of the form "Did you want to play a CD?" These actions will have a similar form to the "submit" action. More complex actions, such as those that attempt to disambiguate frequently confused terms (e.g., "Did you want watch TV or play a CD?") are also possible as long as every action a has, for every permutation π, a corresponding action a_π satisfying the reward, transition and observation conditions.

4.1 Solving the Permutable POMDP

We now show how to efficiently solve a permutable POMDP. Recall that point-based POMDP solvers generally have two parts: belief set selection and value iteration. We modify the first part with an additional step to reduce the sampled beliefs to a set of representative beliefs. Since any permutation is valid, without loss of generality, let us require the representative belief b to have values sorted in descending order. We then slightly modify value iteration to ensure that the updated step will behave as if we had the full set of beliefs represented by our small set.

To generate the belief set, either initially or online, one should use any belief sampling or expansion technique [6, 12, 10, 9], sort all the beliefs in the belief set in descending order, and remove similar beliefs (we used an L1 metric, but again, one should choose a similarity measure appropriate for the problem).[3] Regardless of how the beliefs are chosen, the point is that we only need consider one canonical ordering of the values in the belief. Once we have the representative belief set, we are ready to compute the value function. We show below how to adapt the PBVI [6] algorithm to our approach; however, the steps are nearly equivalent for any other point-based approximation.

Recall that the first step in point-based value iteration is computing the $\Gamma^{a,*}$ and $\Gamma^{a,o}$ sets as described in equation 4:

$$\Gamma^{a,*} = \{\alpha | \alpha(s) = R(s,a)\} \quad (11)$$

$$\Gamma^{a,o} = \{\alpha | \alpha(s) = \gamma \sum_{s' \in S} T(s'|s,a)\Omega(o|s',a)\alpha'(s')\}, \forall \alpha' \in \Gamma_n \quad (12)$$

These equations depend only on the previous α-vectors and are unchanged if we only use our sorted set of belief points. We note that our initial set of α-vectors Γ_n only contains vectors corresponding to the sorted beliefs in our belief set. The number of α-vectors is bounded by the number of supporting beliefs, so the sizes of $\Gamma^{a,*}$ and $\Gamma^{a,o}$ will be relatively small and fast to compute.

The next step in the standard point-based value iteration algorithm combines the $\Gamma^{a,*}$ and $\Gamma^{a,o}$ sets into a Γ_b^a set for each belief:

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{o \in O} \arg \max_{\alpha \in \Gamma^{a,o}} (\alpha \cdot b) \quad (13)$$

We must now recall that if a specific α-vector is explicitly in Γ_n, all permutations of α are also in Γ_n. However, the vectors in our $\Gamma^{a,*}$ and $\Gamma^{a,o}$ sets were created with only the α-vectors corresponding to our sorted beliefs. Thus, they represent only a small fraction of the vectors that should be present in the $\Gamma^{a,*}$ and $\Gamma^{a,o}$ sets if we had a full representation of Γ_n with all permutations of α-vectors explicitly represented. We must consider all permutations, implicit and explicit, when choosing the best α-vectors for each belief.

Since the members of the $\Gamma^{a,o}$ set are made of linear combinations of α-vectors in Γ_n, we can apply the observation and transition conditions from Section 4 to argue that if a vector α is explicitly represented in $\Gamma^{a,o}$, then we could have constructed all permutations of α by using other (not explicitly represented) permutations of vectors in Γ_n. Similarly, from the reward conditions, if a vector r is in the $\Gamma^{a,*}$ set, all permutations of r will also be in the set. Thus, equation 13 should read:

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{o \in O} \arg \max_{\alpha \in perm(\Gamma^{a,o})} (\alpha \cdot b), \quad (14)$$

where $perm(\Gamma^{a,o})$ is the set containing all permutations of the vectors explicitly represented in $\Gamma^{a,o}$.

Equation 14 may at first seem disheartening, because although we have a very small number of supporting belief points, the set $perm(\Gamma^{a,o})$ is exponentially larger than our small $\Gamma^{a,o}$ set. However, recall that we are seeking vectors to maximize the dot product

[3] Sampling and removing similar beliefs was the slowest part of our approach; depending on the designer's knowledge of the problem, much time could be saved by "seeding" the initial belief set with beliefs that we know will be important, reducing the number of trajectories required to ensure good coverage of the belief space.

Table 2: Point-Based Value Iteration for Permuted POMDPs.

