Exploiting Belief Bounds: Practical POMDPs for Personal Assistant Agents

Pradeep Varakantham, Rajiv Maheswaran, and Milind Tambe
Department of Computer Science
University of Southern California
Los Angeles, CA, 90089
{varakant, maheswar, tambe}@usc.edu

ABSTRACT

Agents or agent teams deployed to assist humans often face the challenges of monitoring the state of key processes in their environment (including the state of their human users themselves) and making periodic decisions based on such monitoring. POMDPs appear well suited to enable agents to address these challenges, given the uncertain environment and cost of actions, but optimal policy generation for POMDPs is computationally expensive. This paper introduces three key techniques to speedup POMDP policy generation that exploit the notion of progress or dynamics in personal assistant domains. Policy computation is restricted to the belief space polytope that remains reachable given the progress structure of a domain. We introduce new algorithms; particularly one based on applying Lagrangian methods to compute a bounded belief space support in polynomial time. Our techniques are complementary to many existing exact and approximate POMDP policy generation algorithms. Indeed, we illustrate this by enhancing two of the fastest existing algorithms for exact POMDP policy generation. The order of magnitude speedups demonstrate the utility of our techniques in facilitating the deployment of POMDPs within agents assisting human users.

Categories and Subject Descriptors

I.2.8 [Artificial Intelligence]: Distributed Artificial Intelligence - Multi Agent Systems

General Terms

Algorithms

Keywords

Task Allocation, Meeting Rescheduling, Partially Observable Markov Decision Process (POMDP)

1. INTRODUCTION

Recent research has focused on individual agents or agent teams that assist humans in offices, at home, in medical care and in many other spheres of daily activities [16, 13, 6, 15, 9, 12]. Such agents must often monitor the evolution of a process or state over time (including that of the human, the agents are deployed to assist) and make periodic decisions based on such monitoring. For example, in office environments, agent assistants may monitor the location of users in transit and make decisions such as delaying, canceling meetings or asking users for more information [15]. Similarly, in assisting with caring for the elderly [13] and therapy planning [9, 12], agents may monitor users' states/plans and make periodic decisions such as sending reminders.

Unfortunately, such agents (henceforth referred to as personal assistant agents (PAAs)) must monitor and make decisions despite significant uncertainty in their observations (as the true state of the world may not be known explicitly) and actions (outcome of agents' actions may be non-deterministic). Furthermore, actions have costs, e.g., delaying a meeting has repercussions on attendees. Researchers have turned to decision-theoretic frameworks to reason about costs and benefits under uncertainty. However, this research has mostly focused on Markov decision processes (MDPs) [15, 9, 12], ignoring the observational uncertainty in these domains, and thus potentially degrading agent performance significantly and/or requiring unrealistic assumptions about PAAs' observational abilities. POMDPs (Partially Observable Markov Decision Processes) address such uncertainty, but the long run-times for generating optimal policies for POMDPs remains a significant hurdle in their use in PAAs.

Recognizing the run-time barrier to POMDP usage, previous work on POMDPs has made encouraging progress using two approaches. First is an exact approach, where one finds the optimal solution [1, 4]. However, despite advances, exact algorithms remain computationally expensive and currently do not scale to problems of interest in PAA domains. Second is an approximate approach, where solution quality is sacrificed for speed [17, 7, 5, 18]. Unfortunately, approximate algorithms often provide loose (or no) quality guarantees on the solutions, even though such guarantees are crucial for PAAs to inhabit human environments.

This paper aims to practically apply POMDPs to PAA domains by introducing novel speedup techniques that are particularly suitable for such settings. The key insight is that when monitoring users or processes over time, large but shifting parts of the belief space in POMDPs (i.e., regions of uncertainty) remain unreachable. Thus, we can focus policy computation on this reachable belief-space polytope that changes dynamically. For instance, consider a PAA monitoring a user driving to a meeting. Given knowledge of user's current location, the reachable belief region is bounded by the maximum probability of the user being in different locations at the next time step as defined by the transition function. Similarly, in a POMDP where decisions are made every 5 minutes, an agent can exploit the fact that there is zero probability of going from a world state with $Time$ = 1:00 PM to a world state with $Time$ = 1:30 PM. Current POMDP algorithms typically fail to exploit such belief region reachability properties. POMDP algorithms that restrict belief regions fail to do so dynamically [14, 10].

Our techniques for exploiting belief region reachability exploit three key domain characteristics: (i) not all states are reachable at each decision epoch, because of limitations of physical processes or progression of time; (ii) not all observations are obtainable, because not all states are reachable; (iii) the maximum probability of reaching specific states can be tightly bounded. We introduce polynomial time techniques based on Lagrangian analysis to compute tight bounds on belief state probabilities. These techniques are complementary to most existing exact and approximate POMDP algorithms. We enhance two state-of-the-art exact POMDP algorithms [1, 4] delivering over an order of magnitude speedup for two different PAA domains.

2. MOTIVATING PERSONAL ASSISTANT AGENT (PAA) DOMAINS

We present two motivating examples, where teams of software PAAs assist human users in an office setting[15, 6]. The first is a meeting rescheduling problem (MRP), as implemented in the Electric-Elves system [15]. In this large-scale operationalized system, agents monitored the location of users and made decisions such as: (i) delaying the meeting if the user is projected to be late; (ii) asking user for information if he/she plans to attend the meeting; (iii) canceling the meeting; (iv) waiting. The agent relied on MDPs to arrive at decisions, as its actions such as asking had non-deterministic outcomes (e.g. a user may or may not respond) and decisions such as delaying had costs. The MDP state represented user location, meeting location and time to the meeting (e.g., user@home, meeting@USC, 10 minutes) and a policy mapped such states to actions. Unfortunately, observational uncertainty about user location was ignored while computing the policy.

