Implementation Techniques for Solving POMDPs in Personal Assistant Domains

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 two key implementation techniques (one exact and one approximate), where the policy computation is restricted to the belief space polytope that remains reachable given the progress structure of a domain. One technique uses Lagrangian methods to compute tighter bounds on belief space support in polynomial time, while the other technique is based on approximating policy vectors in dense policy regions of the bounded belief polytope. 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 implementation techniques in facilitating the deployment of POMDPs within agents assisting human users.

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 [4,6,8,9,12,13]. Such agents must often monitor the evolution of some 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 [12]. Similarly, in assisting with caring for the elderly [9] and therapy planning [6, 8], 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) [6, 8, 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. Recent work [9] reports using POMDPs for robotic assistants: we complement that work with novel techniques potentially enabling significant scale-up.

Recognizing the run-time barrier to POMDP usage, previous work on POMDPs has made encouraging progress using two approaches. The first is an exact approach, where one tries to find the optimal solution [1, 2]. However, despite advances, exact algorithms remain computationally expensive and currently do not scale to problems of interest in PAA domains. The second is an approximate approach, where one sacrifices solution quality for speed [3,5,14,15]. Unfortunately, current approximate algorithms often provide loose (or no) quality guarantees on the solutions, even though such guarantees are crucial for PAAs to inhabit human environments.

Earlier work in programming multi agents has used BDI type frameworks, however this paper aims to practically apply POMDPs to PAA domains by introducing novel implementation techniques that are particularly suitable for such settings. One key insight is that when monitoring users or processes over time, large but shifting parts of the belief space in POMDPs (i.e., regions or states of uncertainty) remain unreachable. Thus, we can focus policy computation on the reachable belief-space polytope, which changes dynamically due to progress in the domain. For instance, consider a PAA monitoring a user driving to a meeting. Given knowledge of the user's current location, the reachable belief region is bounded by the maximum probability of the user's 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 [7, 11]. Another key contribution of this paper is an approximation technique that discretizes the value space, considering value vectors which are seperated by ϵ (approximation parameter) as a single vector. As shown in later sections, this method provides an error bound, which depends on the exact structure of the value function, rather than depending on a bound for expected value. This fact in itself can help provide tighter bounds. We enhance two state-of-the-art exact POMDP algorithms [1, 2] delivering over an order of magnitude speedup for two different PAA domains.

2 Motivating PAA Domains

We present two motivating examples, where teams of software PAAs are deployed in office environments to assist human users [4,12]. The first is a meeting rescheduling problem (MRP), 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 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 [4]. 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). Agents (e.g. A1, A2, A3 in Figure 1) 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 reallocated to U3 despite the potential loss in capability. POMDPs provide a

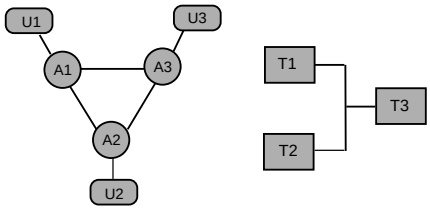

Fig. 1. Comm. Structure and Task Dependency

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 the deadline $t \in \{1, 2, 3, 4, 5\}$. Observations are the 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 of observations and future costs of decisions, and maps observations to actions, for the above scenario is shown in Figure 2 (nodes=actions, links=observations). In more complex domains with additional actions such as delaying deadlines, the cascading effects of actions will require even more careful planning afforded by POMDP policy generation. Such scenarios in TMP and MRPs are investigated and discussed in Section 5.

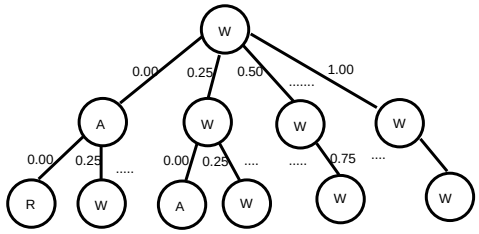

Fig. 2. Partial Sample Policy for a TMP

3 POMDPs and 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 [2]. 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 [2]):

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. $\mathcal{V}_{t-1}^{a} = PRUNE(\cdots(PRUNE(\mathcal{V}_{t-1}^{a,o_1} \oplus \mathcal{V}_{t-1}^{a,o_2})\cdots \oplus \mathcal{V}_{t-1}^{a,o_{|\Omega|}})$
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 in exact algorithm [1, 2]. Our approach effectively translates into obtaining speedups by reducing the quantity of these calls.