1. Sample a set of beliefs.
2. **Sort beliefs in descending order and remove nearby beliefs.**
3. Begin point-based value iteration loop:
   - Compute $\Gamma^{a,*}$ and $\Gamma^{a,o}$ sets (equation 4).
   - **Sort vectors in $\Gamma^{a,o}$ sets in descending order.**
   - Compute Γ_b^a sets (equation 15).
   - Compute new value function (equation 8).

Table 3: Action Selection for Permuted POMDPs.

1. Sort current belief b to b_s; let π_s be the permutation that takes $b_s \rightarrow b$.
2. Determine the optimal action a_s for b_s using the α-vectors explicitly in Γ_n.
3. Perform the action a_p that corresponds to permutation π_s and action a_s.

$\alpha \cdot b$, and our sorted beliefs are in descending order. If α is a member of $\Gamma^{a,o}$, the maximally-rewarding permutation of α is the permutation that sorts in the values of α in descending order. Thus, we do not need to consider all the permutations of α, only the "best" permutation that has its values in descending order. Our equation to compute Γ_b^a becomes:

$$\Gamma_b^a = \Gamma^{a,*} + \sum_{o \in O} \arg \max_{\alpha \in sort(\Gamma^{a,o})} (\alpha \cdot b), \qquad (15)$$

where $sort(\Gamma^{a,o})$ is a set of the same (small) size as $\Gamma^{a,o}$ in which each α-vector has been sorted to have its values in descending order. Sorting the vectors in the $\Gamma^{a,o}$ set can be done efficiently with available numerical tools.

The final step in the standard point based value iteration algorithm involves retaining only the members of the Γ_b^a set that are maximal for some belief in our set:

$$\Gamma_{n+1} = \arg \max_{\Gamma_b^a, \forall a \in A} (\Gamma_b^a \cdot b), \forall b \in sort(B). \qquad (16)$$

We produced the best possible vector in the Γ_b^a set by using the sorted α-vectors from the $\Gamma^{a,o}$ sets. Thus no more changes are required to the standard algorithm to compute the new value function.

Table 2 summarizes how one should apply point-based value iteration to these special POMDPS; those steps specific to our solution technique are highlighted in bold. The algorithm requires only small changes to an already implemented point-based value iteration scheme, yet the computational benefits are substantial since exponentially fewer belief points are needed.

4.2 Using the Permutable POMDP solution

Given a normal value function V of vectors Γ and a current belief b, the standard way to determine the next action to take is to find the α-vector from Γ that maximizes the dot product $\alpha \cdot b$. From equation 16, each vector has the action associated with the Γ_b^a from which it came; this action is the best action for the agent to take.

If we use only sorted beliefs to build our value function, then our value function will explicitly only contain sorted α-vectors (since the sorted permutation of the α-vectors will be the one that maximizes the dot product $\alpha \cdot b$). Given some arbitrary belief, we cannot simply multiply our current belief with the α vectors explicitly in our value function—we must identify which permutation of the α vectors maximizes the expected reward of the belief b. To do so, we permute the belief into the representative belief set, identify the best α-vector (and corresponding action equivalence class) and then reverse the permutation to identify the true action.

To efficiently choose an action, we first sort our current belief and determine which action would have been appropriate to the sorted version of the belief. Formally, the sort implies that we applied a permutation to sort the current belief in descending order. Let action a_s be the action with the highest expected value for the sorted belief, and let π_s be the permutation that takes the *sorted* belief to our current belief (note that sorting the current belief is the reverse of π_s). Recall from our symmetry conditions, for each state permutation π_s and action a_s, there existed some permuted action a_p for which the conditions held. Now we have a π_s and an a_s for our sorted belief; the correct action to take in our current belief is the corresponding a_p. Table 3 summarizes this procedure.

While the action selection step may at first appear complicated, its application in the simple preference elicitation context is quite intuitive. We use the action associated with the sorted belief to determine what type of action to perform (a general query, a confirmation, or a submission). If the action is of a confirm or submit type, we use the distribution of the current (unsorted) belief to determine what state should be confirmed or submitted. For example, suppose there are two possible tasks, "play CD" and "play TV," and the current belief is (.1, .9). If the value function states for the sorted belief (.9 , .1), the correct action is to confirm that the user wants the music turned on, then we take the action type—confirm—and attach it to the most likely state in our actual belief—the TV—to determine that the correct action to take on our belief is to confirm if the user wants to watch TV.

