The Permutable POMDP: Fast Solutions to POMDPs for Preference Elicitation

Finale Doshi, Nicholas Roy
Massachusetts Institute of Technology

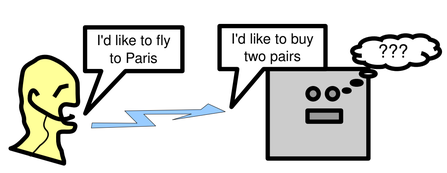
Abstract

The ability for an agent to reason under uncertainty is crucial for many planning applications, since the 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 taking advantage of structure in the problem domain. We show that the structure of many preference elicitation problems—in with 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 full quality of the solution.

Motivation

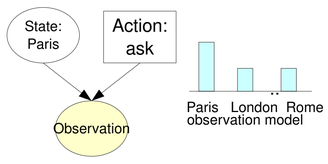
In many real world scenarios, agents must make decisions based on noisy measurements of the environment. In these domains, it is critical for the agents to reason under uncertainty. Let's consider a toy example: suppose an automated telephone agent is trying to help book a flight. A successful agent must manage uncertainty in this dialog to book the correct location.



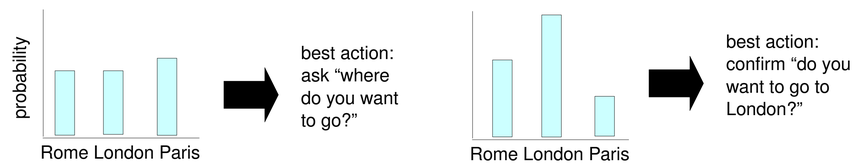
The POMDP Model

Model: Partially Observable Markov Decision Processes (POMDPs) provide a formal way to manage uncertainty in the dialog. A POMDP consists of:

-States (hidden!): the user's wants
- Observations: what the agent hears
- Actions: queries the agents can ask
- Reward Model R(s,a)
- Transition Model T(s'|s,a)
- Observation Model O(o|s,a)

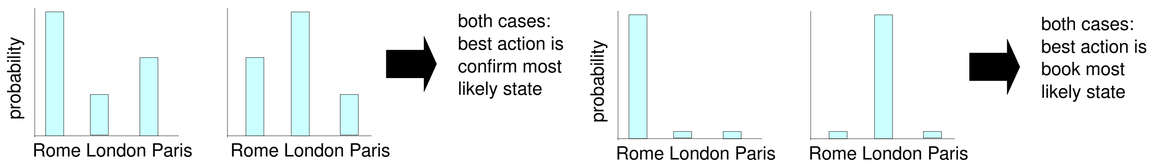


The POMDP will optimally trade between actions that gather information and actions that will increase reward. It does so by tracking the belief, a probability distribution over states. By choosing actions based on this belief, it can considers both where the user wishes to fly and the agent's uncertainty about the location:



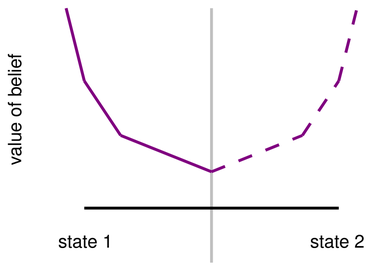
Key Insight

Note in our simple dialog scenario case (with a few assumptions), the correct type of action really only depends on the "shape" of the belief, not the belief itself.



In fact, many problems have a similar structure. Examples include determining a user's desired flight number, determining the appointment time and location in an automated calendar, determining a desired location in a robotic wheelchair, or determining reputations in electronic markets. Even non-dialog based problems may have this structure—for example, consider a system that is trying to do an optimal series of experiments to determine the identity of a particular mineral. All these problems share the property there is a fixed, hidden state that must be determined via a series of information gathering actions that fall into classes having similar but state-dependent effects.

Why is this useful? Suppose that we have a POMDP in which only the shape of the belief matters, that is, the value function V(b) = V(b') for all b' that are permutations of b. In this case, **when solving the POMDP, we can consider only a single canonical permutation of the belief, exponentially decreasing the belief space we need to consider.** For example, suppose we had a system with only two states. If we could solve for the value function in cases where Pr(state1) > Pr(state 2), we can simply mirror the part of the value function we have solved to the other half of the state space. (In high dimensions, the effect is much more dramatic.)



