Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings

R. Nair and M. Tambe
Computer Science Dept.
University of Southern California
Los Angeles CA 90089
{nair,tambe}@usc.edu

M. Yokoo
Coop. Computing Research Grp.
NTT Comm. Sc. Labs
Kyoto, Japan 619-0237
yokoo@cslab.kecl.ntt.co.jp

D. Pynadath, S. Marsella
Information Sciences Institute
University of Southern California
Marina del Rey CA 90292
{pynadath, marsella}@isi.edu

Abstract

The problem of deriving joint policies for a group of agents that maximize some joint reward function can be modeled as a *decentralized* partially observable Markov decision process (POMDP). Yet, despite the growing importance and applications of decentralized POMDP models in the multiagents arena, few algorithms have been developed for efficiently deriving joint policies for these models. This paper presents a new class of locally optimal algorithms called "Joint Equilibrium-based search for policies (JESP)". We first describe an exhaustive version of JESP and subsequently a novel dynamic programming approach to JESP. Our complexity analysis reveals the potential for exponential speedups due to the dynamic programming approach. These theoretical results are verified via empirical comparisons of the two JESP versions with each other and with a globally optimal brute-force search algorithm. Finally, we prove piece-wise linear and convexity (PWLC) properties, thus taking steps towards developing algorithms for continuous belief states.

1 Introduction

As multiagent systems move out of the research lab into critical applications such as multi-satellite control, researchers need to provide high-performing, robust multiagent designs that are as nearly optimal as feasible. To this end, researchers have increasingly resorted to decision-theoretic models as a framework in which to formulate and evaluate multiagent designs. Given a group of agents, the problem of deriving separate policies for them that maximize some joint reward can be modeled as a decentralized POMDP (Partially Observable Markov Decision Process). In particular, the DEC-POMDP (Decentralized POMDP) [Bernstein *et al.,* 2000] and MTDP (Markov Team Decision Problem [Pynadath and Tambe, 2002]) are generalizations of a POMDP to the case where there are multiple, distributed agents basing their actions on their separate observations. These frameworks allow a variety of multiagent analysis. Of particular interest here, they allow us to formulate what constitutes an optimal policy for a multiagent system and in principle derive that policy.

However, with a few exceptions, effective algorithms for deriving policies for decentralized POMDPs have not been developed. Significant progress has been achieved in efficient single-agent POMDP policy generation algorithms [Monahan, 1982; Cassandra *et al,* 1997; Kaelbling *et al,* 1998]. However, it is unlikely such research can be directly carried over to the decentralized case. Finding optimal policies for decentralized POMDPs is NEXP-complete [Bernstein *et al,* 2000]. In contrast, solving a POMDP is PSPACE-complete [Papadimitriou and Tsitsiklis, 1987]. As Bernstein et al. [2000] note, this suggests a fundamental difference in the nature of the problems. The decentralized problem cannot be treated as one of separate POMDPs in which individual policies can be generated for individual agents because of possible cross-agent interactions in the reward, transition or observation functions. (For any one action of one agent, there may be many different rewards possible, based on the actions that other agents may take.) In some domains, one possibility is to simplify the nature of the policies considered for each of the agents. For example, Chades *et al.* [2002] restrict the agent policies to be memoryless (reactive) policies. Further, as an approximation, they define the reward function and the transition function over observations instead of over states thereby simplifying the problem to solving a multi-agent MDP [Boutilier, 1996]. Xuan *et al.* [2001] describe how to derive decentralized MDP (not POMDP) policies from a centralized MDP policy. Their algorithm, which starts with an assumption of full communication that is gradually relaxed, relies on instantaneous and noise free communication. Such simplifications reduce the applicability of the approach and essentially side-step the question of solving decentralized POMDPs. Peshkin *et al.* [2000] take a different approach by using 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, while our algorithm finds locally optimal policies from an unrestricted set of possible policies, with a finite planning horizon.