4 Dynamic Belief Supports

We propose two new implementation techniques for solving POMDPs in PAA domains: (i) dynamic belief supports (DB); and (ii) expected value approximation (EVA). These ideas may be used to enhance existing POMDP algorithms such as GIP and RBIP. The key intuition in DB is that for PAA 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 with (i) are due to the elimination of policies dominant in regions outside this polytope, which reduces the number of LP calls. On the other hand, EVA exploits the density of policy vectors in the belief polytope calculated using DB. EVA works by using a lesser density set to represent the optimal set, thus sacrificing on quality of the solution.

4.1 Dynamic Belief Spaces (DB)

Before introducing the general belief support technique, we introduce a special case of it called as DBSimple. In this only states that are reachable (given the transitional dynamics) at each epoch are considered. This is a special case of the general belief restriction in that the belief support is bounded by only 0.00, rather than any number less than 1.00. By introducing DBSimple, 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 0.00 or 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 such that by applying the standard belief update rule, one would get a belief vector b_t not

Algorithm 1 `DB + GIP`

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

$(\{S_t\}, \{O_t\}, \{B_t^{max}\})$ = DSDODB-GIP $(L, S, A, T, \Omega, O, R)$
$t \leftarrow L$;$V_t \leftarrow 0$
for $t = L$ to 1 **do**
  $\mathcal{V}_{t-1}$ = DP-UPDATE$(\mathcal{V}_t, t)$

Func DP-UPDATE $(\mathcal{V}, t)$

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

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 DB-GIP$(L, S, A, T, \Omega, O, R)$

$t \leftarrow 1$; $S_t =$Set of starting states
for all $s_t \in S_t$ **do**
  $b_t^{max}(s_t) = 1$
for $t = 1$ to $L - 1$ **do**
  for all $s \in S_t$ **do**
    ADD-TO(S_{t+1},REACHABLE-STATES(s, T))
    Ω_{t+1} = GET-RELEVANT-OBS(S_{t+1}, O)
    C = GET-CONSTRAINTS (s_t)
    $b_{t+1}^{max}(s_{t+1})$ = MAX$_{c \in C}$(GET-BOUND(s_{t+1}, c))
return $(\{S_t\}, \{\Omega_t\}, \{b_t^{max}\})$

Func GET-BOUND($s_t, constraint$)

```
y_min = MIN_{s∈S_{t-1}}(constraint.c[s]/constraint.d[s])
y_max = MAX_{s∈S_{t-1}}(constraint.c[s]/constraint.d[s])
INT = GET-INTERSECT-SORTED(constraint, y_min, y_max)
for all i ∈ INT do
    Z = SORT(((i + ϵ) * constraint.d[s] − constraint.c[s]),∀s ∈ S_{t-1}
    sumBound = 1, numer = 0, denom = 0
    /* IN ASCENDING ORDER */
    for all z ∈ Z do
        s = FIND-CORRESPONDING-STATE(z)
        if sumBound − bound[s_{t-1}] > 0 then
            sumBound− = bound[s_{t-1}]
            numer+ = bound[s_{t-1}] * constraint.c[s_{t-1}]
            denom+ = bound[s_{t-1}] * constraint.d[s_{t-1}]
        if sumBound − bound[s_{t-1}] <= 0 then
            numer+ = sumBound * constraint.c[s_{t-1}]
            denom+ = sumBound * constraint.d[s_{t-1}]
            BREAK-FOR
    if numer/denom > i and numer/denom < max then
        return numer/denom
```

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 we have the belief polytope

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

then we have $P(b_{t+1} \notin B_{t+1}) = 0$. The proof of optimality preservation for dynamic beliefs is omitted due to lack of space.

We now show how $b^{\max}_{t+1}(s_{t+1})$ (and similarly $b^{\min}_{t+1}(s_{t+1})$) can be generated through a polynomial-time procedure deduced from Lagrangian methods. The method involves iterating over all a and ω, where for a given action a and observation ω, we can express the problem as

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

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

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

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

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

Because λ is identical in all three conditions and $\bar{\mu}_k$ and μ_k are non-negative for all k, the component x_k associated with the lowest value of $[(c^T x)/(d^T x)]d_k - c_k$ 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 $z_k = yd_k - c_k$, 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 the component not already at its bound with the lowest z_k.