A second key example is a task management problem (TMP) domain [6]. In this domain, a set of dependent tasks (e.g. T1, T2, T3 in Figure 1) is to be performed by human users (e.g. users U1, U2, U3 in Figure 1) before a deadline. Agents (e.g. A1, A2, A3 in Figure 1) monitor progress of humans and make reallocation decisions. Lines connecting agents and users indicate the communication links. An illustration of reallocation is the following scenario: suppose T1, T2 and T3 are assigned to U1, U2 and U3 respectively based on their capabilities. However, if U1 is observed to be progressing too slowly on T1, e.g., U1 may be unwell, then A1 may need to reallocate T1 to ensure that the tasks finish before the deadline. A1 may reallocate T1 to U2, if U2's original task T2 is nearing completion and U2 is known to be more capable than U3 for T1. However, if U2 is also progressing slowly, then T1 may have to be reallocated to U3 despite potential loss in capability. POMDPs provide a framework to analyze and obtain policies in domains such as MRP and TMP. In a TMP, a POMDP policy can take into account the possibly uneven progress of different users, e.g., some users may make most of their progress well before the deadline, while others do the bulk of their work closer to the deadline. In contrast, an instantaneous decision-maker cannot take into account such dynamics of progress. For instance, consider a TMP scenario where there are five levels of task progress $x \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$ and five decision points before

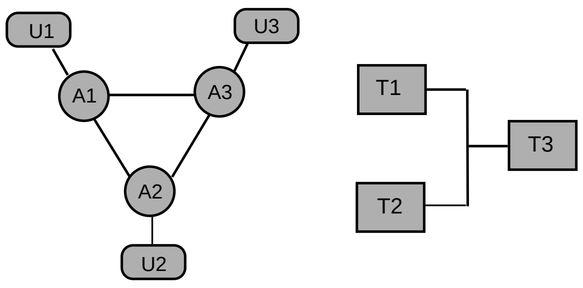

Figure 1: Comm. Structure and Task Dependency

the deadline $t \in \{1, 2, 3, 4, 5\}$. Observations are five levels of task progress $\{0.00, 0.25, 0.50, 0.75, 1.00\}$ and time moves forward in single steps, i.e. $T([x, t], a, [\tilde{x}, \tilde{t}]) = 0$ if $\tilde{t} \neq t + 1$. While transition uncertainty implies irregular task progress, observation uncertainty implies agent may observe progress x as for instance x or $x + 0.25$ (unless $x = 1.00$). Despite this uncertainty in observing task progress, a PAA needs to choose among waiting (W), asking user for info (A), or reallocate (R). A POMDP policy tree that takes into account both the uncertainty in observations and future costs of decisions, and maps observations to actions, for this scenario is shown in Figure 2 (nodes=actions, links=observations). In more complex domains with additional actions such as delaying deadlines, cascading effects of actions will require even more careful planning afforded by POMDP policy generation. Such scenarios in TMP and MRPs are investigated in Section 5.

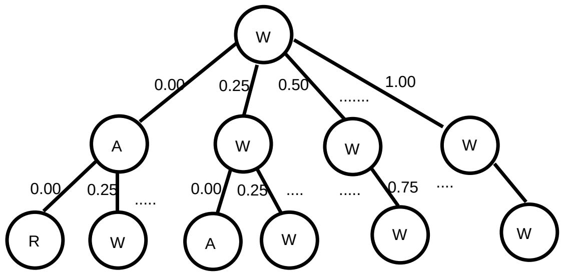

Figure 2: Partial Sample Policy for a TMP

3. POMDPS AND GENERALIZED INCREMENTAL PRUNING

A POMDP can be represented using the tuple $\{S, A, T, O, \Omega, R\}$, where S is a finite set of states; A is a finite set of actions; Ω is a finite set of observations; $T(s, a, s')$ provides the probability of transitioning from state s to s' when taking action a; $O(s', a, o)$ is probability of observing o after taking an action a and reaching s'; $R(s, a)$ is the reward function. A belief state b, is a probability distribution over the set of states S. A value function over a belief state is defined as:
$V(b) = \max_{a \in A} \{R(b, a) + \beta \, \Sigma_{b' \in B} T(b, a, b') V(b')\}$.
Currently, the most efficient exact algorithms for POMDPs are value iteration algorithms, specifically GIP [1] and RBIP [4]. These are dynamic programming algorithms, where at each iteration the value function is represented with a minimal set of dominant vectors called the parsimonious set. Given a parsimonious set at time t,

$\mathcal{V}_t$, we generate the parsimonious set at time $t-1$, $\mathcal{V}_{t-1}$ as follows (notation similar to the one used in [1] and [4]):

1. $\{v_{t-1}^{a,o,i}(s) = r(s,a)/|\Omega| + \beta\, \Sigma_{s' \in S} Pr(o, s'|s,a) v_t^i(s')\} =: \hat{\mathcal{V}}_{t-1}^{a,o}$ where $v_t^i \in \mathcal{V}_t$.
2. $\mathcal{V}_{t-1}^{a,o} = PRUNE(\hat{\mathcal{V}}_{t-1}^{a,o})$
3. $\mathcal{V}_{t-1}^{a} = PRUNE(\cdots(PRUNE(\mathcal{V}_{t-1}^{a,o_1} \oplus \mathcal{V}_{t-1}^{a,o_2}) \cdots \oplus \mathcal{V}_{t-1}^{a,o_{|\Omega|}})$
4. $\mathcal{V}_{t-1} = PRUNE(\bigcup_{a \in A} \mathcal{V}_{t-1}^{a})$

Each $PRUNE$ call executes a linear program (LP) which is recognized as a computationally expensive phase in the generation of parsimonious sets [1, 4]. Our approach effectively translates into obtaining speedups by reducing the quantity of these calls.

4. DYNAMIC BELIEF SUPPORTS

Our approach consists of three key techniques: (i) dynamic state spaces (DS); (ii) dynamic observation sets (DO); (iii) dynamic belief supports (DB). These ideas may be used to enhance existing POMDP algorithms such as GIP and RBIP. The key intuition is that for personal assistant domains, *progress* implies a dynamically changing polytope (of belief states) remains reachable through time, and policy computation can be speeded up by computing the parsimonious set over just this polytope. The speedups are due to the elimination of policies dominant in regions outside this polytope. DS provides an initial bound on the polytope and DO exploits this bound to limit the space of possible vectors at the next iteration. DB (which captures DS) provides tighter bounds on reachable belief states through a polynomial-time technique obtained from Lagrangian analysis.

