Practical POMDPs for Personal Assistant Domains

Pradeep Varakantham, Rajiv T. 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 challenge of monitoring state of key processes in their environment, including the state of their human users, and making periodic decisions based on such monitoring. The challenge is particularly difficult given the significant observational uncertainty, and uncertainty in the outcome of agent's actions. POMDPs (partially observable markov decision problems) appear well-suited to enable agents to address such uncertainties and costs; yet slow run-times in generating optimal POMDP policies presents a significant hurdle. This slowness can be attributed to cautious planning for all possible belief states, e.g., the uncertainty in the monitored process is assumed to range over all possible states at all times.
This paper introduces three key techniques to speedup POMDP policy generation that exploit the notion of progress or dynamics in personal assistant domains. The key insight is that given an initial (possibly uncertain) starting set of states, the agent needs to be prepared to act only in a limited range of belief states; most other belief states are simply unreachable given the dynamics of the monitored process, and no policy needs to be generated for such 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 generalized incremental pruning (GIP), one of the most efficient exact algorithms for POMDP policy generation and illustrate orders of magnitude speedup in policy generation. Such speedup would facilitate agents' deploying POMDPs in assisting human users.

1. Introduction

Several recent research projects are focussing on individual agents or teams of agents that assist humans in offices, at home, in medical care and indeed in all spheres of daily human activities [2, 5, 9, 12]. One major area of responsibility for such agents is to monitor the evolution of some process or state over time, including that of the human the agents are deployed to assist, and to make periodic decisions based on such monitoring [2, 7, 9, 11]. Agents may accomplish such monitoring and decision making fully autonomously or with adjustable autonomy (dynamically reducing own autonomy for seeking human input). For example, in office environments, agent assistants have been deployed to monitor the location of individual users in transit, and make periodic decisions on delaying or canceling a meeting, or seeking more information from users [12]. Similarly, in caring for elderly, agent assistants monitor the progress of a plan of activites of the user and make decisions at regular intervals based on the activities performed [9]. Such monitoring and periodic decision making is also seen in therapy planning [11].

Unfortunately, such agents (henceforth referred to as personal assistant agents) must monitor and make decisions despite significant uncertainty in observations (so the true state of the world may not be known), and uncertainty in actions (outcome of agent's decisions or actions may be uncertain). Furthermore, actions have costs, e.g., delaying a meeting may have a cost to the meeting attendees. Researchers have naturally turned to decision-theoretic frameworks to reason about costs and benefits under uncertainty. However, this research has traditionally focused on markov decision problems or MDPs and its variants for decision making [7, 11, 12], ignoring the observational uncertainty in these domains, and thus potentially significantly degrading agent performance, and/or requiring unrealistic assumptions about agent's ability to monitor the state of the world. POMDPs (Partially Observable Markov Decision Problems) suggest themselves as natural candidates to address such observational uncertainty, and these have been recognized for some time as the natural next step [12] in the evolution of soft-

ware assistants. However, the long run-times for generating optimal policies for POMDPs remains a significant hurdle in their use in agent assistants.

Recognizing the hurdle in POMDP usage in practical domains, previous work on POMDPs has made encouraging progress using two types of approaches. The first approach is exact algorithms, which try to find the optimal solution [1, 3]. However, these exact algorithms remain computationally expensive and currently do not scale to large-scale problems of interest in personal assistant domains. The second approach is approximate algorithms, which tradeoff solution quality for speed [4, 6, 8, 14, 15]. Unfortunately, while approximate algorithms achieve significant speedups over exact algorithms, they often provide loose (or no) quality guarantees on the solutions — even though such quality guarantees are crucial if personal assistants are to inhabit human environments.