4.3 Extensions to more complex models

We describe extensions to more complex models.

Filling Multiple Slots.

In many dialog management scenarios, the goal of the system may be to fill a number of "slots" in a knowledge base. For example, if an agent is managing a user's personal calendar, it may need to discover the intended date and location for a meeting. Slot filling dialogs are also common in automated booking systems [13], in which the agent must determine the caller's origin, desired destination, and travel date. In these cases, the agent usually proceeds by filling one slot at a time; in the personal calendar example, the agent might first determine the date of the user's meeting and then the location. This approach can be shown to be optimal if the agent's actions provide information about only one slot. While not often true—the user might also mention the meeting location when asked for the meeting time—it can be a useful approximation.

The state space in a slot-filling dialog is usually expressed by a vector of factors, such as $\vec{s} = \{s_d, s_l\}$, where s_d might represent a meeting date and s_l represents a meeting location. If the slots are independent—that is, each action and observation only affects the belief about one slot—and the symmetry conditions hold for each slot, we can get significant computational gains by noting that the

belief may be expressed as

$$b = kron(b_d, b_l) \quad (17)$$

where $kron$ is the Kronecker tensor product and b_d and b_l can be updated independently. We compute permuted solutions for each slot separately. If all the slots' actions are "submit," then the agent should submit the information in its slots. Otherwise, it should perform a non-submit action from any slots (to not annoy the user, it makes sense to fill one slot before continuing to the next one).

Partially Permutable POMDPs.

In other scenarios, there may be parts of the model that cannot be completely expressed in the symmetric form we described. For example, suppose that there exist two different observation error rates, depending on whether the agent is in a noisy or quiet area. We can still express the state as a vector of state features $\vec{s} = \{s_g, s_n\}$, where s_g is the user's goal state and s_n is the noise state. Here, we may not be able to treat the noise and goal state independently: the value of the noise directly affects our state update, and depending on its value, the optimal policy may require greater or fewer confirmation questions before submitting a state. Instead, we note that for a particular value of the noise, the ordering of the goal states does not matter (since we posited that the user model satisfies the symmetry conditions). Thus, our value function will be symmetric in blocks. Let there be k noise states and m goal states, and we write the belief in the following form:

$$b = [p_{n1,g1}, p_{n1,g2}, ...p_{n1,gm}, p_{n2,g1}, ...p_{nk,gm}]. \quad (18)$$

Let the vector

$$\alpha = [a_{n1,g1}, a_{n1,g2}, ...a_{n1,gm}, a_{n2,g1}, ...a_{nk,gm}] \quad (19)$$

be part of the value function; then all vectors

$$\begin{aligned} \alpha_p &= [perm(a_{n1,g1}...a_{n1,gm}), perm(a_{n2,g1}...a_{n2,gm}), ... \\ &\quad , perm(a_{nk,g1}, ...a_{nk,gm})] \end{aligned} \quad (20)$$

will also be part of the value function. Such a value function can be solved for by considering beliefs that have been sorted by blocks:

$$\begin{aligned} b_s &= [sort(p_{n1,g1}...p_{n1,gm}), sort(p_{n2,g1}...p_{n2,gm}), ... \\ &\quad , sort(p_{nk,g1}, ...p_{nk,gm})] \end{aligned} \quad (21)$$

and the only change required in the standard point-based value iteration algorithm is to sort the $\Gamma^{a,o}$ sets also by blocks. Applying this approach to the scenario of trying to simultaneously learn a user's preference (reward) model and goal, we found we could find reasonable approximate solutions in situations where using the standard algorithm was computationally intractable.

Approximately Permutable POMDPs.

Finally, in some cases, the true model may not be symmetric. Since similar models have similar policy returns ([5], Lemma 2), approximating the true model by a symmetric model may be reasonable in certain situations. In other cases, depending type of point-based value iteration used, one can speed up computation time by first computing the solution as symmetric model and using the resulting vectors to initialize value iteration for the true model (the solution will converge to the solution for the true model with sufficient backups [4]).

5. RESULTS

We present simulation results on an abstract preference elicitation POMDP to demonstrate the computational savings from our approach. In this POMDP, the state space consisted of n possible user goal states $\{s_1, ..., s_n\}$. The observation space consisted of n observations $\{o_1, ..., o_n\}$ associated with each of the n states as well as observations o^+ and o^- for positive and negative confirmations, respectively. Finally, the agent could choose from three types of actions. A general query asked the user to state his goal. A confirmation question confirmed a specific goal with the user. Lastly, the agent could choose to submit a particular goal state.