These techniques do not alter the relevant parsimonious set w.r.t. reachable belief states and thus, *yield an optimal solution* over the reachable belief states. The resulting algorithms (DS,DO,DB) applied to enhance GIP are shown in Algorithm 1, where the functions GET-BOUND and DSDODB-GIP are the main additions, with significant updates in other GIP functions (otherwise, the GIP descriptions follows [1,3]). We discuss our key enhancements in Algorithm 1 at the end of each subsection below. Since the enhancements exploit the dynamics (transitions over time), these are applicable only to finite horizon problems.

4.1 Dynamic State Spaces (DS)

A natural method for PAAs to represent a user's state (such as in TMP) is with one consisting of a spatial element, (in a TMP, capturing the progress of each task), and a temporal element, capturing the stage of the decision. The transition matrix is then a static function of the state. This approach is used in [15] for an adjustable autonomy problem addressed with MDPs. We note that in these kinds of domains, one cannot reach all states from a given state. For example, in the scenario presented in Section 2, if there are limits on how tasks progress (one cannot advance more than one progress level in one time step, $T([x,t],a,[\tilde{x},t+1]) = 0$ if $\tilde{x} - x > 0.25$) and we know that at $t = 1$ we are at either $x = 0.00$ or $x = 0.25$, then we know at $t = 2$, $x \notin \{0.75, 1.00\}$ and at $t = 3$, $x \neq 1.00$. This implies that the state space at each point in time can be represented more compactly in a dynamic fashion.

Algorithm 1 `DSDODB + GIP`

```
Func POMDP-SOLVE ($L, S, A, T, \Omega, O, R$)
 1: $(\{S_t\}, \{O_t\}, \{B_t^{max}\})$ = DSDODB-GIP $(L, S, A, T, \Omega, O, R)$
 2: $t \leftarrow L$;$V_t \leftarrow 0$
 3: for $t = L$ to 1 do
 4:     $\mathcal{V}_{t-1}$ = DP-UPDATE($\mathcal{V}_t, t$)
Func DP-UPDATE ($\mathcal{V}, t$)
 1: for all $a \in A$ do
 2:     $\mathcal{V}_{t-1}^a \leftarrow \phi$
 3:     for all $\omega_t \in O_t$ do
 4:         for all $v_t^i \in V$ do
 5:             for all $s_{t-1} \in S_{t-1}$ do
 6:                 $v_{t-1}^{a,\omega_t,i}(s_{t-1}) = r_{t-1}(s_{t-1},a)/|O_t| + \gamma \Sigma_{s_t \in S_t} Pr(\omega_t, s_t|s_{t-1},a) v_t^i(s_t)$
 7:         $\mathcal{V}_{t-1}^{a,\omega_t} \leftarrow PRUNE(\{v_{t-1}^{a,\omega_t,i}\}, t)$
 8:     $\mathcal{V}_{t-1}^a \leftarrow PRUNE(\mathcal{V}_{t-1}^a \oplus \mathcal{V}_{t-1}^{a,\omega_t}, t)$
 9: $\mathcal{V}_{t-1} \leftarrow PRUNE(\bigcup_{a \in A} \mathcal{V}_{t-1}^a, t)$
10: return $\mathcal{V}_{t-1}$
Func POINT-DOMINATE($w, U, t$)
 1: for all $u \in U$ do
 2:     if $w(s_t) \leq u(s_t), \forall s_t \in S_t$ then return true
 3: return false
Func LP-DOMINATE($w, U, t$)
 1: LP vars: $d, b(s_t)[\forall s_t \in S_t]$
 2: LP max $d$ subject to:
 3:         $b \cdot (w - u) \geq d, \forall u \in U$
 4:         $\Sigma_{s_t \in S_t} b(s_t) \leftarrow 1$
 5:         $b(s_t) <= b_t^{max}(s_t)$; $b(s_t) >= 0$
 6: if $d \geq 0$ return $b$ else return nil
Func BEST($b, U$)
 1: $max \leftarrow Inf$
 2: for all $u \in U$ do
 3:     if $(b \cdot u > max)$ or $((b \cdot u = max)$ and $(u <_{lex} w))$ then
 4:         $w \leftarrow u$; $max \leftarrow b \cdot u$
 5: return $w$
Func PRUNE($U, t$)
 1: $W \leftarrow \phi$
 2: while $U \neq \phi$
 3: $u \leftarrow$ any element in $U$
 4: if POINT-DOMINATE($u, W, t$) = true then
 5:     $U \leftarrow U - u$
 6: else
 7:     $b \leftarrow$LP-DOMINATE($u, W, t$)
 8:     if $b = nil$ then $U \leftarrow U - u$
 9:     else $w \leftarrow BEST(b, U)$;$W \leftarrow W \bigcup w$;$U \leftarrow U - w$
10: return $W$
Func DSDODB-GIP($L, S, A, T, \Omega, O, R$)
 1: $t \leftarrow 1$; $S_t$ =Set of starting states
 2: for all $s_t \in S_t$ do
 3:     $b_t^{max}(s_t) = 1$
 4: for $t = 1$ to $L - 1$ do
 5:     for all $s \in S_t$ do
 6:         ADD-TO($S_{t+1}$,REACHABLE-STATES($s, T$))
 7:         $\Omega_{t+1}$ = GET-RELEVANT-OBS($S_{t+1}, O$)
 8:         C = GET-CONSTRAINTS ($s_t$)
 9:         $b_{t+1}^{max}(s_{t+1})$ = MAX$_{c \in C}$(GET-BOUND($s_{t+1}, c$))
10: return $(\{S_t\}, \{\Omega_t\}, \{b_t^{max}\})$
```

```
Func GET-BOUND($s_t, constraint$)
1:  y$_{min}$ = MIN$_{s \in S_{t-1}}(constraint.c[s]/constraint.d[s])$
2:  y$_{max}$ = MAX$_{s \in S_{t-1}}(constraint.c[s]/constraint.d[s])$
3:  INT = GET-INTERSECT-SORTED($constraint, y_{min}, y_{max}$)
4:  for all $i \in$ INT do
5:    $Z$ = SORT((($i + \epsilon$) $*$ $constraint.d[s] - constraint.c[s]$),$\forall s \in S_{t-1}$
6:    $sumBound = 1, numer = 0, denom = 0$
7:    /* IN ASCENDING ORDER */
8:    for all $z \in Z$ do
9:      $s$ = FIND-CORRESPONDING-STATE($z$)
10:     if $sumBound - bound[s_{t-1}] > 0$ then
11:       $sumBound- = bound[s_{t-1}]$
12:       $numer+ = bound[s_{t-1}] * constraint.c[s_{t-1}]$
13:       $denom+ = bound[s_{t-1}] * constraint.d[s_{t-1}]$
14:     if $sumBound - bound[s_{t-1}] <= 0$ then
15:       $numer+ = sumBound * constraint.c[s_{t-1}]$
16:       $denom+ = sumBound * constraint.d[s_{t-1}]$
17:       BREAK-FOR
18:   if $numer/denom > i$ and $numer/denom < max$ then
19:     return $numer/denom$
```