This paper aims to practically apply POMDPs to personal assistant domains by introducing novel speedup techniques that are particularly suitable for such domains. The key insight is that when monitoring users or processes over time, large parts of belief states in POMDPs — i.e., regions or states of uncertainty — are fundamentally unreachable. Such unreachable belief states change over time dynamically. However, current POMDP algorithms do not exploit such unreachable belief state regions, and plan for the entire set of belief states at each point in time. For instance, when a personal assistant is monitoring a user who is travelling to a meeting, the user can only travel a certain distance within a given time, i.e., the uncertainty over where the user may be at the next time step is limited to a few locations. Current POMDP algorithms would create policies assuming the uncertainty (belief states) is spread over all locations. Similarly, the state of the world cannot move forward from five minutes before a meeting to fifteen minutes past the meeting time without passing through a state of the actual meeting time. However, such information about dynamics of the world will currently be ignored by the POMDP algorithms.

This paper introduces three key techniques that exploit the sparseness of the reachable belief space in personal assistant domains. The enhancements that we present provide ways of simply avoiding policy generation for unreachable regions in the belief space. Following characteristics of the domains provide an insight about the enhancements: (1) Not all states are reachable at each decision epoch, because of increasing progress or time. (2) Not all observations are observable, because of the reduced number of states (from 1 above) (3) Belief probability of a state can be tightly bounded (rather than with just 0 and 1). These enhancements apart from being tailored to above kinds of personal assistant domains, are also complementary to most of the existing algorithms. While the improvements we suggest could be applied to a variety of algorithms (see Related Work), we demonstrate these improvements in the context of generalized incremental pruning [1], for three reasons. First, incremental pruning provides a public domain implementation, with a well-documented and well-understood algorithm. Second, incremental pruning provides an extremely efficient baseline algorithm — indeed, except for a recent report that improves a section of the algorithm, it is recognized as the most efficient exact algorithm to compare against. Third, the recent improvements are orthogonal to the key improvements discussed in our paper, and they do not change the basic framework of incremental pruning. Thus, these recent improvements would only add to the speedups demonstrated in our paper, although we demonstrate orders of magnitude improvements in performance already. Finally, it is critical to note that the same set of ideas could be applied to other algorithms also.

2. Personal Assistant Domains

Recent research in personal assistant domains has involved agents monitoring the state of the user or some other process in the environment and making periodic decisions based on the status of the user (or the monitored process). Multiple personal assistants may collaborate in order to assist collaborative user tasks. We discuss here two specific example domains from personal assistants deployed in office environments [5, 12] illustrating the challenges in such domains. One example is from an actual system implemented previously, and another is from an on-going system under development. However, the key challenges outlined arise in other personal assistant domains as well.

One key example is a meeting rescheduling domain, as implemented in the Electric-Elves system [12]. 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 the 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 nondeterministic 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 min-

utes) 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 [5]. In this domain, a set of dependent tasks (e.g., T1, T2, T3 in Figure 2) is to be performed by human users (e.g,. users U1, U2, U3 in Figure 2). Agents (e.g., A1, A2, A3 in Figure 2) monitor the progress of humans and make reallocation decisions. The lines connecting agents and users indicate the lines of communication. An illustration of reallocation is the following scenario: suppose T1, T2 and T3 are assigned to U1, U2 and U3 respectively based on their initial 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 three tasks finish before a given 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 be reallocated to U3 despite the potential loss in capability. Agents monitoring progress of dependent tasks is important. One reason for this can be seen in the following situation. When T1 is not progressing A1 needs to reason about the compromise of allocating T1 to U3 instantly, or waiting for U2 (a more capable user) to finish.

