Winning back the CUP for distributed POMDPs: Planning over continuous belief spaces

Pradeep Varakantham, Ranjit Nair*, Milind Tambe, Makoto Yokoo**
University of Southern California, Los Angeles, CA 90089, {varakant, tambe}@usc.edu
* Automation and Control Solutions, Honeywell Laboratories, Minneapolis, MN 55418, ranjit.nair@honeywell.com
** Dept. of Intelligent Systems, Kyushu University, Fukuoka, 812-8581 Japan, yokoo@is.kyushu-u.ac.jp

ABSTRACT

Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are evolving as a popular approach for modeling multiagent systems, and many different algorithms have been proposed to obtain locally or globally optimal policies. Unfortunately, most of these algorithms have either been explicitly designed or experimentally evaluated assuming knowledge of a starting belief point, an assumption that often does not hold in complex, uncertain domains. Instead, in such domains, it is important for agents to explicitly plan over continuous belief spaces. This paper provides a novel algorithm to explicitly compute finite horizon policies over continuous belief spaces, without restricting the space of policies. By marrying an efficient single-agent POMDP solver with a heuristic distributed POMDP policy-generation algorithm, locally optimal joint policies are obtained, each of which dominates within a different part of the belief region. We provide heuristics that significantly improve the efficiency of the resulting algorithm and provide detailed experimental results. To the best of our knowledge, these are the first run-time results for analytically generating policies over continuous belief spaces in distributed POMDPs.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence - Multi Agent Systems

General Terms

Algorithms, Theory

Keywords

Multi-agent systems, Continuous initial beliefs, Partially Observable Markov Decision Process (POMDP), Distributed POMDP

1. INTRODUCTION

Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are evolving as a popular approach for modeling multiagent teamwork [13, 18, 14, 6, 3]. Distributed POMDPs provide a particularly useful model for domains ranging from sensor nets to disaster rescue, where agents need to work in a team in the presence of domain costs and uncertainties. Since optimal policy computation for these models has been proven to be NEXP-Complete [3], researchers have pursued two approaches in building algorithms for distributed POMDPs. The first approach restricts the domain, limiting agents' interactions (transition independence) [2, 14] or approximating observability of the local state [6]. The second approach, in-line with the thrust of this work, does not restrict the domains, but finds local optimal solutions [11, 7, 16].

Unfortunately, these approaches often assume knowledge of the starting belief state when generating policies[16, 11, 14, 10]. However, in many domains, including sensor nets and disaster rescue, such an assumption is a significant restriction, since it requires that starting belief state be known in advance. Alternatively, policy computation must be done on-line, which, given significant runtime for generating policies for distributed POMDPs, would prevent agents from generating a high-quality response in real-time.

Thus, there is a need in distributed POMDPs to generate policies for continuous belief spaces rather than just an initial single belief point, so policies may be computed off-line or without the need for a known starting belief state. This would allow agents to respond in real-time in time-critical domains. Unfortunately, while some of the approaches for locally optimal policy generation in distributed POMDPs can be applied or directly extended to apply to policy generation over continuous belief spaces[16, 7], these are based on finite-state controllers, which restrict policy representations; a restriction that is even more significant when planning over continuous belief spaces. Alternatively, a globally optimal policy generation algorithm has been suggested for continuous spaces[8], but this approach must surmount the substantial computational barrier given the NEXP-complete complexity result. In fact, no experimental evaluations are currently available for run-time efficiency when generating policies over continuous belief spaces with either of these approaches.

To remedy this situation, we present a novel algorithm for locally optimal policy generation for continuous starting belief region that does not restrict the space of policies considered. In particular, we build on the "Joint Equilibrium-based Search for Policies" (JESP) algorithm[11] which finds locally optimal policies from an unrestricted set of possible policies, with a finite planning horizon. However, JESP currently assumes a known starting belief state. To enable planning over continuous belief spaces, we combine JESP with algorithms that plan optimal policies over continuous belief spaces for single agent POMDPs [1, 5, 15]. In particular, whereas the original JESP performed iterative best-response computations from a single starting belief state, the combined algorithm exploits the single-agent POMDP techniques to perform best-response computations over continuous regions of the belief space.

The paper thus presents a new algorithm, CS-JESP (Continuous Space JESP), that allows us to generate a piece-wise linear and convex value function over continuous belief spaces for the optimal policy of one agent in the distributed POMDP, given fixed policies of other agents — the familiar cup-like shape of this value function is what we reference in the title of this paper [9]. The cup-shape implies that when dealing with a continuous starting belief space, agents usually have more than one policy, each of which dominates in a different region of the belief space. This region-wise dominance highlights the three important challenges addressed in CS-JESP. First, CS-JESP requires computation of best response policies for one agent, given that different policies dominate over different regions of the belief space for the second agent. To efficiently compute best response policies per belief region, it is critical to employ techniques that prune out unreachable future belief states. To that end, we illustrate application of the belief bound techniques[15] for improved efficiency. Second, owing to these best response calculations for different belief regions, often the policies for contiguous belief regions can be identical. To address this inefficiency, we implement a merging method that combines such adjacent regions with equivalent policies. Third, to improve the performance of the algorithm, we implement region-based convergence, i.e. once policies have converged for a region, these are not considered for subsequent best response computations.

2. BACKGROUND

As mentioned earlier, this paper presents a solution that combines value iteration algorithms of single agent POMDPs and JESP. In this section, we present a brief overview of these techniques.

2.1 Single Agent POMDPs and DB-GIP

A single-agent POMDP can be represented using the tuple $\left\langle\hat{\mathcal{S}},\hat{\mathcal{A}},\hat{\mathcal{P}},\hat{\Omega},\hat{\mathcal{O}},\hat{\mathcal{R}}\right\rangle$, where $\hat{\mathcal{S}}$ is a finite set of states; $\hat{\mathcal{A}}$ is a finite set of actions; $\hat{\Omega}$ is a finite set of observations; $\hat{\mathcal{P}}(s,a,s')$ provides the probability of transitioning from state s to s' when taking action a; $\hat{\mathcal{O}}(s',a,o)$ is probability of observing o after taking an action a and reaching s'; $\hat{\mathcal{R}}(s,a)$ is the reward function. We use, $\hat{\mathcal{B}}$ to denote the feasible belief space for a POMDP. A belief point $b\in\hat{\mathcal{B}}$, is a probability distribution over the set of states $\hat{\mathcal{S}}$. A value function over a belief state is defined as:
$V(b) = \max_{a\in\hat{\mathcal{A}}}\left\{\hat{\mathcal{R}}(b,a)+\gamma\,\Sigma_{b'\in\hat{\mathcal{B}}}\Pr(b'|b,a)V(b')\right\}$. Currently, the most efficient exact algorithms for POMDPs are value iteration algorithms, specifically GIP [1] and RBIP [5]. 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 that in [1] and [5]):