Solving the POMDP

The optimal policy in a POMDP is one that maximizes the expected discounted reward:

$$max\, E_{s_t}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$$

It can be shown that this policy satisfies the Bellman optimality equations below, where V(b) is the value of being in belief b (and acting optimally) and Q(b,a) is the value of taking action a in belief b and then acting optimally:

$$V_n(b) = max_a Q_n(b,a)$$

$$Q_n(b,a) = R(b,a) + \gamma \sum_{b' \in B} T(b'|b,a) V_{n-1}(b')$$

$$Q_n(b,a) = R(b,a) + \gamma \sum_{o \in O} O(o|b,a) V_{n-1}(b_a^o)$$

The solution to the POMDP value function is piecewise linear, and is often modelled as the maximum of a set of vectors $\{\alpha\}$. In general, we have to ensure that the equation above is satisfied for all beliefs in the belief space; this can be done by computing the following series of operations (known as backups). We start with an initial approximation to the value function $\Gamma = \{\alpha\}$ and loop:

$$\Gamma^{a,\cdot} = \{\alpha | \alpha(s) = R(s,a)\}$$

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

$$\Gamma^{a} = \Gamma^{a,\cdot} \circ \Gamma^{a,o_1} \circ \Gamma^{a,o_2} \ldots$$

$$\Gamma_{n+1} = . \cup \Gamma^{a}$$

Formal Derivation

Theorem: The following conditions are sufficient for all permutations of the belief function to have the same value:

For every state permutation π(s) and action a
There exists an action a_π and observation permutation $\pi_{\pi,a}(o)$

Such that

R(s , a) = R(π(s) , a_π)
O(o | s ,a) = O($\pi_{\pi,a}$(o) | π(s) , a_π)
T(s' | s , a) = T(π(s') | π(s) , a_π)

For any action, no matter how we swap around the states,

we can find an action and a way to swap around the observations

so that the parameters are remain the same

The proof follows from substituting the sufficient condition into the Bellman optimality equations for POMDPs. In particular, we show that if a particular alpha vector is part of the value function solution, then all permutations of alpha are also part of the solution.

Suppose we have an alpha vector created in the following way from the equations above:

$$\alpha_i(s) = R(s,a) + \gamma T(.|s,a) \sum_{o \in O} O(o|s,a) \alpha_{i-1,o}$$

Applying the sufficient conditions:

$$\alpha_i(\pi(s)) = R(\pi(s),a_\pi) + \gamma T(.|\pi(s),a_\pi) \sum_{o \in O} O(\pi_{\pi,a}(o)|\pi(s),a_\pi) \alpha_{i-1,o}$$

So its permutation is also present.

Example: given π(Rome) = London and action "Give location"
Let a_π = "Give location" and $\pi_{\pi,a}(o_s) = o_{\pi(s)}$ and note:

- R(Rome , "Give location") = R(π(Rome) , "Give location")
- O(Rome | Rome , "Give location") = O(London | London , "Give location")
- T(Rome | Rome , "Give location") = T(London | London , "Give location")

Algorithm

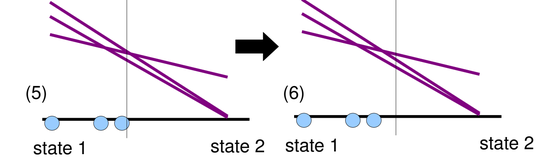
Point-based solvers. We can only solve a POMDP exactly if it is very small. A set of approximation algorithms perform backup operations on a limited set of beliefs. A generic point based solver might have the form:

- Sample a set of beliefs.
- Loop:
  - Compute action-observation value vectors: $\Gamma^{a,o} = \{\alpha | \alpha(s) = \gamma \sum_{s' \in S} T(s'|s,a) O(o|s',a) \alpha'(s')\}, \forall \alpha' \in \Gamma_n$
  - Compute the action value vectors: $\Gamma_b^a = R(,a) + \sum_{o \in O} argmax_{\alpha \in \Gamma^{a,o}} (\alpha \cdot b)$
  - Compute the new value function: $\Gamma_{n+1} = argmax_{\Gamma_b^a, a} (\Gamma_b^a \cdot b)$

Where
Γ: value function; dot(Γ , b) is the value of being in belief b.
Γ^a_b: action value function; dot(Γ^a_b , b) is the value of being in b and taking action a.
$\Gamma^{a,o}$: action-observation value function; dot($\Gamma^{a,o}$, b) is the value of being in b, taking action a, and seeing observation o.