Unfortunately, this task of monitoring involves a lot of transitional (progress made by users might not be the same in all time steps) and observational uncertainty (it may be difficult to monitor the exact progress made during a time step). However, agents can ask the human to make a decision when there is lot of uncertainty about the progress. Human assistance, on the other hand, involves a cost for disturbing the user. The user also might not always be present, i.e. the problem cannot be solved by just transferring control to user. It requires a sequence of decisions to be made. The model of the problem should then capture the sequential decision making in the presence of observational uncertainty. The necessity of a sequence of decisions can be seen in the TMP model because a users progress on a task may not be uniform. Some users may make most of their progress well before the deadline and others may do the bulk of their work closer to the deadline. Thus, an instantaneous assessment may not take into account the dynamics of progress. For example, consider an action space where an agent can observe P (progress) or NP (no progress) and can take three actions: W (wait), A (Ask user), R (reallocate). A sequential decision model could yield a policy tree as shown in Figure 1. A policy such as this takes into account both the uncertainty of observation but also includes the costs of decisions that will be made at later stages of the management problem in the analysis. In more complex domains with additional actions such as delaying the task deadline at some cost or choosing the appropriate user to whom a task will be reallocated, the cascading affects of these actions will require planning into time. POMDPs provide us the the framework to analyze and obtain policies in these domains, where simple rule-based strategies fail [17].

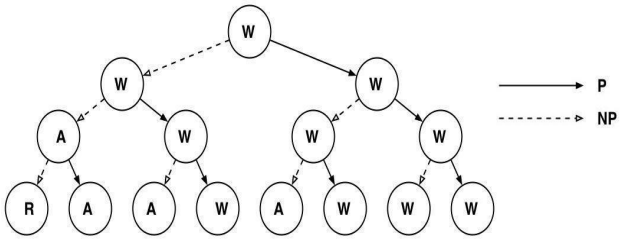

Figure 1. Sample POMDP TMP Policy

In the TMP domain, each state in the POMDP represents the progress of various tasks it is monitoring (its task and dependent tasks) and the time to the deadline. The policy provides a mapping from the observed progress of the task to the action. Employing a POMDP to solve a TMP problem can be very helpful. If this problem were modelled using an MDP, then the negligence of observational uncertainties could be disastrous in some circumstances. The following situations in the example of Figure 2 make the problem clear:

- A1 gets noisy information about U1 being complete with T1 and thus ignores a possible reallocation decision which might have been prudent.
- A1 gets noisy information about U1 not having started T1 and reallocates the task prematurely.

3. An Overview of POMDPs

A POMDP can be represented using the tuple $\{S, A, Tr, O, \Omega, R\}$, 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 is a probability distribution over the set of states S. A value function over a belief state is defined as:

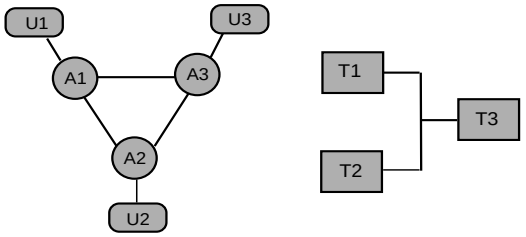

Figure 2. Communication Structure and Task Dependency Diagram

$V(b) = \max_{a \in A} \{R(b,a) + \beta \, \Sigma_{b' \in B} T(b,a,b') V(b')\}$. Currently, the most efficient exact algorithms for POMDPs are value iteration algorithms, specifically GIP [1] and RBIP [3]. 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:

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

Each $PRUNE$ call executes a linear program (LP) which is recognized as a computationally expensive phase in the generation of parsimonious sets in exact algorithm. Our approach is to obtain speedups by reducing the quantity of these calls.

4. Approaches

Our approach named Belief Support based Value Iteration (BSVI) consists of three enhancements to GIP. They include introducing ideas and algorithms that enable dynamic state (DS) spaces, dynamic observation (DO) sets, and dynamic belief (DB) supports. The key is the realization that for many domains for personal assistant agents interacting with users on tasks, the idea of progress implies properties of state transitions and observation generations that restrict states through time. We provide algorithms to extract these states and the corresponding viable algorithms. We also use these ideas to better calculate an accurate belief support over which to apply GIP. The theoretical bases, proofs of validity are provided below along with Algorithm 1 which details the implementation.