1. $\left\{v^{a,o,i}_{t-1}(s)=\hat{\mathcal{R}}(s,a)/|\hat{\Omega}|+\gamma\,\Sigma_{s'\in\hat{\mathcal{S}}}\Pr(o,s'|s,a)v^i_t(s')\right\}=\hat{\mathcal{V}}^{a,o}_{t-1}$ where $v^i_t\in\mathcal{V}_t$.
2. $\mathcal{V}^{a,o}_{t-1}=PRUNE(\hat{\mathcal{V}}^{a,o}_{t-1})$
3. Doing crosssum ($\oplus$) of vectors .
$\mathcal{V}^{a}_{t-1}=PRUNE(\cdots(PRUNE(\mathcal{V}^{a,o_1}_{t-1}\oplus\mathcal{V}^{a,o_2}_{t-1})\cdots\oplus\mathcal{V}^{a,o_{|\hat{\Omega}|}}_{t-1})$
4. $\mathcal{V}_{t-1}=PRUNE(\bigcup_{a\in\hat{\mathcal{A}}}\mathcal{V}^{a}_{t-1})$

Each $PRUNE$ call executes a linear program (LP) which is recognized as a computationally expensive phase in the generation of parsimonious sets [1, 5]. As shown in [15], the performance of these dynamic programming algorithms can be improved considerably by exploiting the dynamics of the domain.

DB-GIP: This is an exact (not approximate) technique that builds on top of GIP, to provide significant performance improvements. There are two key ideas in DB-GIP (we focus on DB-GIP as it is shown to be more efficient than DB-RBIP)[15]. The first technique called DB, prunes unreachable belief spaces, given information about the starting belief space, B_0. To that end, it computes belief bounds over states at future time steps given B_0. These belief bounds are obtained by solving the maximization and minimization problems on the equation for belief probability as follows. Given an action a and observation ω, we can express the maximization problem as

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

where $c(s_t)=\hat{\mathcal{O}}_t(s_{t+1},a,\omega)\hat{\mathcal{P}}_t(s_t,a,s_{t+1})$ and
$d(s_t)=\sum_{s_{t+1}\in\hat{\mathcal{S}}_{t+1}}\hat{\mathcal{O}}_t(s_{t+1},a,\omega)\hat{\mathcal{P}}_t(s_t,a,s_{t+1})$.

Rewriting the problem in terms of the new variables as follows:

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

where $\sum_i b^{\max}_t(s_i)\ge 1$ to ensure existence of a feasible solution.

Expressing this problem as a Lagrangian enables efficient polynomial time solution that provides us a bound on the belief probability over a state at time step t+1, given the belief probabilities over states at time t. By bounding the belief probability over states, we can significantly improve the run-time of the LP, since the LP now performs the PRUNE operations mentioned above over narrower ranges of the belief space. DB leads to orders of magnitude improvement in existing algorithms [15] .

A second idea in DB-GIP is the Dynamic States (DS) technique, that exploits dynamic state spaces. Given an initial set of starting states, this technique initially finds the reachable states at each iteration, by using the transition function. Then, it finds dominant policies at each iteration over all belief probabilities of only these reachable states rather than over all belief probabilities of all the states. This reduction in reachable belief space, can significantly improve the performance of the existing algorithms[15]. DS in essence is a specialization of DB in that DS may be considered to impose a bound of 0 or 1 on the maximum belief probabilities over states, whereas DB may find a bound in between (e.g. 0.63).

2.2 Distributed POMDPs: MTDP

The distributed POMDP model that we base our work on is MTDP [17], however other models [3] could also be used. These distributed POMDP models are more than just two single agent POMDPs working independently. In particular, given a team of n agents, an MTDP [17] is defined as a tuple: $\langle S,A,P,\Omega,O,R\rangle$. S is a finite set of world states $\{s_1,\ldots,s_m\}$. $A=\times_{1\le i\le n}A_i$, where $A_1,\ldots,A_n$, are the sets of action for agents 1 to n. A joint action is represented as $\langle a_1,\ldots,a_n\rangle$.
$P(s_i,\langle a_1,\ldots,a_n\rangle,s_f)$, the transition function, represents the probability that the current state is s_f, if the previous state is s_i and the previous joint action is $\langle a_1,\ldots,a_n\rangle$. $\Omega=\times_{1\le i\le n}\Omega_i$ is the set of joint observations where Ω_i is the set of observations for agents i.
$O(s,\langle a_1,\ldots,a_n\rangle,\omega)$, the observation function, represents the probability of joint observation $\omega\in\Omega$, if the current state is s and the previous joint action is $\langle a_1,\ldots,a_n\rangle$. For the purpose of this paper, we assume that observations of each agent is independent of each other's observations. Thus the observation function can be expressed as $O(s,\langle a_1,\ldots,a_n\rangle,\omega)=O_1(s,\langle a_1,\ldots,a_n\rangle,\omega_1)$ $\cdot\ldots\cdot O_n(s,\langle a_1,\ldots,a_n\rangle,\omega_n)$. The agents receive a single immediate joint reward $R(s,\langle a_1,\ldots,a_n\rangle)$ which is shared equally.

Each agent i chooses its actions based on its local *policy*, π_i, which is a mapping of its observation history to actions. Thus, at time t, agent i will perform action $\pi_i(\vec{\omega}_i^t)$ where $\vec{\omega}_i^t = \omega_i^1, \ldots, \omega_i^t$. $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ refers to the joint policy of the team of agents. In this model, execution is distributed but planning is centralized.

2.3 Illustrative Domain

We consider the standard multiagent tiger problem from [11]. Two agents are in a corridor facing two doors "left" and "right". Behind one door lies a hungry tiger, and behind the other lies a reward. The set of states, S, is {SL, SR}, where SL indicates tiger behind the left door, and SR indicates tiger behind right door. The agents can jointly or individually open either door. In addition, the agents can independently listen for the presence of the tiger. Thus, the set of actions, A1 = A2 = {'OpenLeft', 'OpenRight', 'Listen'}. The transition function, P specifies that the problem is reset whenever an agent opens one of the doors. However, if both agents listen, the state remains unchanged. After every action each agent receives an observation about the new state. The observation functions are identical and will return either TL or TR with different probabilities depending on the joint action taken and the resulting world state. For example, if both agents listen and the tiger is behind the left door (state is SL), each agent independently receives the observation TL with probability 0.85 and TR with probability 0.15. For more details on this domain, refer to [11].