Permutable POMDP. In this case, we know that if one alpha vector is part of the set, all permutations of the alpha vector are part of the set. Thus, we can limit ourselves to a small set of of belief samples. The algorithm proceeds as follows:

- (1) Sample a set of beliefs
- (2) Sort beliefs in descending order, remove nearby points.
- Loop:
  - (3) Compute action-observation value vectors:
    $\Gamma^{a,o} = \{\alpha | \alpha(s) = \gamma \sum_{s' \in S} T(s'|s,a) O(o|s',a) \alpha'(s')\}, \forall \alpha' \in \Gamma_n$
  - (4) Sort the $\Gamma^{a,o}$ in descending order. This is equivalent to choosing the permutation of $\Gamma^{a,o}$ that will maximize the dot product with the sorted belief. (Permuting is allowed because we know that all permutations of $\Gamma^{a,o}$ are valid vectors).
  - (5) Compute the action value vectors:
    $\Gamma_b^a = R(,a) + \sum_{o \in O} argmax_{\alpha \in \Gamma^{a,o}} (\alpha \cdot b)$
  - (6) Compute the new value function:
    $\Gamma_{n+1} = argmax_{\Gamma_b^a, a} (\Gamma_b^a \cdot b)$



If the permutation symmetry exists in only part of the POMDP, we can also simply sort the parts of the beliefs where the symmetry exists.

To choose an action using the value function, we must transform the original, unsorted belief into the permuted space:

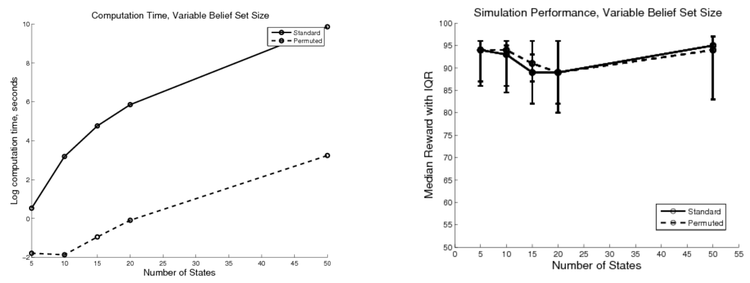
- Sort the current belief b to b_s; let s be the permutation π_s that takes b to b_s.
- Determine the optimal action a for b_s using the vectors explicitly represented in Γ.
- Perform the action a_p that corresponds to the permutation π_s and action a.

Results

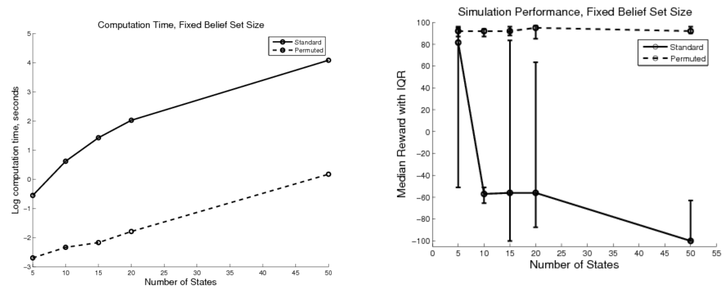
Scenario:
- Three types of actions: "ask," "confirm," "submit"
- Unique observation associated with each state
- Pr[hear correct state] from "ask" = .5
- Pr[hear correct confirmation] from "confirm" = .8
- Varied the number of states.

Two types of tests:
- let the belief set size grow with the number of states (more fair to the generic algorithm)
- fix belief set size (more reasonable if time or memory is limited)

Growing belief size: we see that performance is similar, but the permuted solver takes exponentially less time.



Fixed belief size: we see that the permuted version is again faster and its performance does not degrade even with a small set of supporting beliefs.



Conclusions and Future Work

- Conclusion: The permuted POMDP s a useful trick for solving POMDPs with a particular structure. This trick can also be used if a substructure of the POMDP has this property.
- Future work:
  - determine a more useful set of necessary conditions.
  - are there approximate ways to apply this trick for POMDPs with not quite permutable structure?