4.1. Dynamic State Spaces (DS)

A natural method for personal assistant agents to represent a user's state (such as in a 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 [12] for an adjustable autonomy problem addressed with MDPs. We note that in our domain due to the nature of how task progress and time evolve, one cannot reach all states from a given state. Consider a 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 the deadline $t \in \{1,2,3,4,5\}$. A static state space model would have states $S = [x,t]$. However, in our domain, time can only move forward in single steps, i.e. $T([x,t],[\tilde{x},\tilde{t}]) = 0$ if $\tilde{t} \neq t+1$. Thus a dynamic state model would have at time $\tilde{t}$ would only have states $S_{\tilde{t} \subset S}$ where $t = \tilde{t}$. Furthermore, if there are limits on how tasks progress, such as one cannot advance more than one progress level in one time step ($T([x,t],[\tilde{x},t+1]) = 0$ 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. This will require the transition matrix and reward function to be dynamic themselves. Dynamic state spaces, transition matrices and reward functions do not affect the dynamic programming process in a finite horizon problem because the value functions generated at a particular stage do not depend on the transition and reward functions of other states once the value function of the previous stage is known. 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. Line 7 in 'GEN-D-POMDP' function and the 'DP-UPDATE' function of Algorithm 1 provides the algorithm for DS. In this part of the algorithm, we describe how we use information from the transition matrix to extract the appropriate compact dynamic state spaces.

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

Algorithm 1 BSVI

Func POMDP-SOLVE $(L, S, A, T, \Omega, O, R)$

$(\{S_t\}, \{O_t\}, \{B_t^{max}\}) =$ GEN-D-POMDP $(L, S, A, T, \Omega, O, R)$
$t \leftarrow L; V_t \leftarrow 0$
for $t = L$ to 1 **do**
  $V_{t-1} =$ DP-UPDATE(V_t, t)

Func DP-UPDATE (V, t)

for all $a \in A$ **do**
  $V_{t-1}^a \leftarrow \phi$
  for all $\omega_t \in O_t$ **do**
    for all $v_t^i \in V$ **do**
      for all $s_{t-1} \in S_{t-1}$ **do**
        $v_{t-1}^{a,\omega_t,i}(s_{t-1}) = r_{t-1}(s_{t-1}, a)/|O_t| + \gamma \Sigma_{s_t \in S_t} Pr(\omega_t, s_t | s_{t-1}, a) v_t^i(s_t)$
    $V_{t-1}^{a,\omega_t} \leftarrow PRUNE(\{v_{t-1}^{a,\omega_t,i}\}, t)$
  $V_{t-1}^a \leftarrow PRUNE(V_{t-1}^a \oplus V_{t-1}^{a,\omega_t}, t)$
$V_{t-1} \leftarrow PRUNE(\bigcup_{a \in A} V_{t-1}^a, t)$
return V_{t-1}

Func POINT-DOMINATE(w, U, t)

for all $u \in U$ **do**
  if $w(s_t) \leq u(s_t), \forall s_t \in S_t$ then return true
return false

Func LP-DOMINATE(w, U, t)

LP vars: $d, b(s_t)[\forall s_t \in S_t]$
LP max d subject to:
  $b \cdot (w - u) \geq d, \forall u \in U$
  $\Sigma_{s_t \in S_t} b(s_t) \leftarrow 1$
  $b(s_t) <= b_t^{max}(s_t); b(s_t) >= 0$
if $d \geq 0$ return b else return nil

Func BEST(b, U)

$max \leftarrow Inf$
for all $u \in U$ **do**
  if $(b \cdot u > max)$ or $((b \cdot u = max)$ and $(u <_{lex} w))$ **then**
    $w \leftarrow u; max \leftarrow b \cdot u$
return w

Func PRUNE(U, t)