For general queries and confirmation questions, the transition model was mostly static: with probability 0.99, the user's goal state did not change. With probability 0.01, the user changed his goal to another state chosen uniformly. If the agent submitted the correct goal state, the goal state was reset uniformly. The agent received a small negative reward for making various queries, with higher penalties for confirming an incorrect goal state and lower penalties for confirming the true goal state. It received a large negative reward for submitting an incorrect goal and a large positive reward for submitting the correct goal (table 4).

Table 4 also lists the key parameters of the observation model. If the agent made a general query when the user's goal state was s_i, then it observed the associated observation o_i with probability 0.5; otherwise it received a noisy observation uniformly at random from the remaining observations. For confirmation questions, if the agent confirmed the true user goal state, it received a positive response o^+ with probability 0.8 and an arbitrary response with probability 0.2. Similarly, if the agent confirmed an incorrect state, then it received a negative response o^- with probability 0.8 and an arbitrary response with probability 0.2.

Table 4: Parameters for Preference Elicitation POMDP.

Pr[hear correct state] from a general query	0.50
Pr[hear correct confirmation] from a confirmation	0.80
Reward for a general query	-2
Reward for a correct confirmation	- 1
Reward for an incorrect confirmation	-5
Reward for a correct submission	100
Reward for an incorrect submission	-200

In our experiments, we varied the number of goal states from 5 to 50 and measured the empirical simulation performance, the total computation time, and the total number of α-vectors for our technique as well as an optimized version of PBVI [6]. Initially, the agent believed that all possible goal states were equally likely to be the user's true goal state. Each simulated trial was run until the agent submitted the true user goal state; then the user's goal state was resampled and the agent's belief reset. Both implementations were run in Matlab on a 1.6GHz computer with 2GB RAM.

Variable Number of Belief Vectors.

In the first set of experiments, we let the number of sampled belief points grow linearly with the size of state space; the number of beliefs was equal to fifty times the number of hidden states. To be strictly fair, this approach provided the Permutable POMDP with an advantage in that it could represent an exponentially growing set of beliefs for each linear growth in the standard PBVI implementation. However, including an exponentially growing belief set would have been computationally intractable, even for the relatively small numbers of states in question. To speed up convergence (each approach performed 10 backups), we included "corner beliefs," that is, beliefs corresponding to each user goal, in the initial belief set. Since all corner beliefs are permutations of each other, we note that adding corner beliefs added $|S|$ beliefs to the standard belief set

and one belief to the permuted belief set.

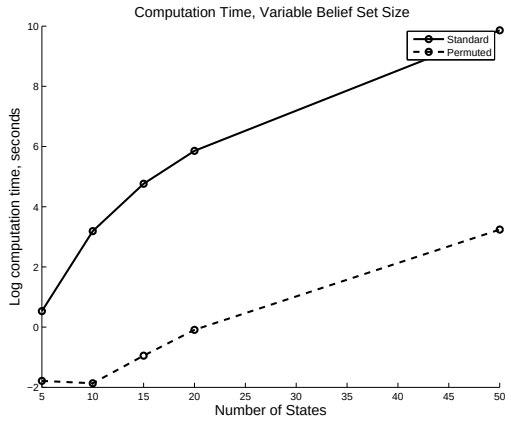

Figure 1: Computation time for solutions with beliefs growing linearly with the number of states. Note the (natural) log scale in the time axis of the figure.

Figure 1 shows that computation time required compute the POMDP solution as the state space grew. Note the (natural) log scale on the time axis of the plot—the Permutable POMDP was orders of magnitude faster than standard PBVI. The computation time for our approach included the time required to sort and remove nearby beliefs from the belief set used for the permuted solution.[4]

Figure 2 showed that the time required to remove nearby beliefs almost doubled the total time required to compute the permuted solution. We used a naïve algorithm that computed the distance between all pairs of beliefs to remove near-duplicates; a more sophisticated algorithm would additionally speed up our approach.