2.4 JESP algorithm for MTDPs

Given the NEXP-complete complexity of generating globally optimal policies for distributed POMDPs[3], locally optimal approaches [16, 4, 11] have emerged as viable solutions. In this paper, we build on the "JESP" (Joint Equilibrium-Based Search for Policies) [11] algorithm, outlined below (Algorithm 1). The key idea is to find the policy that maximizes the joint expected reward for one agent at a time, keeping policies of the other $n-1$ agents fixed. This process is repeated until an equilibrium is reached (local optimum is found). Multiple local optima are not encountered since planning is centralized. Key innovation in JESP is based on the realization that if policies of all other $n-1$ agents are fixed, then the remaining agent faces a normal single-agent POMDP, but with an extended state space. Thus, in line 4, given a known starting belief state, we use dynamic programming over belief states of this newer more complex single-agent POMDP, to compute agent 1's optimal response to fixed policies of the remaining $n-1$ agents.

Algorithm 1 JESP()

```
1:  $\Pi' \leftarrow$ randomly selected joint policy, prevVal $\leftarrow$ value of $\Pi'$,
    conv $\leftarrow 0$, $\Pi \leftarrow \Pi'$
2:  while conv $\neq n$ do
3:      for $i \leftarrow 1$ to $n$ do
4:          val, $\Pi_i \leftarrow$ OPTIMALBESTRESPONSE($b, \Pi', T$)
5:          if val = prevVal then
6:              conv $\overset{+}{\leftarrow} 1$
7:          else
8:              $\Pi'_i \leftarrow \Pi_i$, prevVal $\leftarrow$ val, conv $\leftarrow 1$
9:          if conv $= n$ then break
10: return $\Pi$
```

The key is then to define the extended state in JESP. For a two agent case, for each time t, the extended state of agent 1 is defined as a tuple $e_1^t = \langle s^t, \vec{\omega}_2^t \rangle$, where $\vec{\omega}_2^t$ is the observation history of the other agent. By treating e_1^t as the state of agent 1 at time t, we can define the transition function and observation function for the resulting single-agent POMDP for agent 1 as follows:

$$\begin{aligned} P'(e_1^t, a_1^t, e_1^{t+1}) &= \Pr(e_1^{t+1} | e_1^t, a_1^t) \\ &= P(s^t, (a_1^t, \pi_2(\vec{\omega}_2^t)), s^{t+1}) \\ &\quad \cdot O_2(s^{t+1}, (a_1^t, \pi_2(\vec{\omega}_2^t)), \omega_2^{t+1}) \end{aligned} \quad (1)$$

$$\begin{aligned} O'(e_1^{t+1}, a_1^t, \omega_1^{t+1}) &= \Pr(\omega_1^{t+1} | e_1^{t+1}, a_1^t) \\ &= O_1(s^{t+1}, (a_1^t, \pi_2(\vec{\omega}_2^t)), \omega_1^{t+1}) \end{aligned} \quad (2)$$

In other words, when computing agent 1's best-response policy via dynamic programming given the fixed policy of its teammate, we maintain a distribution over the extended states e_1^t, rather than over the world states s^t. Figure 1 shows a trace of the belief state evolution for the multi-agent tiger domain, described in Section 2.3, e.g. e_1^2 of SL(TR) indicates an extended state where the tiger is behind the left door and agent 2 has observed TR. However, as noted above, the main shortcoming of this technique is that it computes a locally optimal policy assuming a fixed starting belief state, and this assumption is embedded in its dynamic programming as shown in line 4 of algorithm 1 — it does not generate policies over continuous belief spaces.

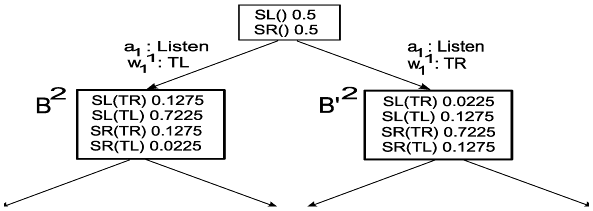

Figure 1: Trace of tiger scenario in JESP

3. CONTINUOUS SPACE JESP (CS-JESP)

One of the key insights in CS-JESP is the synergistic interaction between the JESP algorithm for distributed POMDPs and the DB-GIP technique of single agent POMDPs. We illustrate these interactions with a two-agent example in Section 3.1, and present key ideas in Section 3.2. Further, we describe the algorithm for n agents in Section 3.3 and some theoretical guarantees in Section 3.4.

Unlike previous work, our work focuses on continuous starting belief spaces and thus requires modifications for policy representation. In particular, because different policies may be dominant over different regions in the belief space, we introduce the notion of a *general policy*. A *general policy*, Π_i for an agent i is defined as a mapping from belief regions to policies. Π_i is represented as the set $\{(B_0^1,\pi_i^1),...(B_0^m,\pi_i^m)\}$, where $B_0^1, .., B_0^m$ are belief regions in the starting belief space B_0 and $\pi_i^1, ..\pi_i^m$ are the policies that will be executed starting from those regions. Henceforth we refer π_i^k as *specialized policies*. Thus, given a starting belief point $b_0^k \in B_0^k$, agent i on receiving observations $\omega_i^1, \ldots, \omega_i^t$ will perform the action $\pi_i^k(\vec{\omega}_i^t)$ where $\vec{\omega}_i^t = \omega_i^1, \ldots, \omega_i^t$. $\Pi = \langle \Pi_1, \ldots, \Pi_n \rangle$ refers to the joint general policy of the team of agents.