$W \leftarrow \phi$
while $U \neq \phi$
$u \leftarrow$ any element in U
if POINT-DOMINATE$(u, W, t) =$ true **then**
  $U \leftarrow U - u$
else
  $b \leftarrow$LP-DOMINATE(u, W, t)
  if $b = nil$ **then**
    $U \leftarrow U - u$
  else
    $w \leftarrow BEST(b, U)$
    $W \leftarrow W \bigcup w$
    $U \leftarrow U - w$
return W

Func GET-BOUND(s_t)

max $\leftarrow 0$
$* c_1, c_2$ pair for each Action and Observation $*$
for all c_1, c_2 **do**
  maxVal $\leftarrow \max_{i \leq |s_t|} c1[i]/c2[i])$
  if maxVal $>$ max **then**
    max $\leftarrow$ maxVal
return max

Func GEN-DPOMDP(T, S, A, Tr, O, Ob, R)

$t \leftarrow 1$
$S_t =$Set of starting states
for all $s_t \in S_t$ **do**
  $b_t^{max}(s_t) = 1$
for $t = 1$ to $T - 1$ **do**
  for all $s \in S_t$ **do**
    ADD-TO(S_{t+1},REACHABLE-STATES(s, Tr))
    $O_{t+1} =$ GET-RELEVANT-OBS(S_{t+1}, Ob)
    $b_{t+1}^{max}(s_{t+1}) =$GET-BOUND(s_{t+1})
return $(\{S_t\}, \{O_t\}, \{b_t^{max}\})$

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:

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

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.

Modeling the process in this manner does not effect the optimality of the solution obtained using value function methods. Let $V_t(b_t)$ be the value of the optimal policy at time t (or equivalently at the $L-t+1$-th stage of dynamic programming). 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 $V_{L-1}(b_{L-1}) = \max_{p \in P_{L-1}} \tilde{b}_{L-1} \cdot \tilde{\alpha}^p_{L-1}$ where $|\tilde{b}_{L-1}| = |\tilde{\alpha}^p_{L-1}| = |S_{L-1}|$ and $\tilde{\alpha}^p_{L-1} = [V^p_L(\tilde{s}_1) \cdots V^p_L(\tilde{s}_{|S_{L-1}|})]$ for $\tilde{s}_i \in S_{L-1}$. Applying this reasoning inductively, we can show that we only need $V^p_t(s)$ for $s \in S_t$. Furthermore, if $s \in S_t$, then

$$V^p_t(s) = R_t(s, a(p)) + \gamma \sum_{s' \in S_{t+1}} T_t(s, a(p), s') \sum_{\omega \in \Omega} O(s', a, \omega) V^{p_\omega}_{t+1}(s') ----(I).$$

Thus, we only need $\{V^{\omega(p)}_{t+1}(s') : s' \in S_{t+1}\}$. The value functions 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 unreachable, which further reduces the possible policies at the previous time.

4.2. Dynamic State and Observation Spaces(DS+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 type will also be dynamic. Consider a situation where there are five progress levels: $S := \{0.00, 0.25, 0.50, 0.75.1.00\}$ and five observations $\Omega := \{0.00, 0.25, 0.50, 0.75.1.00\}$. Furthermore, let us assume first, that one can move up at most one progress level in one stage (e.g. you cannot go to 0.75 or 1.00 from 0.25 in one step) and second, if one has reached a state $s \in S$ by taking some action, the only viable observations are progress levels at or below the current level of progress (e.g. if you are at 0.50, you can only get observations 0.00, 0.25, or 0.50). Now, if we assume that the dynamic state space for a particular stage 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 that they do not affect the value iteration process. Line 8 in 'GEN-D-POMDP' function and the 'DP-UPDATE' function of Algorithm 1 provides the algorithm for DS+DO.