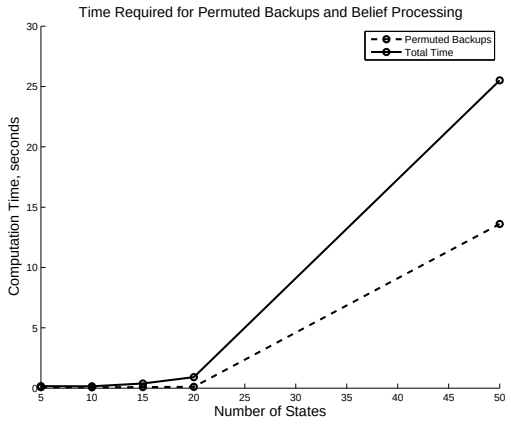

Figure 2: Total computation time for the permuted solution and the permuted backups. Processing the beliefs almost doubles computation time.

Figure 3 shows the median reward from 500 trials with upper and lower quartiles. Although there were small fluctuations in the median performance, PBVI and the Permutable POMDP were within each other's region of variation.[5] Even though the permuted solution required much less computation, it had essentially the same performance as the standard solution. Also, both approaches maintained their level of performance as the number of states increased (we note that this level is near-optimal).

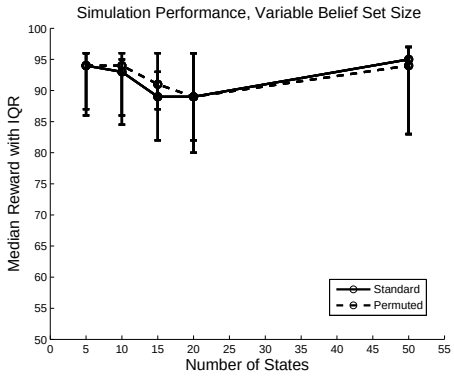

Figure 3: Performance of solutions with beliefs growing linearly with the number of states.

Finally, figure 4 shows the number of α vectors in each solution. Note that the permuted solution required many fewer vectors than the standard solution, and did not grow appreciably as the number of states increased. This effect fits our intuition that in this goal-discovery task, the complexity of the task should not increase greatly as additional states are added: the optimal policy essentially needs to determine when to make a general query, when to confirm the most likely state, and when to submit the most likely state. These thresholds depend largely on the reward values; however, the standard approach was blind to the problem symmetry and thus required beliefs for each option to determine an appropriate policy.

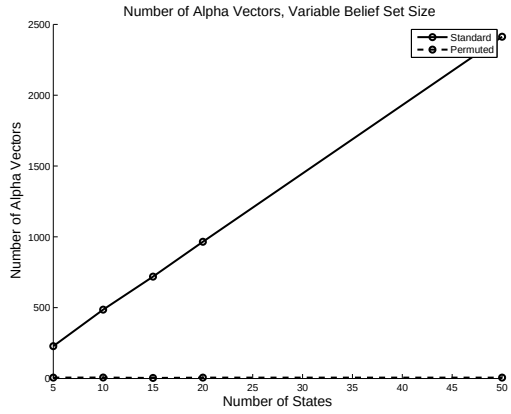

Figure 4: Number of α-vectors in solutions with beliefs growing linearly with the number of states.

Fixed Number of Belief Vectors.

In the second set of experiments, we capped the number of beliefs to 100 points, regardless of the size of the state space, and computed 15 backups for each technique. Figure 5 shows the amount of computation time spent on the solution as the size of the state

[4]If we did not remove nearby beliefs, our sorted belief set would be the same size as the original belief set, and we would see no gains in computational time. However, we would see a much more accurate policy, since computing the permuted POMDP solution on n beliefs is equivalent to computing the full solution on $n|S|!$ beliefs). Simulations results demonstrating this effect were omitted for lack of space.

[5]The solution techniques optimized the mean reward, as is standard in POMDPs, but the median and the inter-quartile ranges are shown to more accurately reflect the asymmetric spread in the data.

space increased. Since the number of beliefs was fixed, the increase in time came only from handling larger matrices as the state space size increased. For larger state spaces, having fewer operations with large transition and observation matrices led to a significant reduction in the computation time for the Permutable POMDP, yet another practical reason for using our approach.

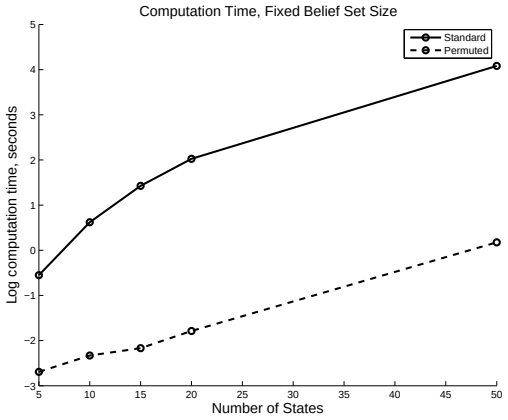

Figure 5: Computation time for solutions with 100 belief cap.