This will require the transition matrix and reward function to be dynamic themselves. Given knowledge about the initial belief space (e.g. possible beginning levels of task progress), we show how we can obtain dynamic state spaces and also that this representation does not affect the optimality of the POMDP solution. Let L be the length of a finite horizon decision process. Let S be the set of all possible states that can be occupied during the process. At time t, let $S_t \subset S$ denote the set of all possible states that could occur at that time. Thus, for any reachable belief state, we have $\sum_{s_t \in S_t} b_t(s_t) = 1$. Then, we can obtain S_t for $t \in 1, \ldots L$ inductively if we know the set $S_0 \subset S$ for which $s \notin S_0 \Rightarrow b_0(s) = 0$, as follows:

$$S_{t+1} = \left\{ s' \in S : \exists\, a \in A, s \in S_t \text{ s.t. } T_t(s, a, s') > 0 \right\} \quad (1)$$

The belief probability for a particular state $\tilde{s}$ at time $t+1$ given a starting belief vector at time t (b_t) action (a) and observation (ω) can be expressed as follows:

$$b_{t+1}(\tilde{s}) := \frac{O_t(\tilde{s}, a, \omega) \sum_{s_t \in S_t} T_t(s_t, a, \tilde{s}) b_t(s_t)}{\sum_{s_{t+1} \in S_{t+1}} O_t(s_{t+1}, a, \omega) \sum_{s_t \in S_t} T_t(s_t, a, s_{t+1}) b_t(s_t)}$$

This implies that the belief vector b_{t+1} will have support only on S_{t+1}, i.e. $\tilde{s} \notin S_{t+1} \Rightarrow b_{t+1}(\tilde{s}) = 0$, if b_t only has support in S_t and S_{t+1} is generated as in (1). Thus, we can model a process that migrates among dynamic state spaces $\{S_t\}_{t=1}^{L}$ indexed by time or more accurately, the stage of the decision process as opposed to a transitioning within static global state set S.

PROPOSITION 1. *Given S_0, we can replace a static state space S with dynamic state spaces $\{S_t\}$ generated by (1), dynamic transition matrices and dynamic reward functions in a finite horizon POMDP without affecting the optimality of the solution obtained using value function methods.*

Proof. If we let P_t denote the set of policies available at time t, V_t^p denote the value of policy p at time t and, V_t^* denote the value of the optimal policy at time t, we have $V_L^*(b_L) = \max_{p \in P_L} b_L \cdot \alpha_L^p$ where $\alpha_L^p = [V_L^p(s_1) \cdots V_L^p(s_{|S|})]$ for $s_i \in S$.

When $t = L$, we have $V_L^p(s) = R_L(s, a(p))$ where R_L is the reward function at time L and $a(p)$ is the action prescribed by the policy p. Since $b_L(s) = 0$ if $s \notin S_L$, then $V_L^*(b_L) = \max_{p \in P_L} \tilde{b}_L \cdot \tilde{\alpha}_L^p$ where $|\tilde{b}_L| = |\tilde{\alpha}_L^p| = |S_L|$ and $\tilde{\alpha}_L^p = [V_L^p(\tilde{s}_1) \cdots V_L^p(\tilde{s}_{|S_L|})]$ for $\tilde{s}_i \in S_L$. Calculating the value function at time $L-1$, we have $V_{L-1}^*(b_{L-1}) = \max_{p \in P_{L-1}} b_{L-1} \cdot \alpha_{L-1}^p$ where $\alpha_{L-1}^p = [V_{L-1}^p(s_1) \cdots V_{L-1}^p(s_{|S|})]$ for $s_i \in S$.

When $t = L-1$, we have
$V_{L-1}^p(s) = R_{L-1}(s, a(p)) +$
$\gamma \sum_{s' \in S} T_{L-1}(s, a(p), s') \sum_{\omega \in \Omega} O(s', a, \omega) V_L^{p_\omega}(s')$,

where $p_\omega \in P_L$ is the policy subtree of the policy tree $p \in P_{L-1}$ when observing ω after the initial action. Since $b_{L-1}(s) = 0$ if $s \notin S_{L-1}$, then $V_{L-1}(b_{L-1}) = \max_{p \in P_{L-1}} \tilde{b}_{L-1} \cdot \tilde{\alpha}_{L-1}^p$ where $|\tilde{b}_{L-1}| = |\tilde{\alpha}_{L-1}^p| = |S_{L-1}|$ and $\tilde{\alpha}_{L-1}^p = [V_L^p(\tilde{s}_1) \cdots V_L^p(\tilde{s}_{|S_{L-1}|}])$ for $\tilde{s}_i \in S_{L-1}$. Applying this reasoning inductively, we can show that we only need $V_t^p(s_t)$ for $s_t \in S_t$. Furthermore, if $s_t \in S_t$, then

$$V_t^p(s_t) = R_t(s_t, a(p)) + \gamma \sum_{s_{t+1} \in S_{t+1}} T_t(s_t, a(p), s_{t+1}) \sum_{\omega \in \Omega} O(s_{t+1}, a, \omega) V_{t+1}^{p_\omega}(s_{t+1}). \quad (2)$$

Thus, we only need $\{V_{t+1}^{\omega(p)}(s_{t+1}) : s_{t+1} \in S_{t+1}\}$. ■

The value functions expressed for beliefs over dynamic state spaces S_t have identical expected rewards as when using S. The advantage in this method is that in generating the set of value vectors which are dominant at some underlying belief point (i.e. the parsimonious set) at a particular iteration, we eliminate vectors that are dominant over belief supports that are not reachable. This reduces the set of possible policies that need to be considered at the next iteration. Line 6 of DSDODB-GIP function and the DP-UPDATE function of Algorithm 1 provide the algorithm for finding the dynamic states.