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 $(|s_t| - 1)|s_t|/2$ intersections points. We can then partition the support $[y_{\min}, y_{\max}]$ into disjoint intervals using these intersection points yielding at most $(|s_t| - 1)|s_t|/2 + 1$ 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 4.1. 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 8 and 9 of the DSDODB-GIP function in Algorithm 1 provide the procedure for DB. The GET-CONSTRAINTS function on Line 8 gives the set of c and d vectors for each state at time t for each action and observation. By using dynamic beliefs, we increase the costs of pruning by adding some constraints on maximum probability $b^{max}(s_t)$ as shown in line 5 of LP-dominate. However, there is an overall gain because we are looking for dominant

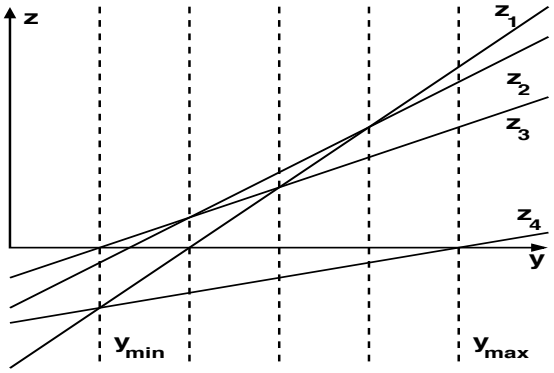

Fig. 3. Partition Procedure for Solving Belief Maximization Lagrangian

vectors over a smaller belief polytope. Thus, reducing the cardinality of the parsimonious set, leaving fewer vectors/policies to consider at the next iteration.

4.2 Expected Value Approximation (EVA)

Expected Value Approximation (EVA) is an approximate approach for solving POMDPs. Most of the approximate algorithms for solving POMDPs [5, 15] discretize the belief space to obtain, however here we provide an algorithm that discretizes the expected value space. As is known from the literature, the value function in a POMDP can be expressed using a finite set of linear vectors. EVA approximates the parsimonious set of the linear vectors (in the "CUP") using lesser number of vectors given the approximation parameter α, which indicates the maximumm error allowed in expected value at any belief point.

Algorithm 2 LP-DOMINATE(w, U, t, ϵ)

solve the following linear program
variables: $d, b(s_t)[\forall s_t \in S_t]$
maximize d
subject to the constraints
    $b \cdot (w - u) \geq d + \epsilon, \forall u \in U$
    $\Sigma_{s_t \in S_t} b(s_t) \leftarrow 1$
    $b(s_t) <= b_t^{max}(s_t)$
    $b(s_t) >= 0$
if $d \geq 0$ **then**
    return b
else
    return nil

Algorithm 2 provides the procedure used for checking whether a vector is dominated by a set of vectors in algorithms such as GIP and RBIP. EVA uses the same procedure except for $d + \epsilon$ instead of d in RHS of line 5. This extra ϵ (in line 5) implies that for a vector to dominate a set of vectors, it should dominate each of the vectors by at least ϵ. That is to say, all vectors which don't dominate all the vectors in the set by at least ϵ are pruned out, hence decreasing the size of the parsimonious set. Savings provided by EVA are in the number of vectors in the parsimonious set (vectors after pruning) at each epoch. Reduced number of vectors after pruning has a chain effect, since it leads to less number of projections (or vectors before pruning) at the next epoch, which in turn might lead to reduced number of vectors after pruning in that epoch.

The main difference between some of the existing methods (like the point based or grid based approaches) and EVA is the space in which approximation is done. In point-based or grid-based, the approximation is in the belief space, while in our approach it is in the value space. EVA can provide better bounds because it is based on value space based approximation that approximates based on the exact structure of the value function rather than take worst case bounds on the value function. This is studied extensively in [10].

Proposition 1. *Error of the EVA algorithm can be bounded by* $2 * \epsilon * |\Omega|$ *for GIP type cross sum pruning.*

Proof. EVA algorithm introduces an error whenever a pruning operation is performed. Since there are three stages where pruning operations are performed, this proof proceeds by summing the error introduced at each of these stages.