Thus, there remains a critical need for new efficient algorithms for generating optimal policies in distributed POMDPs. In this paper, we present a new class of algorithms for solving decentralized POMDPs, which we refer to as Joint Equilibrium-based Search for Policies (JESP). JESP iterates through the agents, finding an optimal policy for each agent assuming the policies of the other agents are fixed. The iteration continues until no improvements to the joint reward is achieved. Thus JESP achieves a local optimum similar to a Nash Equilibrium. We discuss Exhaustive-JESP which uses exhaustive search to find the best policy for each agent. Since this exhaustive search for even a single agent's policy can be very expensive, we also present DP-JESP which improves on Exhaustive-JESP by using dynamic programming to incrementally derive the policy. We conclude with several empirical evaluation that contrast JESP against a globally optimal algorithm that derives the globally optimal policy via a full search of the space of policies. Finally, we prove piece-wise linear and convexity (PWLC) properties, thus taking steps towards developing algorithms for continuous initial belief states.

2 Model

We describe the Markov Team Decision Problem (MTDP) [Pynadath and Tambe, 2002] framework in detail here to provide a concrete illustration of a decentralized POMDP model. However, other decentralized POMDP models could potentially also serve as a basis [Bernstein *et ai,* 2000; Xuan et al.,2001].

Given a team of n agents, an MTDP [Pynadath and Tambe, 2002] is defined as a tuple: $\langle S, A, P, \Omega, O, R\rangle$. 5 is a finite set of world states $\{1, 2, \ldots, m\}$. *A* = $\times_{1<=i<=n} A_i$, where *A_1,...,* *A_n,* are the sets of action for agents 1 to n. A joint action is represented as *(a_1,...,* a„). P(s_i,(a_1,...,a_n) ,s_f), the transition function, represents the probability of the current state is .s_f, if the previous state is *S_i* and the previous joint action is (o_1,...,a„). $\Omega_1, \ldots, \Omega_n$ are the set of observations for agents 1 to n. 0(s, (a_1,... ,a_n) *,u)* , the observation function, represents the probability of joint observation ω, if the current state is *$* and the previous joint action is (a_1,..., a_n). The agents receive a single, immediate joint reward R(s,a_1,.. . ,a_n) which is shared equally.

Practical analysis using models like MTDP often assume that observations of each agent is independent of each other's observations. Thus the observation function can be expressed as 0(s, (ai,..., a_n), ω) = *O_1(s,* *(a_u...* ,a„), ω_1) •...• ω_n)

Each agent *i* chooses its actions based on its local *policy,* *7Tj*, 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. The important thing to note is that in this model, execution is distributed but planning is centralized. Thus agents don't know each other's observations and actions at run-time but they know each other's policies.

3 Example Scenario

For illustrative purposes it is useful to consider a familiar and simple example, yet one that is capable of bringing out key difficulties in creating optimal policies for MTDPs. To that end, we consider a multiagent version of the classic tiger problem used in illustrating single agent POMDPs[Kaelbling *et ai,* 1998] and create an MTDP ((S, *A, P,* Ω, O, *R))* for this example. In our modified version, two agents are in a corridor facing two doors:"left" and "right". Behind one door lies a hungry tiger and behind the other lies untold riches but the agents do not know the position of either. Thus, 5 = *{SL,SR},* indicating behind which door the tiger is present. The agents can jointly or individually open either door. In addition, the agents can independently listen for the presence of the tiger. Thus, *A_1* = *A_2* — *{'OpenLeft', 'OpenRight', 'Listen'}.* The transition function *P,* specifics that every time either agent opens one of the doors, the state is reset to *SL* or *SR* with equal probability, regardless of the action of the other agent, as shown in Table 1. However, if both agents listen, the state remains unchanged. After every action each agent receives an observation about the new state. The observation function, *0_1* or 0_2 (shown in Table 2) will return either *HL* or *HR* 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 receives the observation *HL* with probability 0.85 and *HR* with probability 0.15.

Action/Transition	SL → SL	SL → SR	SR → SR	SR → SL
<OpenRight,*>	0.5	0.5	0.5	0.5
<OpenLeft,*>	0.5	0.5	0.5	0.5
<*,OpenLeft>	0.5	0.5	0.5	0.5
<*,OpenRight>	0.5	0.5	0.5	0.5
<Listen,Listen>	1.0	0.0	1.0	0.0

Table 1: Transition function

Action	State	HL	HR
<Listen,Listen>	SL	0.85	0.15
<Listen,Listen>	SR	0.15	0.85
<OpenRight,*>	*	0.5	0.5
<OpenLeft,*>	*	0.5	0.5
<*,OpenLeft>	*	0.5	0.5
<*,OpenRight>	*	0.5	0.5

Table 2: Observation function for each agent

If either of them opens the door behind which the tiger is present, they are both attacked (equally) by the tiger (sec Table 3). However, the injury sustained if they opened the door to the tiger is less severe if they open that door jointly than if they open the door alone. Similarly, they receive wealth which they share equally when they open the door to the riches in proportion to the number of agents that opened that door. The agents incur a small cost for performing the '*Listen*' action.