Figure 6 compares the performance of the resulting policies. As expected, the performance of the PBVI policy declined as the number of states increased. The standard approach suffered from the belief cap because as the number of states increased, the standard solver required beliefs that reflected confusion between all possible states. The high variance in performance reflects the fact that insufficient belief sampling led to policies that were effective in some parts of the belief space and not others.

Our approach was shown to be more robust to the limited number of beliefs since the actual policies were relatively simple and the small belief set that we used actually represented an exponential number of beliefs. The performance of the permuted solution remained near-optimal for all the state space sizes. We note that the permuted approach required only 11 supporting belief points, even with 100 possible goal states to achieve this policy performance.

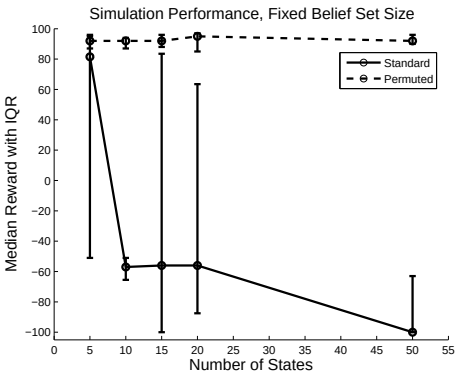

Figure 6: Performance of solutions with 100 belief cap.

6. DISCUSSION AND CONCLUSION

We presented the Permutable POMDP, an approach for quickly and accurately solving POMDPs that commonly arise in preference elicitation frameworks. In some ways, our approach is similar to that of Williams and Young [13], which creates a summary POMDP that determines what type of action to take based on the probability of the most likely state. In [13], the action type is combined with the most likely state to determine the complete action; the dynamics of the summary POMDP must be determined through sample trajectories in the larger POMDP. Our approach differs from the summary POMDP method in that we do not approximate the true POMDP with a smaller POMDP; we show that if a POMDP has a particular structure, it can be solved efficiently directly. Although we presented the permutable POMDP in a preference elicitation context, we also note that it may apply in many settings where an agent must identify a particular target.

Our approach is easily incorporated into most point-based POMDP solvers: the solver will then require exponentially fewer belief points for a given desired solution quality. In empirical tests, we showed that we can speed up computations by several orders of magnitude, allowing us to consider preference elicitation POMDPs for state spaces that might otherwise be too large for POMDP techniques. The technique requires specific symmetries in the POMDP structure, but in many preference elicitation problems, it may be reasonable to approximate the model as having such a structure. Extensions would include developing principled approximation procedures based on the permutable POMDP, as well as finding more compact encoding schemes for models that satisfy the permutable POMDP constraints.

7. REFERENCES

[1] C. Boutilier. Planning and programming with first-order markov decision processes: insights and challenges. Morgan Kaufmann, 2001.

[2] C. Boutilier. A pomdp formulation of preference elicitation problems. *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.

[3] F. Doshi and N. Roy. Efficient model learning for dialog management. In *Proceedings of Human-Robot Interaction (HRI 2007)*, Washington, DC, March 2007.

[4] G. J. Gordon. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*, San Francisco, CA, 1995. Morgan Kaufmann.

[5] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *ICML*, 1998.

[6] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. *IJCAI*, 2003.

[7] K. Regan, R. Cohen, and P. Poupart. The advisor-pomdp: A principled approach to trust through reputation in electronic markets. *Conference on Privacy Security and Trust*, 2005.

[8] N. Roy, J. Pineau, and S. Thrun. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the ACL*, Hong Kong, 2000.

[9] G. Shani, R. Brafman, and S. Shimony. Forward search value iteration for pomdps. *IJCAI*, 2007.

[10] T. Smith and R. Simmons. Heuristic search value iteration for pomdps. In *Proc. of UAI 2004*, Banff, Alberta, 2004.

[11] E. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University, Stanford, California, 1971.

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

[13] J. Williams and S. Young. Scaling up pomdps for dialogue management: The "summary pomdp" method. In *Proceedings of the IEEE ASRU Workshop*, 2005.