4.2 Dynamic Observation Spaces (DO)

We note that in some domains, certain observations can only be obtained from certain states. Consequently, dynamic state spaces imply that the observations capable of being obtained at a particular time will also be dynamic. For instance, consider the TMP scenario presented in Section 2, with the conditions on transition probabilities from Subsection 4.1. Because the dynamic state space at $t = 1$ limits us to being in one of two progress levels, $(0.00 or 0.25)$, then we will not be able to get the observations 0.75 or 1.00 regardless of the action we take at this time. We now show how to obtain these dynamic observation sets and prove that they do not affect the value iteration process.

PROPOSITION 2. *Given dynamic state spaces $\{S_t\}$, we can replace a static observation space Ω with a dynamic observation spaces $\Omega_t := \{\omega \in \Omega : \exists a \in A, s \in S_{t+1}$ s.t. $O(s', a, \omega) > 0\}$ in a finite horizon POMDP without affecting the optimality of the solution obtained using value function methods.*

Proof. Given (2) from Subsection 4.1, we can rewrite $V_t^p(s_t)$ as

$$R_t(s_t, a(p)) + \gamma \sum_{s_{t+1} \in S_{t+1}} T_t(s_t, a(p), s_{t+1})$$
$$\cdot \left\{ \sum_{\omega \in \Omega_t} O(s_{t+1}, a, \omega) V_{t+1}^{p_\omega}(s_{t+1}) + \sum_{\omega \in \Omega_t^C} O(s_{t+1}, a, \omega) V_{t+1}^{p_\omega}(s_{t+1}) \right\}$$

where Ω_t^C is the set complement of Ω. Because of the dynamic observations, the second part of the sum goes to zero. This implies that only the observations in Ω_t are relevant to the value of a strategy at time t. Thus, when creating policy trees, the subtrees p_ω are not necessary if $\omega \notin \Omega_t$. ■

This further reduces the set of policies/vectors being generated before pruning which reduces the number of LP calls during pruning. For consistency, we now index the observation probability matrix with time as it depends on a dynamic state. Line 7 in DSDODB-GIP function and the DP-UPDATE function of Algorithm 1 provide the procedure utilizing DO.

4.3 Dynamic Belief Spaces (DB)

By introducing dynamic state spaces, we are attempting to more accurately model the support on which reachable beliefs will occur. We can make this process more precise by using information about the initial belief distribution, the transition and observation probabilities to bound belief dimensions with positive support. For example, if we know that our initial belief regarding task progress can have at most 0.10 probability of being at 0.25 with the rest of the probability mass on being at 0.00, we can find the maximum probability of being at 0.25 or 0.50 at the next stage, given a dynamic transition matrix. Below we outline a polynomial-time procedure by which we can obtain such bounds on belief support.

Let $B_t \subset [0\ 1]^{|S_t|}$ be a space such that $P(b_t \notin B_t) = 0$. That is, there exists no initial belief vector and action/observation sequence of length $t-1$ such that by applying the standard belief update rule, one would get a belief vector b_t not captured in the set B_t. Then, we have

$$b_{t+1}(s_{t+1}) \geq \min_{a\in A, o\in O_t, b_t \in B_t} F(s_{t+1}, a, o, b_t) =: b_{t+1}^{\min}(s_{t+1})$$
$$b_{t+1}(s_{t+1}) \leq \max_{a\in A, o\in O_t, b_t \in B_t} F(s_{t+1}, a, o, b_t) =: b_{t+1}^{\max}(s_{t+1})$$

where $F(s_{t+1}, a, o, b_t) :=$

$$\frac{O_t(s_{t+1}, a, o)\sum_{s_t\in S_t} T_t(s_t, a, s_{t+1}) b_t(s_t)}{\sum_{\tilde{s}_{t+1}\in S_{t+1}} O_t(\tilde{s}_{t+1}, a, o)\sum_{s_t\in S_t} T_t(s_t, a, \tilde{s}_{t+1}) b_t(s_t)}$$

Thus, if

$$B_{t+1} = [b_{t+1}^{\min}(s_1) b_{t+1}^{\max}(s_1)] \times \cdots \times [b_{t+1}^{\min}(s_{|S_{t+1}|}) b_{t+1}^{\max}(s_{|S_{t+1}|})],$$

then we have $P(b_{t+1} \notin B_{t+1}) = 0$.

We now show how $b_{t+1}^{\max}(s_{t+1})$ (and similarly $b_{t+1}^{\min}(s_{t+1})$) can be generated through a polynomial-time procedure deduced from Lagrangian methods. Given an action a and observation ω, we can express the problem as

$$\max_{b_t\in B_t} b_{t+1}^{a,\omega}(s_{t+1}) \quad \text{s.t.} \quad b_{t+1}^{a,\omega}(s_{t+1}) = c^T b_t / d^T b_t$$

where $c(s) = O_t(s_{t+1}, a, \omega) T_t(s_t, a, s_{t+1})$ and $d(s) = \sum_{s_{t+1}\in S_{t+1}} O_t(s_{t+1}, a, \omega) T_t(s_t, a, s_{t+1})$. We rewrite the problem in terms of the new variables as follows:

$$\min_x \left(-c^T x / d^T x\right) \quad \text{s.t.} \quad \sum_i x_i = 1,\ 0 \leq x_i \leq b_t^{\max}(s_i) =: \bar{x}_i - (3)$$