3.1 Illustrative Example

For ease of explanation, initially the algorithm is explained with two agents, Agent1 and Agent2. However, as we will show in Section 3.3, this algorithm is easily extendable to n agents. Initially, each agent selects a random general policy, Π_i, which will be a singleton set, $\{(B_0, \pi_i)\}$, i.e. a single *specialized policy*,π_i, over the entire starting belief space, B_0. While for exposition purposes this example describes policy computations by individual agents, in reality in CS-JESP these computations are performed by a centralized policy generator. CS-JESP begins when one agent, say Agent2, fixes its *general policy* Π_2, and other agent, Agent1, finds the best response for Agent2's *general policy*. Fixing Agent2's *specialized policy*, π_2, Agent1 creates a single agent POMDP with an extended state space, as explained in Section 2.4. Agent1 solves this POMDP using DB-GIP technique, explained in Section 2.1, with starting belief space as B_0, and obtains a new general policy Π_1, containing a set $\{(B_0^1, \pi_1^1), ...(B_0^m, \pi_1^m)\}$. Each $B_0^k \in B_0$ is a belief region and is represented by a minimum and maximum value for each of the $|S| - 1$ dimensions that represent the belief space. Now, Agent1 freezes its *general policy*, Π_1, and Agent2 solves a POMDP for each $\pi_1^j \in \Pi_1$, with the starting belief region as B^j. Thus, Agent2 solves m POMDPs, and obtains a new *general policy* Π_2. At this point, bordering regions in Π_2 that have identical policies are merged. This process continues until the solutions converge, and a local optimal is reached, i.e. no agent can improve its value vectors in any belief region.

Figure 2 illustrates the working of the algorithm with the multi-agent tiger scenario (Section 2.3) for time horizon, $T = 2$. Each tree in Figure 2 represents a *specialized policy*. All the trees on the left side of the figure are part of the *general policies* of Agent1, and trees on the right are part of the *general policies* of Agent2. For instance, at the end of iteration 3, both agents contain two specialized policies in their *general policy*. Within each tree (specialized policy), the letter inside each node indicates the action, and edges indicate the observation received. Thus, for the highlighted tree in the top left corner, the root node indicates the Listen(L) action, and upon either observing TL or TR, the *specialized policy* requires the agent to take a Listen(L) action. In this example, belief region over which a *specialized policy* dominates, consists of two numbers, namely the minimum and maximum belief probability of the state SL. These belief regions are indicated below each *specialized policy* in the figure. For instance, for the highlighted tree it is [0,1], but for other trees, regions such as [0.18,0.85] are shown.

The algorithm begins with both agents randomly selecting a *specialized policy* for the entire belief space [0,1]. In iteration 1, Agent2 fixes its *general policy*, and Agent1 comes up with its best response *general policy*. For calculating the best response, the Agent1 solves a POMDP with the starting belief range as [0,1], since Agent2's general policy is defined over this range. After the first iteration, Agent1 contains three *specialized policies* as part of its *general policy*, dominating over ranges [0,0.15], [0.15,0.85], [0.85,1]. In iteration 2, Agent1 fixes its *general policy*, and Agent2 begins its best response calculation with region [0,0.15]. For this range [0,0.15], Agent2 has only one dominant *specialized policy* and same is the case for [0.85,1]. However, for the range [0.15,0.85], Agent2 has two dominant *specialized policies*, one that dominates in the range [0.15,0.5], and the other that dominates in the range [0.5,0.85]. Thus after iteration 2, Agent2 has four *specialized policies* as part of its best response *general policy*. However, regions highlighted (with dotted rectangular boxes) have identical policies and thus after merging we are left with only two *specialized policies*. This algorithm continues with Agent2 fixing its *general policy* at iteration 3. Finally at convergence, each agent contains two *specialized policies* as part of their *general policies*.

3.2 Key Ideas

In this section, we explain in detail the key ideas in the CS-JESP algorithm, namely: (a) JESP and DB-GIP synergy; (b) Calculation of dominanant belief regions for *specialized policies*; (c) Region-based convergence; and (d) Merging of adjacent regions with identical *specialized policies*.

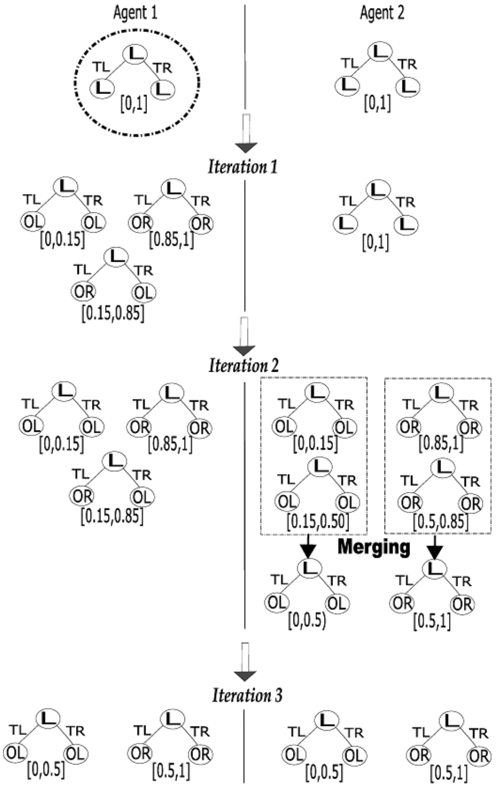

Figure 2: Trace of the algorithm for T=2 in Multi Agent tiger example with a specific starting joint policy

JESP and DB-GIP synergy: Both the DS and DB techniques of DB-GIP can provide significant performance improvements in CS-JESP. First, with respect to DS, JESP's state space is dynamic, where the set of states reachable at time t, e_i^t differ from the set of states at $t + 1$, e_i^{t+1}. DS can exploit this dynamism by computing dominant policies at time t over the belief space generated by the states in e_i^t thus reducing the dimensionality of the state space considered. For instance, in Figure 1, we have two initial states e_1^1= SL or SR, while there are four states e_1^2, e.g.$SL(TL), SL(TR)$ etc. Given a time horizon of T=2, instead of constructing a belief space over (2+4=) 6 dimensions, DS will lead to constructing a belief space over two states at the first time step and four states over the second time step. Such dimensionality reduction leads to significant speedups in CS-JESP. Second, with respect to DB, each agent solves a POMDP over the belief regions in the *general policies* of the other agents. DB is able to exploit forward projections of such starting belief regions to bound the maximum probabilities over states, and thus again restrict the belief space over which dominant policies are planned per belief region, obtaining additional speedups. For instance in Figure 2, at iteration2, Agent2 solves three POMDPs — these POMDPs are defined over extended states given three separate fixed policies of Agent1— one with the starting belief region as [0,0.15], another with [0.15,0.85], and a third with [0.85,1]. Thus, in solving the POMDP starting with the belief range [0,0.15], DB helps prune all the unreachable portions of the belief space given that the starting range is [0,0.15]. In all three POMDPs, the belief region is narrower compared to [0,1].