1. $\mathcal{V}^{a,o} = PRUNE(\mathcal{V}^{a,o,i})$
   After this pruning step, each of $\mathcal{V}^{a,o}$'s $(\forall a, \forall o)$ are away from the optimal by at-most ϵ.
2. $\mathcal{V}^{a} = PRUNE(\cdots(PRUNE(\mathcal{V}^{a,o_1} \oplus \mathcal{V}^{a,o_2}) \cdots \oplus \mathcal{V}^{a,o_{|\Omega|}})$ To calculate the error bound after this pruning step, we start from the innermost cross-sum PRUNE. The innermost prune would give a set of vectors which in the worst case is $\epsilon + \epsilon + \epsilon$ away from the optimal set. In the above bound, first and second epsilon follow from the fact that there is a cross sum and that each term is away from optimal by ϵ in the worst case, while the third epsilon is because of the PRUNE on this cross sum. Each subsequent prune adds a further $2 * \epsilon$ to the bound. Thus each $\mathcal{V}^{a,o}$ is away from the optimal by at-most $2 * \epsilon * (|\Omega| - 1) + \epsilon$.
3. $\mathcal{V}' = PRUNE(\bigcup_{a \in A} \mathcal{V}^{a})$ Now since this step does a PRUNE over UNION of $\mathcal{V}^{a}, \forall a$, it further adds an ϵ to the bound. Hence making the final error bound to be $2 * \epsilon * |\Omega|$.

Thus proved. ■

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 enhance-

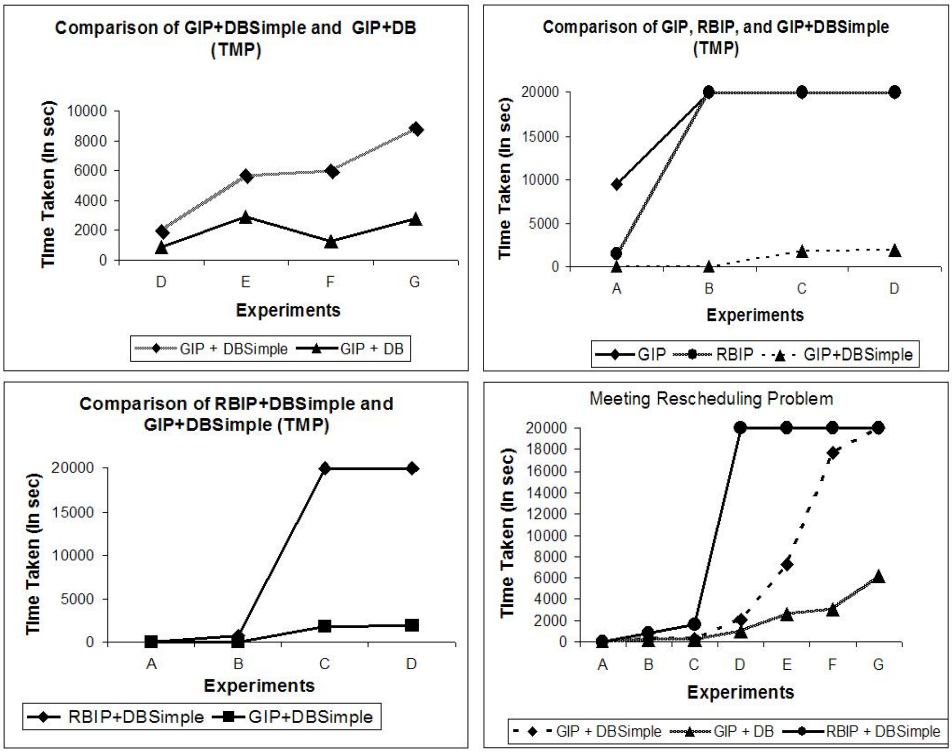

Fig. 4. TMP: (a) DBSimple+GIP gives orders of magnitude speedup over GIP and RBIP (b) DB+GIP dominates DBSimple+GIP (c) DBSimple+GIP dominates DB-Simple+RBIP; MRP: (d) DB+GIP dominates

ments, DBSimple (Dynamic States), and DB (Dynamic Beliefs), were implemented over both GIP and RBIP [2] (RBIP is itself a recent enhancement to GIP). All the experiments compare the performance (run-time) of GIP, RBIP and our enhancements over GIP and RBIP. For both domains, we ran 6 problems over all methods (GIP, RBIP, DBSimple+GIP, DB+GIP, DB+GIP, DB-Simple+RBIP, DB+RBIP). Each problem had pre-specified upper limit of 20000 seconds, after which it was terminated.

Figure 4(a)-(c) present results for the TMP domain. Experimental setup in TMP consisted of a set 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 in only Problem A, as shown in Figure 4(a). DBSimple+GIP provides 100-fold speedup in Problem B, and 10-fold speedup in Problems C and D (however, the actual speedup which we expect to be even larger cannot be seen due to our cutoff).