Clearly, acting jointly is beneficial (e.g., *A_1* — *A2* = *'OpenLeft')* because the agents receive more riches and sustain less damage by acting together. However, because the agents receive independent observations (they do not share observations), they need to consider the observation histories of the other agent and what action they are likely to perform.

Action/State	SL	SR
<OpenRight,OpenRight>	+20	-50
<OpenLeft,OpenLeft>	-50	+20
<OpenRight,OpenLeft>	-100	-100
<OpenLeft,OpenRight>	-100	-100
<Listen,Listen>	-2	-2
<Listen,OpenRight>	+9	-101
<OpenRight,Listen>	+9	-101
<Listen,OpenLeft>	-101	+9
<OpenLeft,Listen>	-101	+9

Table 3: Reward function A

We also consider consider another case of the reward function, where we vary the penalty for jointly opening the door to the tiger (See Table 4).

Action/State	SL	SR
<OpenRight,OpenRight>	+20	0
<OpenLeft,OpenLeft>	-50	+20
<OpenRight,OpenLeft>	-100	-100
<OpenLeft,OpenRight>	-100	-100
<Listen,Listen>	-2	-2
<Listen,OpenRight>	+9	-101
<OpenRight,Listen>	+9	-101
<Listen,OpenLeft>	-101	+9
<OpenLeft,Listen>	-101	+9

Table 4: Reward function B

4 Optimal Joint Policy

When agents do not share all of their observations, they must instead coordinate by selecting policies that are sensitive to their teammates' possible beliefs, of which each agent's entire history of observations provides some information. The problem facing the team is to find the *optimal* joint policy, i.e. a combination of individual agent policies that produces behavior that maximizes the team's expected reward.

One sure-fire method for finding the optimal joint policy is to simply search the entire space of possible joint policies, evaluate the expected reward of each, and select the policy with the highest such value. To perform such a search, we must first be able to determine the expected reward of a joint policy. We compute this expectation by projecting the team's execution over all possible branches on different world states and different observations. We present here the 2-agent version of this computation, but the results easily generalize to arbitrary team sizes. At each time step, we can compute the expected value of a joint policy, $\pi = \langle \pi_1, \pi_2 \rangle$, for a team starting in a given state, S^t, with a given set of past observations, $\vec{\omega}_1^t$ and $\vec{\omega}_2^t$, as follows:

$$\begin{aligned} V_\pi^t(S^t, \langle \vec{\omega}_1^t, \vec{\omega}_2^t \rangle) = R(S^t, \langle \pi_1(\vec{\omega}_1^t), \pi_2(\vec{\omega}_2^t) \rangle) + \\ \sum_{S^{t+1} \in S} P\left(S^t, \langle \pi_1\left(\vec{\omega}_1^t\right), \pi_2\left(\vec{\omega}_2^t\right) \rangle, S^{t+1}\right) \\ \cdot \sum_{\omega_1^{t+1} \in \Omega_1} \sum_{\omega_2^{t+1} \in \Omega_2} O\left(S^{t+1}, \langle \pi_1(\vec{\omega}_1^t), \pi_2(\vec{\omega}_2^t) \rangle, \langle \omega_1^{t+1}, \omega_2^{t+1} \rangle\right) \\ \cdot V_\pi^{t+1}\left(S^{t+1}, \langle \vec{\omega}_1^{t+1}, \vec{\omega}_2^{t+1} \rangle\right) \end{aligned} \tag{1}$$