Region-based convergence: Given continuous initial belief space, we obtain value vectors (vector containing values for all the states) for all the belief regions in the *general policy*. Thus, convergence is attained when for all agents the value vectors at the current iteration for all the belief regions are equal to those in the previous iteration. For instance, in Figure 2, the convergence is attained in the fourth iteration, with the *general policy* of Agent2 containing the two exact same *specialized policies* from iteration 3. However, once one region has converged — the value vectors for all agents do not change from one iteration to the next for that region — CS-JESP will not test that region further for convergence, but only continue changing policies in regions that have failed to converge.

Merging of adjacent regions with identical *specialized policies*: Merging such regions can be important as the other agent would have to solve fewer number of POMDPs in the next iteration. For instance, in the *general policy* of Agent2 before merging at iteration 2, belief regions [0,0.15] and [0.15,0.5] have identical specialized policies. Similarly, regions [0.5, 0.85] and [0.85,1] have identical specialized policies. Thus Agent2 has only two *specialized policies* after merging (instead of four before merging) and this leads to agent1 solving two instead of four POMDPs at iteration 3.

Merging requires identifying regions adjacent to each other. In the Tiger domain, this is done by doing adjacency check for regions along one dimension. However, finding bordering regions in a $|S|$ dimensional state space requires comparisons along $|S|-1$ dimensional space.

Calculation of dominant belief regions for *specialized policies*: One standard way of representing solutions in single agent POMDPs is through value vectors. In this representation, the best policy for a belief point, *b*, is computed by testing for a vector that provides the maximum expected value for that belief point.
$\pi_1^* \leftarrow argmax_{\pi \in \{\pi_1\}} v_\pi \cdot b$.

However, in CS-JESP, one agent uses the belief regions of the other agent to calculate the best responses over each of those belief regions. We develop a linear program to address the dominant belief region computation for each policy. Algorithm 2 computes the maximum belief probability of a state, s_j, where a policy or value vector,v dominates all the other policies or value vectors, $\mathcal{V} - v$ in the final policy. Constraint 1 in Algorithm 2, computes points where v dominates all the other vectors in $\mathcal{V}$. Objective function of the algorithm is a maximization over $b(s_j)$, thus finding highest possible belief probability for state s_j amongst all those dominating points. In a similar way, the minimum for s_j can be found by doing a minimize, instead of maximize, in line 1 of the LP. The belief region is calculated by solving these max, min LPs for each state $s_j \in S$. Thus, requiring $2 * |\mathcal{V}| * |S|$ number of LPs to be solved for the computation of an entire belief region.

```
Algorithm 2 MaximumBelief($s_j, v, \mathcal{V}, B^{min}, B^{max}$)
  Maximize $b(s_j)$
  subject to constraints
      1. $b.(v - v') > 0, \forall v' \in \mathcal{V} - v$
      2. $\Sigma_{s \in S} b(s) = 1$
      3. $B^{min}(s) < b(s) < B^{max}(s), \forall s \in S$
```

3.3 Algorithm for n agents

In this section, we present the CS-JESP algorithm (Algorithm 3) for n agents. In the initialization stage (lines 1-4), each agent i has only one belief region that corresponds to its entire belief space (Π_i'*.beliefPartition*). Also, each agent has a single randomly selected *specialized policy*, $\Pi_i'.\pi[\langle[0,1],\ldots,[0,1]\rangle]$ (i.e. π is the *specialized policy*), for the entire belief space (line 3). Every *general policy* has "*count*" for each belief region, to track the convergence of policies in that belief region (region-based convergence) — if the *count* reaches n then the region has converged, because no agent will change any further. The flag "*converged*" monitors if joint *general policies* in all the regions have converged.

In each iteration (one execution of lines 6-23) of Algorithm 3, we choose an agent i and find its optimal response to the fixed *general policies* of the remaining agents by calling optimalBestResponse(). This is repeated until no agent acting alone can improve upon the joint expected reward by changing its own *general policy*.

Although each agent i starts off with the same belief set partition, Π_i'*.beliefPartition*, this will not be true after calling optimalBestResponse() as seen in Figure 2. The function updatePartition() (Algorithm 4) is responsible for creating a new belief set partition for an agent i, depending on the belief regions of the other $n-1$ agents. This new belief set partition is obtained by splitting the overlapping belief regions of the $n-1$ agents, in a way that no two resulting belief regions, which now belong to this partition, overlap. Furthermore, this function computes the Π_i*.count* for all the new regions, from the $count$ values for the regions in Π_j', where j was the free agent in the last iteration (i.e the agent who computed the best response in the last iteration).

findNewPartition() (Algorithm 5) takes two arguments, (i) *partition* and (ii) a belief region, *br*, and it generates all feasible partitions from the two arguments. To illustrate the working of this function, we provide an example with three states $\{s_1, s_2, s_3\}$. Belief regions in the corresponding belief space can be represented with minimum and maximum belief probabilities for just s_1 and s_2, i.e. $\{(b^{min}[s_1], b^{max}[s_1]), (b^{min}[s_2], b^{max}[s_2])\}$. For example, let *partition* $=\{\langle[0,0.8],[0.5,0.9]\rangle\}$ (has only one region) and $br = \{[0.4,0.9],[0.3,0.6]\}$. In the first step (line 3), partitions are found for each state, s_i separately. Thus, for the first state, s_1, $[0,0.8]$ and $[0.4,0.9]$ yields partitions, $[0,0.4]$, $[0.4,0.8]$, $[0.8,0.9]$. Similarly for the second state, s_2, the partitions found are $[0.3,0.5]$, $[0.5,0.6]$,$[0.6,0.9]$. In the second step (line 4), we compute the cross product of these individual dimension partitions. This gives rise to nine belief regions, viz. $\{[0,0.4],[0.3,0.5]\},\ldots,\{[0.8,0.9],[0.4,0.8]\}$, $\{[0.8,0.9],[0.8,0.9]\}$. Finally, in the third step (line 5), we prune regions which do not contain any valid points, i.e. $\sum_{s \leq |S|-1} b^{min}[s] > 1$. For instance, the region {[0.8,0.9],[0.4,0.8]} can be pruned, because a belief point in this region has probability of atleast $1.2\ (= 0.8 + 0.4)$.

The function optimalBestResponse() (Algorithm 6) is then called separately for each belief region in agent i's belief set partition. It returns a new partitioniong of the initial belief space and the optimal policy for each belief region in this partition. ConstructExtendedPOMDP() constructs a POMDP with extended state space, as explained in Section 2.4, while the function CalculateBeliefRegion() computes the belief regions where each vector v ($\in \mathcal{V}$) dominates.

After computing best responses, CS-JESP() ensures that the number of belief partitions obtained are finite (in lines 14-16) and that Π*.count* is updated correctly for each belief region (in lines 18-21).