Let Ω be the set of all possible observations. Let us define $\Omega_t := \{\omega \in \Omega : \exists a \in A, s \in S_{t+1} \text{ s.t. } O(s', a, \omega) > 0\}$. Given (I) from 4.1, we can rewrite $V^p_t(s_t)$ as

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

where Ω^C_t 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, subtrees p_ω are not necessary if $\omega \notin \Omega_t$. This further reduces the set of policies being generated before pruning. This improves our performance by reducing the number of vectors that need to be considered by the linear program. For consistency, we now index the observation probability matrix with time as it depends on a dynamic state.

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 incorporating information about the initial belief distribution, the transition and observation probabilities. For example, if we know that our initial belief regarding task progress can have at most 0.10 probability of being one quarter done with the rest of the probability mass on being not started, what is the highest probability of being one quarter or one half done at the next stage, given a dynamic transition matrix? Below we outline a polynomial-time procedure by which we can obtain bounds on belief support which answer that question. Line 9 of "GEN-DPOMDP" function of Algorithm 1 provides the algorithm for DB.

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

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

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

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

then we have $P(b_{t+1} \notin B_{t+1}) = 0$. We now show how $b^{\max}_{t+1}(s_{t+1})$ (and similarly $b^{\min}_{t+1}(s_{t+1})$) can be generated through linear programming. Given an action a and observation ω, we can express the problem as

$$\max \frac{b_t \cdot c_1}{b_t \cdot c_2} - - - - - (1)$$

where $c_1(s) = O_t(s_{t+1}, a, \omega)T_t(s_t, a, s_{t+1})$ and $c_2(s) = \sum_{s_{t+1} \in S_{t+1}} O_t(s_{t+1}, a, \omega)T_t(s_t, a, s_{t+1})$.

It can be shown that the $\max_i c_1[i]/c_2[i]$ is an upper bound for the expression in (1). This proof is done using mathematical induction on the number of variables n.However, to obtain $b^{\max}_{t+1}(s_{t+1})$, we further optimize $b^{a,o}_{t+1}(s_{t+1})$ over each A and O_t combination. The set B_1 must be chosen such that it can be modeled with a set of linear constraints over b_1. The sets$\{B_t\}^L_{t=2}$ can be represented with linear constraints using $b^{\min}_t$ and $b^{\max}_t$. By using dynamic beliefs, we increase the costs of the LP 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 TMP domain explained in Section 2. As explained earlier, each agent uses a POMDP to reason about reallocation and transfer of control to humans. Complexity of these POMDPs can be increased/decreased by increasing/decreasing the number of progress steps (for tasks) and/or monitoring steps(deadline). The criterion used for comparing various approaches (GIP, DS, DS+DO, DS + DO + DB) are time taken, vectors before and after pruning. Vectors before pruning is an important criterion for experiments involving DS+DO, since the main contribution of DS+DO is in reducing the number of vectors before pruning at each iteration. Number of vectors after pruning serves as an indicator of the belief space planned for. If one approach gives more vectors after pruning than another approach in finding an equal quality solution, then it implies that the former is planning for some unreachable belief space. DS, and DS+DO are fully implemented, while DB is only partially implemented and hence only initial results with DS+DO+DB. Our experimental setup consisted of 10 problems (A..J) in order of increasing complexity.

Table 1 provides time results for GIP, DS, and DS + DO. "–" indicates that the problem did not provide results even for the initial decision epochs, within the specified time limit. As can be seen from the table DS provides orders of magnitude speedup over GIP.

Experiment	GIP	DS	DS+DO
A	0.75	0.02	0.02
B	10000	0.04	0.04
C	10000	0.06	0.06
D	10000	0.07	0.07
E	–	34.4	45.17
F	–	55.62	121.22
G	–	56.16	124.34
H	–	182.84	283.9
I	–	239.34	417.77
J	–	2883.37	4335.63

Table 1. Comparing time taken (in seconds)