At each time step, the computation of V_π^t performs a summation over all possible world states and agent observations, so the time complexity of this algorithm is $O\left((|S| \cdot |\Omega_1| \cdot |\Omega_2|)^T\right)$. The overall search performs this computation for each and every possible joint policy. Since each policy specifies different actions over possible histories of observations, the number of possible policies for an individual agent i is $O\left(|A_i|^{\frac{|\Omega_i|^T - 1}{|\Omega_i| - 1}}\right)$ The number of possible *joint* policies for n agents is thus $O\left(\left(|A_*|^{\frac{|\Omega_*|^T - 1}{|\Omega_*| - 1}}\right)^n\right)$,, where A_* and Ω_* correspond to the largest individual action and observation spaces, respectively, among the agents. The time complexity for finding the optimal joint policy by searching this space is thus: $O\left(\left(|A_*|^{\frac{|\Omega_*|^T - 1}{|\Omega_*| - 1}}\right)^n \cdot (|S| \cdot |\Omega_*|^n)^T\right)$

5 Joint Equilibrium-based Search for Policies

Given the complexity of exhaustively searching for the optimal joint policy, it is clear that such methods will not be successful when the amount of time to generate the policy is restricted. In this section, we will present algorithms that are guaranteed to find a locally optimal joint policy. We refer to this category of algorithms as "JESP" (Joint Equilibrium-Based Search for Policies). Just like the solution in Section 4, the solution obtained using JESP is a Nash equilibrium. In particular it is a locally optimal solution to a partially observable identical payoff stochastic game(POIPSG) [Peshkin *et al.,* 2000]. The key idea is to find the policy that maximizes the joint expected reward for one agent at a time, keeping the policies of all the other agents fixed. This process is repeated until an equilibrium is reached (local optimum is found). The problem of which optimum the agents should select when there are multiple local optima is not encountered since planning is centralized.

5.1 Exhaustive approach(Exhaustive-JESP)

The algorithm below describes an exhaustive approach for JESP. Here we consider that there are *n* cooperative agents. We modify the policy of one agent at a time keeping the policies of the other n - 1 agents fixed. The function bestPolicy, returns the joint policy that maximizes the expected joint reward, obtained by keeping *n* — 1 agents' policies fixed and exhaustively searching in the entire policy space of the agent whose policy is free. Therefore at each iteration, the value of the modified joint policy will always either increase or remain unchanged. This is repeated until an equilibrium is reached, i.e. the policies of all n agents remains unchanged. This policy is guaranteed to be a local maximum since the value of the new joint policy at each iteration is non-decreasing.

Algorithm 1 EXHAUSTIVE-JESPQ

```
1: prev ← random joint policy, conv ← 0
2: while conv ≠ n - 1 do
3:    for i ← 1 to n do
4:       fix policy of all agents except i
5:       policy Spac← list of all policies for i
6:       new ← bestPolicy(i,policySpace,prev)
7:       if new.value = prev.value then
8:          conv←conv + 1
9:       else
10:         prev ← new, conv ← 0
11:      if conv = n - 1 then
12:         break
13: return new
```

The best policy cannot remain unchanged for more than *n* — 1 iterations without convergence being reached and in the worst case, each and every joint policy is the best policy for at least one iteration. Hence, this algorithm has the same worst case complexity as the exhaustive search for a globally optimal policy. However, it could do much better in practice as illustrated in Section 6. Although the solution found by this algorithm is a local optimum, it may be adequate for some applications. Techniques like *random restarts* or *simulated annealing* can be applied to perturb the solution found to see if it settles on a different higher value.

The exhaustive approach to Steps 5 and 6 of the Exhaustive-JESP algorithm enumerates and searches the entire policy space of a single agent, *i.* There are $O\left(|A_i|^{\frac{|\Omega_i|^T-1}{|\Omega_i|-1}}\right)$h policies, and evaluating each incurs a time complexity of $O\left(\left(|S|\cdot|\Omega|^T\right)\right)$. Thus, using the exhaustive approach incurs an overall time complexity in Steps 5 and 6 of: $O\left(|A_i|^{\frac{|\Omega_i|^T-1}{|\Omega_i|-1}}|S|\cdot|\Omega|^T\right)$.. Since we incur this complexity cost in each and every pass through the JESP algorithm, a faster means of performing the bestPolicy function call of Step 6 would produce a big payoff in overall efficiency. We describe a dynamic programming alternative to this exhaustive approach for doing JESP next.