where $\sum_i b_t^{\max}(s_i) \geq 1$ to ensure existence of a feasible solution. Expressing this problem as a Lagrangian, we have

$$\mathcal{L} = \left(-c^T x / d^T x\right) + \lambda(1 - \sum_i x_i) + \sum_i \bar{\mu}_i (x_i - \bar{x}_i) - \sum_i \mu_i x_i$$

from which the KKT conditions imply

$$\begin{aligned}
x_k &= \bar{x_k} & \lambda &= [(c^T x) d_k - (d^T x) c_k]/(d^T x)^2 + \bar{\mu}_k \\
0 < x_k &< \bar{x_k} & \lambda &= [(c^T x) d_k - (d^T x) c_k]/(d^T x)^2 \\
x_k &= 0 & \lambda &= [(c^T x) d_k - (d^T x) c_k]/(d^T x)^2 - \mu_k.
\end{aligned}$$

Because λ is identical in all three conditions and $\bar{\mu}_k$ and μ_k are non-negative for all k, the state k with a lowest value of $(d^T x)\lambda = [(c^T x)/(d^T x)] d_k - c_k$ must receive a maximal allocation (assuming $\bar{x}_k < 1$) or the entire allocation otherwise. Using this reasoning recursively, we see that if x^* is an extremal point (i.e. a candidate solution), then the values of its components $\{x_k\}$ must be constructed by giving as much weight possible to components in the order prescribed by $z_k = y d_k - c_k$, where $y = (c^T x^*)/(d^T x^*)$. Given a value of y, one can construct a solution by iteratively giving as much weight as possible (without violating the equality constraint) to a component that is not already at its bound with the lowest z_k.

Here is an example to illustrate the method introduced above. If the size of state space is three (k = 1,2,3) and the values of the expression $[(c^T x)/(d^T x)] d_k - c_k$ for different states, k are 5, 6, 7. Since λ (over all x_k) is identical, the above values need to be made equal by deciding on the allocations for each of the x_k. Since $\sum_k x_k = 1$, it cannot be the case that all these values are reduced, since reduction happens only in the third equation for λ where $x_k = 0$ (since there is a subtraction of non negative variable μ_k). Thus, it is imperative that smaller of these values increase. As can be observed from the equations of λ (for a particular k), values can be increased only in the case of $x_k = \bar{x_k}$ (since there is a non negative variable $\bar{\mu}_k$ in the equation), and hence full allocation for smaller values of $(d^T x)\lambda = [(c^T x)/(d^T x)] d_k - c_k$ i.e value for state 1 (5), can be increased, by assigning $x_1 = \bar{x}_1$. Further, if $(1 - \bar{x}_1) \geq \bar{x}_2$, then value for state 2 (6), can also increase, since it also gets a full assignment. However, if $(1 - \bar{x}_1) < \bar{x}_2$, then value for state 2 should remain the same, while value for state 3 can decrease, since it gets a zero allocation. Based on this reasoning of making the values of λ equal (over all states k), the allocation of bounds is done.

The question then becomes finding the maximum value of y which yields a consistent solution. We note that y is the value we are attempting to maximize, which we can bound with $y_{\max} = \max_i c_i/d_i$ and $y_{\min} = \min_i c_i/d_i$. We also note that for each component k, z_k describes a line over the support $[y_{\min}, y_{\max}]$. We can then find the set of all points where the set of lines described by $\{z_k\}$ intersect. There can be at most $(N-1)N/2$ intersections points. We can then partition the support $[y_{\min}, y_{\max}]$ into disjoint intervals using these intersection points yielding at most $(N-1)N/2 + 1$ regions. In each region, there is a consistent ordering of $\{z_k\}$ which can be obtained in polynomial time. An illustration of this can be seen in Figure 4.3. Beginning with the region furthest to the right on the real line, we can create the candidate solution implied by the ordering of $\{z_k\}$ in that region and then calculate the value of y for that candidate solution. If the obtained value of y does not fall within region, then the solution is inconsistent and we move to the region immediately to the left. If the obtained value of y does fall within the region, then we have the candidate extremal point which yields the highest possible value of y, which is the solution to the problem. By using this technique we can dynamically propagate forward bounds on feasible belief states. Line 12 and 13 of the DSDODB-GIP function in Algorithm 1 provide the procedure for DB. The GET-CONSTRAINTS function on Line 12 gives the set of c^T and d^T vectors for each state at time t for each action and observation. 2

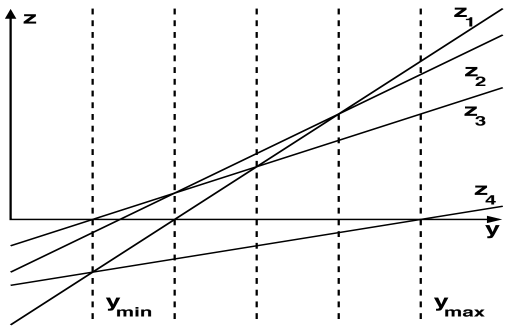

Figure 3: Partition Procedure for Solving Belief Maximization Lagrangian

While the Lagrangian method introduced above works in any general case, there are special cases where simpler methods are feasible to obtain belief bounds. For instance, in the belief maximization equation, if $b_t^{max}(s_i)$ is equal to 1 for all states s_i, then it can be easily proved that the maximum value is equal to max_k c_k/d_k. Thus this special case does not even require the complexity of the lagrangian method, and can be solved in O($|S|log(|S|)$). However, if the maximum possible value of belief probability in the previous stage is not equal to 1, max_k c_k/d_k can serve only as a bound and not the exact maximum. A simple improvement to the above method is assigning x_k their maximum value (until the sum is 1) based on the order of c_k/d_k. However, as can be observed in the example below, this method doesn't yield the maximum.

$$\max\left((0.06x_1 + 0.02x_2 + 0.07x_3)\,/\,(0.2x_1 + 0.09x_2 + 0.15x_3)\right)$$
$$\text{s.t.}\ \ 0 < x_1 < 0.8, 0 < x_2 < 0.6, 0 < x_3 < 0.8, \sum_i x_i = 1$$

In the above example, if the bounds are allocated based on the order of ratios of coefficients, then the value of the expression would be 0.44, while there exists an allocations (0.8 to x_3, and 0.2 to x_2), that gives a value of 0.45.