DB+GIP finished in almost the same time as DS in Problems A-C. Figure 4(b) provides comparisons between the three of our enhancements on GIP. For Problems D-G that are even more complex than A-C, DB dominates the other enhancements providing approximately 5-fold speedup over DBSimple. GIP and RBIP did not terminate within time limit and hence not shown. The key point of Figure 4(c) is to show that DBSimple+GIP provides 10-fold speedup (with cut-off) over DBSimple+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 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 7 problems. DB+GIP provides approximately 6-fold speedups over DBSimple+GIP. DBSimple+RBIP seems comparable with the other three methods in Problems A-C, but for Problems D-G, it fails to even finish before the cutoff. Both domains provide similar conclusions: DB+GIP dominates other techniques (with around 100 fold speedup over GIP and RBIP in some cases) and this dominance becomes more significant in larger problems.

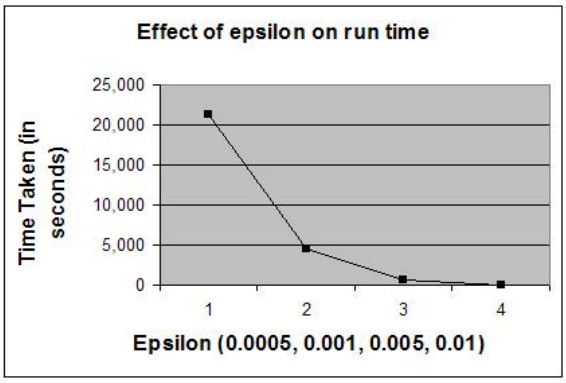

Fig. 5. Effect of epsilon on run times

Figure 5 presents results for the EVA approximation algorithm. x-axis shows different values of ϵ, approximation parameter and y-axis shows the run times. To clearly show the capacity of EVA, we present these results on a bigger problem than A-G. As can be seen, EVA provides orders of magnitude speedup as ϵ is decreased from 0.0001 to 0.01. As can be seen from these results, even though there was orders of magnitude speedup, the error bound was only 0.72 (optimal solution was 9.5).

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 [2] are exact algorithms, which we have enhanced. Other exact algorithms attempt to exploit domain-specific properties to speedup POMDPs. For instance, [7] 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 [5, 14], grid [3, 15], and policy search approaches [**?**, **?**] 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 good quality guarantees in PAA domains, for an agent to gain the trust of a human user. Another recently developed technique uses state space dimensionality reduction using E-PCA, but it does not provide any guarantee on quality of the solution [11]. Point Based Value Iteration (PBVI) [5] provides the best quality guarantees, but to obtain good results it needs to increase sampling, consequently increasing the run-time. As explained earlier, EVA approach can provide tighter bounds because of its approximation in the expected value space.

7 Summary

Typically for programming multi-agent systems, BDI frameworks have been used. However in this paper, we provide novel implementation techniques to make the application of POMDPs a reality. In particular, we provide two key techniques to speedup POMDP policy generation that exploit the key properties of the PAA domains. One 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. 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. Another key insight is to exploit the high density of value vectors, to speedup policy generation, while sacrificing very little in terms of the quality of solution. We provide a detailed algorithm illustrating our enhancements in Algorithm 1, and present proofs of correctness of our techniques. The techniques presented here facilitate agents' utilizing POMDPs for policies when assisting human users.

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. Z. Feng and S. Zilberstein. Region based incremental pruning for POMDPs. In *UAI*, 2004.
3. M. Hauskrecht. Value-function approximations for POMDPs. *JAIR*, 13:33–94, 2000.
4. http://www.ai.sri.com/project/CALO, http://calo.sri.com. *CALO: Cognitive Agent that Learns and Organizes*, 2003.
5. G. Gordon J. Pineau and S. Thrun. PBVI: An anytime algorithm for POMDPs. In *IJCAI*, 2003.
6. 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.
7. H. Fraser M. Hauskrecht. Planning treatment of ischemic heart disease with partially observable markov decision processes. *AI in Medicine*, 18:221–244, 2000.
8. 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.
9. 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.
10. P. Poulpart and Craig Boutilier. Bounded finite state controllers. In *NIPS*, 2003.
11. N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *NIPS*, 2002.
12. P. Scerri, D. Pynadath, and M. Tambe. Towards adjustable autonomy for the real-world. *JAIR*, 17:171–228, 2002.
13. D. Schreckenghost, C. Martin, P. Bonasso, D. Kortenkamp, T.Milam, and C.Thronesbery. Supporting group interaction among humans and autonomous agents. In *AAAI*, 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*, 2001.