5.2 Dynamic Programming (DP-JESP)

If we examine the single-agent POMDP literature for inspiration, we find algorithms that exploit dynamic programming to incrementally construct the best policy, rather than simply search the entire policy space [Monahan, 1982; Cassandra *et al.*, 1997; Kaelbling *et* a/., 1998]. These algorithms rely on a *principle of optimality* that states that each sub-policy of an overall optimal policy must also be optimal. In other words, if we have a T-step optimal policy, then, given the history over the first *t* steps, the portion of that policy that covers the last *T* — *t* steps must also be optimal over the remaining *T*-t steps. In this section, we show how we can exploit an analogous optimality property in the multiagent case to perform more efficient construction of the optimal policy within our JESP algorithm.

To support such a dynamic-programming algorithm, we must define *belief states* that summarize an agent's history of past observations, so that they allow the agents to ignore the actual history of past observations, while still supporting construction of the optimal policy over the possible future. In the single-agent case, a belief state that stores the distribution, B^t_{single} $\Pr(S^t|\vec{\omega}^t)$, is a *sufficient statistic,* because the agent can compute an optimal policy based on B^t_{single} without having to consider the actual observation sequence, $\vec{\omega}^t$ [Sondik, 1971].

In the multiagent case, an agent faces a complex but normal single-agent POMDP if the policies of all other agents are fixed. However, B^t_{single}, is not sufficient, because the agent must also reason about the action selection of the other agents and hence on the observation histories of the other agents. Thus, at each time *t,* the agent *i* reasons about the tuple e^t_i — $\left\langle S^t, \vec{\omega}^t_{\neq i}\right\rangle$, where $\vec{\omega}^t_{\neq i} = \left\langle \vec{\omega}^t_1, \ldots, \vec{\omega}^t_{i-1}, \vec{\omega}^t_{i+1}, \ldots, \vec{\omega}^t_n\right\rangle$ is the joint observation histories of all the agents except *i.* By treating e\ to be the state of the agent *i* at time *t,* we can define the transition function and observation function for the single agent POMDP for agent *i* as follows:

$$\begin{aligned}P'(e^t_i, a^t_i, e^{t+1}_i) &= Pr(e^{t+1}_i|e^t_i, a^t_i)\\ &= P(S^t, (\pi_{\neq i}(\vec{\omega}^t_{\neq i}), a^t_i), S^{t+1})\\ &\quad \cdot\prod_{j\neq i} O_j(S^{t+1}, (\pi_{\neq i}(\vec{\omega}^t_{\neq i}), a^t_i), \omega^{t+1}_j) \quad (2)\end{aligned}$$

$$\begin{aligned}O'(e^{t+1}_i, a^t_i, \omega^{t+1}_i) &= Pr(\omega^{t+1}_i|e^{t+1}_i, a^t_i)\\ &= O_i(S^{t+1}, (\pi_{\neq i}(\vec{\omega}^t_{\neq i}), a^t_i), \omega^{t+1}_i) \quad (3)\end{aligned}$$

where $\pi_{\neq i} = \left\langle \pi^t_1, \ldots, \pi^t_{i-1}, \pi^t_{i+1}, \ldots, \pi^t_n\right\rangle$ is the joint policy for all agents except i.

We now define the novel *multiagent* belief state for an agent *i* given the distribution over the initial state, *b(s)* = *Pr*(S^1 = *s):*

$$B^t_i = \Pr(e^t_i|\vec{\omega}^t_i, \vec{a}^{t-1}_i, b) \quad (4)$$

In other words, when reasoning about an agent's policy in the context of other agents, we maintain a distribution over e^t_i, rather than simply the current state. Figure 1 shows different belief states B^1 , B^2 and B'^2 for agent 1 in the tiger domain. For instance, B^2 shows probability distributions over e^2_1.. In e^2_1 = *(SL, (HR))*9 *(HR)* is the history of agent 2's observations while SL is the current state. Section 5.3 demonstrates how we can use this multiagent belief state to construct a dynamic program that incrementally constructs the optimal policy for agent *i.*

5.3 The Dynamic Programming Algorithm

Following the model of the single-agent value-iteration algorithm, our dynamic program centers around a value function over a T-step finite horizon. For readability, this section presents the derivation for the dynamic program in the