By using dynamic beliefs, we increase the costs of pruning by adding some constraints. However, there is an overall gain because we are looking for dominant vectors over a smaller support and this reduces the cardinality of the parsimonious set, leaving fewer vectors to consider at the next iteration.

5. EXPERIMENTAL RESULTS

Experiments were conducted on the TMPs and MRPs explained in Section 2. Each agent uses a POMDP for decision making in both domains. Our enhancements, DS (Dynamic States), DO (Dynamic Observations) and DB (Dynamic Beliefs), were implemented[1] over both GIP and RBIP [4][2] (RBIP is itself a recent enhancement to GIP). All the experiments[3] compare the performance (run-time) of GIP, RBIP and our enhancements over GIP and RBIP. For both domains, we ran 7 problems over all methods (GIP, RBIP, DS+GIP, DSDO+GIP, DSDODB+GIP, DS+RBIP, DSDO+RBIP, DSDODB + RBIP). Each problem had pre-specified run-time upper limit of 20000 seconds.

In our experiments, the sizes of the problem instances tested are as follows. For the TMP domain, the number of states, actions and observations for the smallest problem A were 65, 3, and 16 respectively, whereas for the largest problem G the numbers were 217, 3, and 37 respectively. For the MRP domain, the number of states, actions and observations for the smallest problem A were 109, 4, 10 respectively, whereas for the largest problem G, the numbers were 252, 4, and 17 respectively.

Figure 4(a)-(c) present results for the TMP domain. Experimental setup consisted of seven problems of increasing complexity (A through G). In all the graphs, the x-axis denotes the problem name, and the y-axis denotes the run-time for a problem. GIP and RBIP finished before the time limit only in Problem A, as shown in Figure 4(a). DS+GIP provides 100-fold speedup in Problem B, and 10-fold speedup in Problems C and D (the actual speedup, which we expect to be even larger cannot be seen due to the cutoff).

DSDO+GIP and DSDODB+GIP have same run-time as DS in A-C. Figure 4(b) provides comparisons among our enhancements on GIP. For Problems D-G that are complex than A-C, DSDODB dominates other enhancements providing approximately 5-fold speedup over DS. GIP and RBIP did not even terminate within time limit and hence not shown. The key point of Figure 4(c) is to show that DS+GIP provides 10-fold speedup (with cut-off) over DS+RBIP, even though RBIP is faster than GIP. This is also the reason for providing the results of enhancements on GIP instead of RBIP in Figure 4(b).

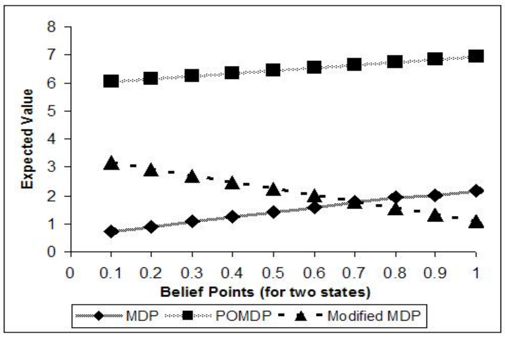

Figure 5: POMDP policy dominates MDP policy

Figure 4(d) presents results for the MRP domain. Experimental setup for MRP consisted of a set of seven problems(A through G). The figure does not show results for GIP and RBIP, because they did not finish before our cutoff for any of the problems. DSDO+GIP is not present in the figure, since it had run-times identical to DS + GIP. DSDODB+GIP provides approximately 6-fold speedup over DS. DS+RBIP seems comparable with the other three methods in A-C, but for D-G, it fails to even finish before the cutoff. Both domains provide similar conclusions: DSDODB+GIP dominates other techniques (with around 100 fold speedup over GIP and RBIP in some cases) and this dominance becomes significant in larger problems.

Figure 5 provides a quantitative reason for using POMDPs instead of MDPs for policy generation in the PAA domains. As in [15], MDPs can be used for obtaining policies, but they suffer from not handling observational uncertainty. Figure 5 emphasizes this point quantitatively, by comparing the expected values of optimal

[1]Our enhancements were implemented on Anthony Cassandra's POMDP solver "http://pomdp.org/pomdp/code/index.shtml"
[2]Since RBIP code was not available, we implemented RBIP presented in [4] on Anthony Cassandra's POMDP solver
[3]Machine specs for all experiments: Intel Xeon 2.8 GHZ processor, 2GB RAM, Linux Redhat 8.1

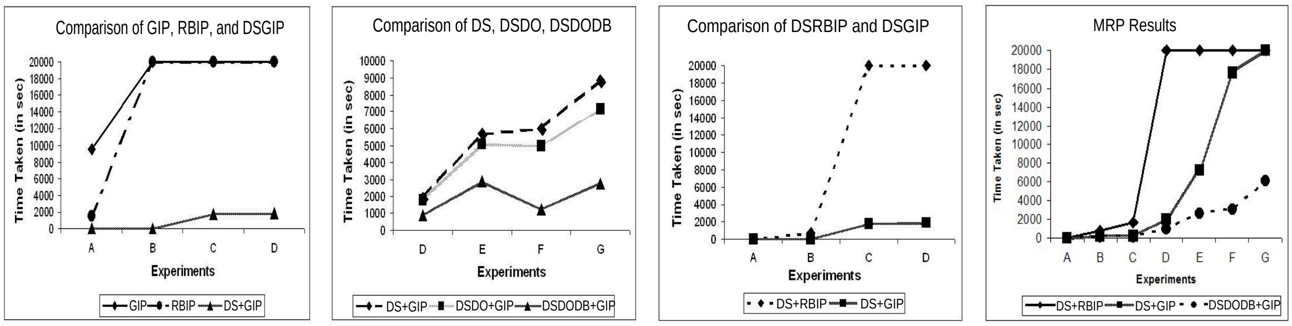

Figure 4: TMP: (a) DS+GIP gives orders of magnitude speedup over GIP and RBIP (b) DSDODB+GIP dominates other enhancements over GIP (c) DS+GIP dominates DS+RBIP; MRP: (d) DSDODB+GIP dominates