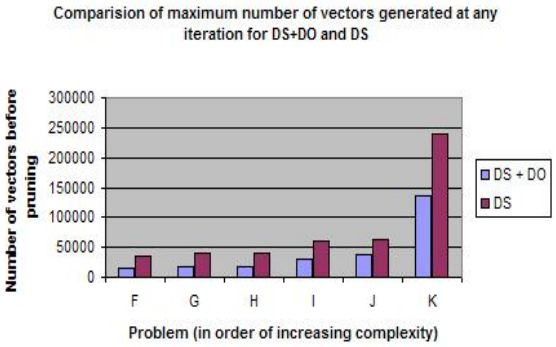

Figure 3. Comparison of DS+DO with DS

Algorithm	Time Taken	After Pruning	Before Pruning
DS + DO + DB	0.29	71	1136
DS + DO	3.89	382	6112
DS	6.33	382	18366

Table 2. Illustrating advantages of DB

As can be seen DS+DO takes less time when compared to DS, with the difference becoming more apparent as the problem complexity increases. Fig.3 compares number of maximum number of α-vectors generated before pruning in DS+DO, to DS. As can be seen, number of vectors before pruning in DS+DO is far less than the number in DS. Table 2 provides results comparing performance of DS+DO+DB to DS+DO and DB in a specific problem. It can be seen that for all the criterion, DS+DO+DB outperforms DS+DO and DS. The number of vectors after pruning in DS+DO+DB gives an indication of the unreachable belief region removed by putting better bounds on belief probabilities of states.

6. Related Work and Conclusions

Techniques for solving POMDPs can be categorized as exact and approximate. GIP [1] and RBIP [3] are the existing exact algorithms complementary to our work. 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 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 state POMDPs; and (iii) extent of speedup provided is not measured. This contrasts with this work, which accounts for dynamic changes in belief space reachability and applies to both flat and factored state POMDPs.

Approximate algorithms are faster than exact algorithms, but at the cost of solution quality. There has been significant work in this area, but point-based [6, 14], and grid-based [4, 15] techniques dominate others. It is critical to have quality guarantees in Personal Assistant domains, for an agent to gain the trust of a human user. One other technique uses state space dimensionality reduction using E-PCA, but it does not provide any guarantee on quality of the solution [13]. Point Based Value Iteration (PBVI) [6] provides good quality guarantees, but to obtain good results it needs to increase sampling, consequently increasing the runtime. Nonetheless, our techniques can benefit approximate algorithms, e.g., PBVI can benefit from DO.

This paper provides techniques to make application of POMDPs in personal assistant agents a reality. In particular, we provide three key techniques that exploit the notion of progress or dynamics in PAA domains to speedup POMDP policy generation. The key insight is that given an initial (possibly uncertain) starting set of states, the agent needs to be prepared to act only in a limited range of belief states. All other belief states are unreachable given the dynamics of the monitored process, and no policy needs is requiredfor such 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 Generalized Incremental Pruning (GIP), one of the 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 facilitate agents' utilizing POMDPs when assisting humans.

Acknowledgements. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

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] Y. Arens D. Pynadath, M. Tambe and H. Chalupsky. Electric elves: Immersing an agent organization in a human organization. In *AAAI Fall Symposium on Socially Intelligent Agents — the human in the loop*, 2000.

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

[4] M. Hauskrecht. Value-function approximations for partially observable markov decision processes. *JAIR*, 13:33–94, 2000.

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

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

[7] T. Y. Leong and C. Cao. Modeling medical decisions in dynamol: A new general framework of dynamic decision analysis. In *MEDINFO*, pages 483–487, 1998.

[8] W. S. Lovejoy. Computationally feasible bounds for partially observable markov decison processes. *Operations Research*, 39:175–192, 1991.

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

[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] 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.

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

[13] N. Roy and G. Gordon. Exponential family pca for belief compression in pomdps. In *NIPS*, pages 707–716, 2002.

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

[15] R. Zhou and E. Hansen. An improved grid-based approximation algorithm for pomdps. In *IJCAI*, pages 707–716, 2001.