It is possible that an agent's best response in adjacent belief regions is the same policy. The function mergeBeliefRegions() (Algorithm 3.3) is responsible for merging such kind of regions (lines 4-7). Further, once the policies in a belief region have converged, that region is not considered for subsequent merging phases (first part of the condition on line 4).

3.4 Theoretical Results

Algorithm 3 CS-JESP()

```
 1: for $i \leftarrow 1$to $n$ do
 2:   $\Pi'_i$.beliefPartition $\leftarrow \{\langle [0,1], \dots, [0,1] \rangle\}$
 3:   $\Pi'_i.\pi[\langle [0,1], \dots, [0,1] \rangle] \leftarrow$ random specialized policy
 4:   $\Pi'_i$.count$[\langle (0,1), \dots, (0,1) \rangle] \leftarrow 0$
 5: converged $\leftarrow$ false; $i \leftarrow n$;
 6: while converged $=$ false do
 7:   $i \leftarrow (i$ MOD $n) + 1$; converged $\leftarrow$ true
 8:   UPDATEPARTITION$(i, \Pi_i, \Pi')$
 9:   for all br in $\Pi_i$.beliefPartition do
10:     if $\Pi_i$.count[br] $< n$ then
11:       converged$\leftarrow$false
12:       $\{\Pi_i$, regions$\} \leftarrow$ OPTIMALBESTRESPONSE$(i, \Pi'$, br$)$
13:       for all $br_1$ in regions do
14:         $\pi \leftarrow \Pi_i.\pi[br_1]$; REMOVE$(\Pi_i, br_1)$
15:         for dim $\leftarrow 1$ to $|S| - 1$ do
16:           $br_1$[dim] $\leftarrow$ ROUNDOFF$(br_1$[dim], precision$)$
17:         if VOLUME$(br_1) > 0$ then
18:           ADD$(\Pi_i$.beliefPartition$, br_1, \pi)$
19:           if $\Pi_i.\pi[br_1] = \Pi'_i.\pi[br]$ then
20:             $\Pi_i$.count$[br_1] \leftarrow \Pi'_i$.count$[br] + 1$
21:           else
22:             $\Pi_i$.count$[br_1] \leftarrow 1$
23:   MERGEBELIEFREGIONS$(\Pi_i)$
24:   $\Pi'_i \leftarrow \Pi_i$
25: return $\Pi$
```

Algorithm 4 UPDATEPARTITION(i, Π)

```
$\Pi_i \leftarrow \Pi_{(i \text{ MOD } n)+1}$
for all $j$ in$\{1, \dots, n\} - \{i, (i$ MOD $n) + 1\}$ do
  for all $br_1$ in $\Pi_j$.beliefPartition do
    if $\Pi_i$.count$[br_1] < n$ then
      $\Pi_i$.beliefPartition $\leftarrow$ FINDNEWPARTITION$(\Pi_i$.beliefPartition$, br_1)$
if $i = 1$ then $j \leftarrow n$else $j \leftarrow i - 1$
for all $br_2$ in $\Pi_i$.beliefPartition do
  $br_3 \leftarrow$ OVERLAPPINGREGION$(\Pi_j$.beliefPartition$, br_2)$
  $\Pi_i$.count$[br_2] \leftarrow \Pi_j$.count$[br_3]$
return
```

Algorithm 5 FINDNEWPARTITION((partition, br))

```
newPartition $\leftarrow \emptyset$
for dim $\leftarrow 1$ to $|S| - 1$ do
  1DPartition $\leftarrow$ SPLITDIMENSION(dim, br, partition)
  newPartition $\leftarrow$ CROSSPRODUCT(newPartition, 1DPartition)
  newPartition $\leftarrow$ PRUNE(newPartition)
return newPartition
```

Algorithm 6 OPTIMALBESTRESPONSE$(i, \Pi'$, br$)$

```
$k \leftarrow 0$
extendedPOMDP $\leftarrow$ CONSTRUCTEXTENDEDPOMDP$(i, \Pi'$, br$)$
$\{\mathcal{V}, \pi^{new}\} \leftarrow$ DB-GIP(extendedPomdp, br)
for $j \leftarrow 1$ to $\mathcal{V}.size$ do
  $v \leftarrow \mathcal{V}[j]$; $\mathcal{V}' \leftarrow \mathcal{V} - v$
  beliefPartition$[k] \leftarrow$ CALCULATEBELIEFREGION$(v, \mathcal{V}'$, br$)$
  $\Pi_i$.beliefPartition$[k] \leftarrow$ beliefPartition$[k]$
  $\Pi_i.\pi$[beliefPartition$[k]] \leftarrow \pi^{new}[j]$; $k \overset{+}{\leftarrow} 1$
return $\{\Pi_i$, beliefPartition$\}$
```

In the following proofs, we use "iteration" to mean one execution of the "while" loop (lines 5-23) of Algorithm 3, n for the number of agents, and "free agent" to denote the i^{th} agent for that iteration.

Algorithm 7 MERGEBELIEFREGIONS(Π_i)

```
for each $b_1$ in $\Pi_i$.beliefPartition do
  if $\Pi_i$.count$(b_1) < n$ then
    for each $b_2$ in $\Pi_i$.beliefPartition do
      if $\Pi_i$.count$(b_2) < n \wedge \Pi_i.\pi[b_1] = \Pi_i.\pi[b_2]$ then
        if ISADJACENT$(b_1, b_2)$ then
          $b \leftarrow$MERGEREGIONS$(b_1, b_2)$
          ADD$(\Pi_i$.beliefPartition$, b, \Pi_i.\pi[b_1])$
          $\Pi_i$.count$[b] \leftarrow$ min$(\Pi_i$.count$[b_1], \Pi_i$.count$[b_2])$
          REMOVE$(\Pi_i, b_1)$; REMOVE$(\Pi_i, b_2)$
return $\Pi_i$
```

PROPOSITION 1. *In CS-JESP, the joint expected reward for all starting belief points is monotonically increasing with each iteration.*

Proof Sketch. In every iteration, each starting belief point must belong to one of the regions in the belief partition of the free agent. Each such belief region corresponds to one of the value vectors, calculated by a call to OPTIMALBESTRESPONSE(Algorithm 6). Since DB-GIP is optimal, these vectors should either equal or dominate the vectors at the previous iteration, in all belief regions.■

PROPOSITION 2. *CS-JESP will terminate iff the joint policy has converged in all the free agent's belief regions.*

Proof Sketch. By construction, CS-JESP (Algorithm 3) terminates iff *converged* $=$ true, which will happen iff Π_i.*count*[*br*] $\geq n$, for all belief regions *br* of the free agent i. Π_i.*count*[*br*] $\geq n$ iff the joint policy for the region *br* remains constant for n iterations. In order for the joint *general policy* to remain constant for n iterations, OPTIMALBESTRESPONSE() should return identical *specialized policies* (to those in previous iteration) for all the belief regions , for $n - 1$ free agents. This happens when no one agent can improve the global value by altering its *general policy*, i.e. when local optima is attained. Furthermore, we round off each dimension of a belief region to *precision* decimal spaces, and hence the number of possible belief regions cannot grow indefinitely.■

From Propositions 1 and 2, we can conclude that CS-JESP will always terminate. At termination, the joint policy will be locally optimal as long as none of the belief regions returned by OPTIMALBESTRESPONSE were eliminated by the ROUNDOFF procedure.