policies generated using an MDP and a POMDP on a small TMP problem. The MDP policy was evaluated in the presence of uncertainty by assuming the state (at any point in policy execution) to be the one with highest belief probability in the belief state (at that point). Modified-MDP policy is where the MDP policy is modified to take a *safe* action ("Ask User"), when there is high uncertainty in the belief state. x-axis denotes belief points (for two states), while the y-axis denotes the expected value of a policy. The expected value of a POMDP policy dominates the MDP policy by approximately 5 in a problem where expected values of policies vary from -2 (the value for always choosing the "Wait" action) to 7 (the highest reward under the POMDP policy). Thus, there could be significant quality loss incurred with a MDP policy.

6. RELATED WORK

We have already discussed some related work in Section 1. As discussed there, techniques for solving POMDPs can be categorized as exact and approximate. GIP [1] and RBIP [4] are exact algorithms, which we have enhanced. There has been work by [2] and [10] on ways of compactly representing dynamics of a domain. These compact representations however do not seem to have advantages in terms of the speedups [8]. Other exact algorithms attempt to exploit domain-specific properties to speedup POMDPs. For instance, [10] presents a hybrid framework that combines MDPs with POMDPs to take advantage of perfectly and partially observable components of the model. They also focus on reachable belief spaces, but: (i) their analysis does not capture dynamic changes in belief space reachability; (ii) their analysis is limited to factored POMDPs; (iii) no speedup measurements are shown. This contrasts with this work which focuses on dynamic changes in belief space reachability and its application to both flat and factored state POMDPs.

Approximate algorithms are faster than exact algorithms, but at the cost of solution quality. There has been a significant amount of work in this area, but point-based [17, 7], grid [5, 18], and policy search approaches [3, 11] dominate other algorithms. Though these approaches can solve larger problems, most of them provide loose (or no) quality guarantees on the solution. It is critical to have quality guarantees in PAA domains, for an agent to gain the trust of a human user. Point Based Value Iteration (PBVI)[7] provides quality guarantees, but to obtain good results, it needs to increase sampling, consequently increasing in run-time. Other approach that works with big problems is the dimensionality reduction technique in [14]. This work applies E-PCA (an improvement to Principal Component Analysis) on a set of belief vectors, to obtain a low dimensional representation of the original state space. Though this work provides reduction of dimension (state space), it doesn't provide any guarantees on the quality of solutions. Nonetheless, our techniques can also benefit approximate algorithms, e.g., PBVI can benefit from our DO technique.

7. SUMMARY

This paper provides techniques to make the application of POMDPs in personal assistant agents a reality. In particular, we provide three key techniques to speedup POMDP policy generation that exploit the key properties of the PAA domains. The key insight is that given an initial (possibly uncertain) starting set of states, the agent needs to generate a policy for a limited range of dynamically shifting belief states. The techniques we propose are complementary to most existing exact and approximate POMDP policy generation algorithms. Indeed, we illustrate our technique by enhancing GIP and RBIP, two of the most efficient exact algorithms for POMDP policy generation and obtain orders of magnitude speedup in policy generation. We provide a detailed algorithm illustrating our enhancements in Algorithm 1, and present proofs of correctness of our techniques. Techniques presented here facilitate agents' utilizing POMDPs for policies when assisting human users.

Acknowledgements. This material is based upon work supported by DARPA, through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010. Thanks to Eric Hansen, Nicholas Roy and Joelle Pineau for their valuable comments.

8. REFERENCES

[1] M. L. Littman A. R. Cassandra and N. L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *UAI*, 1997.

[2] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *AAAI*, 1996.

[3] D. Braziunas and Craig Boutilier. Stochastic local search for POMDP controllers. In *AAAI*, 2004.

[4] Z. Feng and S. Zilberstein. Region based incremental pruning for POMDPs. In *UAI*, 2004.

[5] M. Hauskrecht. Value-function approximations for POMDPs. *JAIR*, 13:33–94, 2000.

[6] http://www.ai.sri.com/project/CALO, http://calo.sri.com. *CALO: Cognitive Agent that Learns and Organizes*, 2003.

[7] G. Gordon J. Pineau and S. Thrun. PBVI: An anytime algorithm for POMDPs. In *IJCAI*, 2003.

[8] M. L. Littman L. P. Kaelbling and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *AI Journal*, 1998.

[9] T. Y. Leong and C. Cao. Modeling medical decisions in DynaMoL: A new general framework of dynamic decision analysis. In *World Congress on Medical Informatics (MEDINFO)*, pages 483–487, 1998.

[10] H. Fraser M. Hauskrecht. Planning treatment of ischemic heart disease with partially observable markov decision processes. *AI in Medicine*, 18:221–244, 2000.

[11] L. P. Kaelbling N. Menleau, K. E. Kim and A. R. Cassandra. Solving POMDPs by searching the space of finite policies. In *UAI*, 1999.

[12] F. Locatelli: P. Magni, R. Bellazzi. Using uncertainty management techniques in medical therapy planning: A decision-theoretic approach. In *Applications of Uncertainty Formalisms*, pages 38–57, 1998.

[13] M. E. Pollack, L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, and I. Tsamardinos. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44:273–282, 2003.

[14] N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *NIPS*, 2002.

[15] P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real-world. *JAIR*, 17:171–228, 2002.

[16] D. Schreckenghost, C. Martin, P. Bonasso, D. Kortenkamp, T.Milam, and C.Thronesbery. Supporting group interaction among humans and autonomous agents. In *AAAI*, 2002.

[17] N. L. Zhang and W. Zhang. Speeding up convergence of value iteration in partially observable markov decision processes. *JAIR*, 14:29–51, 2001.

[18] R. Zhou and E. Hansen. An improved grid-based approximation algorithm for POMDPs. In *IJCAI*, 2001.