4. EXPERIMENTAL RESULTS

This section provides three types of evaluations for CS-JESP using the multiagent tiger domain [11]. The first experiment focuses on run-time evaluations. We provide a comparison of three techniques: (i) CS-JESP+GIP: is the basic version of the combination of the JESP and the value iteration algorithm, GIP of single agent POMDPs. (ii) CS-JESP+DB, is JESP with DB-GIP. (iii)CS-JESP+DBM is CS-JESP+DB with the merging enhancement. Results of this experiment are shown in Figure 3. We experiment with two separate reward structures (presented in [11]). Figure 3(a) and Figure 3(b) focus on reward structure1, while Figure 3(c) and Figure 3(d) focus on reward structure2. In Figure 3(a), x-axis plots varying time horizon while y-axis plots run-time in milliseconds on log-scale[1]. In Figure 3(b), x-axis again plots time horizon, but the y-axis plots run-time in milliseconds (no log-scale is used). Time limit for the problems was set at 7,500,000 ms, after which they

[1] Machine specs for all experiments: Intel Xeon 3.6 GHZ processor, 2GB RAM

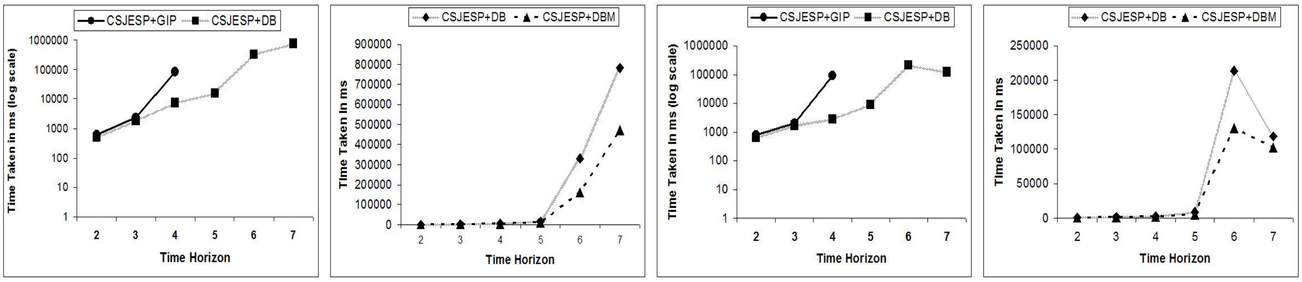

Figure 3: Comparison of (a) CSJESP+GIP, and CSJESP+DB for reward structure 1 (b) CSJESP+DB, and CSJESP+DBM for reward structure 1 (c) CSJESP+GIP, and CSJESP+DB for reward structure 2 (d) CSJESP+DB, and CSJESP+DBM for reward structure 2

were terminated. Figure 3(a) and Figure 3(c) refer to comparisons between CS-JESP+GIP and CS-JESP+DB, while Figure 3(b) and Figure 3(d) refer to comparisons between CS-JESP+DB and CS-JESP+DBM for the two reward structures.

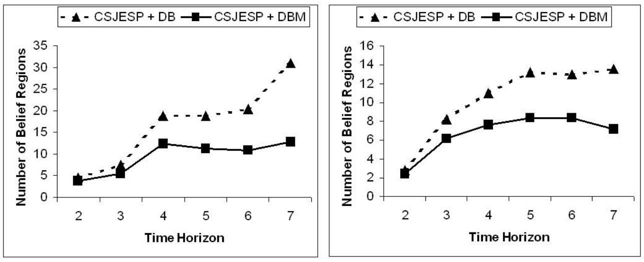

Figure 4: Comparison of the number of belief regions created in CS-JESP+DB and CS-JESP+DBM for reward structures 1 and 2

Figure 3(a) shows that CS-JESP+GIP did not terminate within the specified time limit after T=4. However, CS-JESP+DB converged to the solution even for T = 7, within the specified time limit. Even in cases where CS-JESP+GIP terminates, CS-JESP+DB provides significant speedups. For instance, in Figure 3(a), at T = 4, while CS-JESP+GIP takes in 83717.8 ms, CS-JESP+DB takes only 7345.2 ms leading to a speedup of 11.4 fold. Similar conclusions can be drawn from Figure 3(c). These results illustrate the synergy of JESP and DB-GIP, and the suitability of CS-JESP to take advantage of DB-GIP.

Figure 3(b) shows that CS-JESP+DBM provides further speedups over CS-JESP+DB, as time horizon increases. For instance at T=7 in Figure 3(b), merging in CS-JESP+DBM provided 1.66 fold speedup over CS-JESP+DB. Similar results are obtained with reward structure 2 in Figure 3(d), thus establishing the utility of merging contiguous regions with identical policies. In Figure 3(d) T=7 post merging show a faster execution compared to the T=6 results post merging. This occurs because the number of iterations of CS-JESP required for convergence at T=7 are lower (6) compared with iterations at T=6 (11).

Our second evaluation in Figure 4 focuses on understanding the speedups due to merging in CS-JESP+DBM. The number of belief regions present in the final solution is an indicator of the number of single agent POMDPs getting solved at each iteration. The x-axis in the figures represents the time horizon, while the y-axis is the number of belief regions. Thus in Figure 4, for a time-horizon of 7, using CSJESP+DB led to 31 belief regions, whereas using CSJESP+DBM led to 13 belief regions, a 2.39-fold reduction in the number of belief regions considered. Furthermore, we see that increasing the time horizon leads to increasing reduction in the number of belief regions with CS-JESP+DBM when compared to the number with CS-JESP+DB. Effect of the number of belief regions on the time taken increases with time horizon, because the single agent POMDPs expand in size with the time horizon. This provides the explanation for the timing results for CS-JESP+DBM in Figure 3(b) and Figure 3(d).

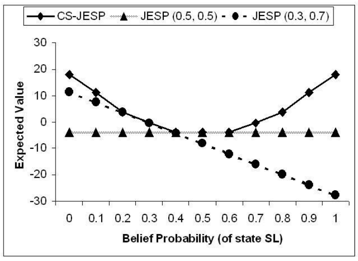

Figure 5: Comparison of the expected values obtained with JESP for specific belief points and CS-JESP

Our third evaluation focused on illustrating that CS-JESP achieves what it set out to do — generating policies over continuous initial belief space as shown in Figure 5. Belief space (in this domain belief probability of SL) is denoted on the x-axis, while the expected value of the policy is depicted on the y-axis. CS-JESP provides a general policy where the expected value is represented by a “CUP”-shape. There are five different policies represented in the cup, each dominant over a single belief region. The figure also indicates that if we were to approximate this entire general policy with a single policy over a single starting belief state, e.g. with JESP, then results may be arbitrarily worse. For instance with JESP (0.3, 0.7), the value at (1, 0) is -27, while the value generated with CS-JESP is 18, a difference of 45. With JESP (0.5, 0.5), the value at (1,0) is -4, where CS-JESP attains a value of 18, a difference of 22.

Of course, we may sample several belief points with JESP and then for a new belief point provide a policy from the nearest sample. Such a proposed heuristic approach naturally leads to our fourth evaluation comparing CS-JESP runtime to an approach that samples the belief space. This evaluation is not meant to be a precise comparison of JESP and CS-JESP, instead the aim is to show that the run-time results for sampled JESP would be comparable to the run times of CS-JESP. In Table 1, we show the run times of JESP and CS-JESP for T = 6, and T=7 for reward structure1. To replicate the policy obtained with CS-JESP, JESP would have to sample atleast as many times as the number of belief regions in the final policy of CS-JESP. For instance, for T=7, the number of samples required for JESP would be thirteen (from Figure 4). Table 1 shows an estimate of such a sampled JESP technique, given runtime results from [12]. We see that CS-JESP run-times are comparable, yet CS-JESP provides guarantees on these results that are unavailable with sampling.

	CS-JESP	JESP	Sampled Regions	Sampled JESP
T = 6	160336	15000	11	165000
T = 7	470398	73000	13	949000

Table 1: Comparison of run times (in ms) for JESP and CS-JESP

5. SUMMARY AND RELATED WORK

Distributed POMDPs are evolving as a popular approach in modeling multiagent systems, and many different algorithms have been proposed to obtain locally or globally optimal policies. Unfortunately, most of these have either been explicitly designed or experimentally evaluated assuming knowledge of a starting belief, an assumption that often does not hold in complex, uncertain domains. Instead, in such domains, it is important for agents to explicitly plan over continuous belief spaces. This paper provides a novel algorithm, CS-JESP, to explicitly compute finite horizon policies over continuous belief spaces, without restricting the space of policies. By marrying an efficient single-agent POMDP solver with a heuristic distributed POMDP policy-generation algorithm, locally optimal joint policies are obtained, each of which dominates within a different belief region. Key ideas in CS-JESP include: (a) Synergistic interaction between JESP and DB-GIP that enables significant performance improvement in CS-JESP; (b) Calculation of dominanant belief regions for *specialized policies*; (c) Region-based convergence; and (d) Merging of adjacent regions with identical *specialized policies*. To the best of our knowledge, these are the first run-time results for analytically generating policies over continuous belief spaces in distributed POMDPs.

In terms of related work, approaches for solving distributed POMDPs can be broadly classified in two main categories: globally optimal approaches and locally optimal algorithms. Becker *et al.* [2] present an exact globally optimal algorithm – the coverage set algorithm for transition-independent distributed MDPs. However unlike CS-JESP, this algorithm starts from a particular known initial state distribution. Hansen *et al.* [8] present an exact algorithm for partially observable stochastic games (POSGs) based on dynamic programming and iterated elimination of dominant policies. Like our work, they too deal with a continuous belief space and an unknown initial state distribution. This algorithm is important from a theoretical standpoint, but because of the inherent complexity of finding an exact solution for general distributed POMDPs, this algorithm does not scale well.

Among locally optimal approaches, Peshkin *et al.* [16] use gradient descent search to find local optimum finite-controllers with bounded memory. Their algorithm finds locally optimal policies from a limited subset of policies, with an infinite planning horizon. Their work does not consider a continuous belief space and starts from a fixed belief point. We have earlier discussed Nair *et al.* [11]'s JESP algorithm that uses dynamic programming to reach a local optimal. Bernstein *et al.* [7] present a locally optimal bounded policy iteration algorithm for infinite-horizon distributed algorithms. This algorithm has been theoretically shown to work in a continuous belief space from an unknown initial belief distribution. While this is an important contribution, the use of finite-state controllers restricts the policy representation. Also, their experimental results are for a single initial belief. Further, unlike our algorithm they use a *correlation device* in order to ensure coordination among the various agents.

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, DARPA/IPTO COORDINATORS program and the Air Force Research Laboratory under Contract No. FA875005C0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

6. 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] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-independent decentralized Markov decision processes. In *AAMAS*, 2003.

[3] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of MDPs. In *UAI*, 2000.

[4] I. Chadès, B. Scherrer, and F. Charpillet. A heuristic approach for solving decentralized-pomdp: Assessment on the pursuit problem. In *SAC*, 2002.

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

[6] Claudia V. Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *AAMAS*, 2003.

[7] D. Bernstein; E. Hansen; and S. Zilberstein. Bounded policy iteration for decentralized pomdps. In *IJCAI*, 2005.

[8] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, 2004.

[9] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *AIJ*, 101(2), 1998.

[10] R.E. Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS*, 2004.

[11] R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, 2003.

[12] R. Nair, M. Roth, M. Yokoo, and M. Tambe. Communication for improving policy computation in distributed pomdps. In *AAMAS*, 2004.

[13] R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *AAMAS*, 2003.

[14] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, 2005.

[15] M. Tambe P. Varakantham, R. Maheswaran. Exploiting belief bounds: Practical pomdps for personal assistant agents. In *AAMAS*, 2005.

[16] L. Peshkin, N. Meuleau, K.-E. Kim, and L. Kaelbling. Learning to cooperate via policy search. In *UAI*, 2000.

[17] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR*, 16:389–423, 2002.

[18] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multiagent cooperation. In *Agents*, 2001.