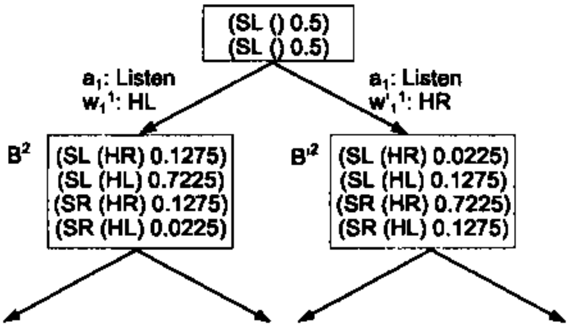

Figure 1: Trace of Tiger Scenario

two-agent case; the results easily generalize to the n-agent case. Having fixed the policy of agent 2, our value function, *Vt{B^t),* represents the expected reward that the team will receive when agent 1 follows its optimal policy from the t-th step onwards when starting with a current belief state, B_1^t. We start at the end of the time horizon (i.e., *t* = T), and then work our way back to the beginning. Along the way, we construct the optimal policy by maximizing our value function over possible action choices:

$$V_t(B_1^t) = \max_{a_1 \in A_1} V_t^{a_1}(B_1^t) \qquad (5)$$

We can define the action value function, $V_t^{a_1}$, recursively:

$$\begin{aligned} V_t^{a_1}(B_1^t) &= ER(a_1, B_1^t) + \sum_{\omega_1^{t+1} \in \Omega_1} \Pr(\omega_1^{t+1} | B_1^t, a_1) \\ &\quad \cdot V_{t+1}\left(B_1^{t+1}\right) \qquad (6) \end{aligned}$$

The first term in equation 6 refers to the expected immediate reward, while the second term refers to the expected future reward, B_1^{t+1} is the belief state updated after performing action a_1 and observing ω_1^{t+1}. In the base case, *t — T,* the future reward is 0, leaving us with:

$$V_T^{a_1}(B_1^T) = ER(a_1, B_1^T) \qquad (7)$$

The calculation of expected immediate reward breaks down as follows:

$$ER(a_1, B_1^t) = \sum_{e_1^t = \langle S^t, \vec{\omega}_2^t \rangle} B_1^t(e_1^t) \cdot R(S^t, (a_1, \pi_2(\vec{\omega}_2^t))) \qquad (8)$$

Thus, we can compute the immediate reward using only the agent's current belief state and the primitive elements of our given MTDP model (See Section 2).

Computation of the expected future reward (the second term in Equation 6) depends on our ability to update agent 1 's belief state from B_1^t to B_1^{t+1} in light of the new observation, ω_1^{t+1}. For example, in Figure 1, the belief state B^1 is updated to B'^2 on performing action a$_1$ and receiving observation $\omega_1'^1$. We now derive an algorithm for performing such an update, as well as computing the remaining $\Pr(\omega|B, a)$ term from Equation 6. The initial belief state based on the distribution over initial state, 6, is:

$$B^1(e_1^1) = b(S^1) \qquad (9)$$

For $e_1^{t+1} = \langle S^{t+1}, \langle \vec{\omega}_2^t, \omega_2^{t+1} \rangle \rangle$, the updated $B_1^{t+1}(e_1^{t+1})$ is obtained using Equations 2 and 3 and Bayes Rule and is given as follows:

$$\begin{aligned} B_1^{t+1}(e_1^{t+1}) &= \sum_{S^t} B_1^t(e_1^t) \cdot P(S^t, (a_1, \pi_2(\vec{\omega}_2^t)), S^{t+1}) \\ &\quad \cdot O_1(S^{t+1}, (a_1, \pi_2(\vec{\omega}_2^t)), \omega_1^{t+1}) \\ &\quad \cdot O_2(S^{t+1}, (a_1, \pi_2(\vec{\omega}_2^t)), \omega_2^{t+1}) / \\ &\quad Pr(\omega_1^{t+1} | B_1^t, a_1) \qquad (10) \end{aligned}$$

We treat the denominator of Equation 10 (i.e., Pr$(\omega_1^{t+1}|B_1^t, a_1))$ as a normalizing constant to bring the sum of the numerator over all e_1^t to be 1. This result also enters into our computation of future expected reward in the second term of Equation 6. Thus, we can compute the agent's new belief state (and the future expected reward and the overall value function, in turn) using only the agent's current belief state and the primitive elements of our given MTDP model. Having computed the overall value function, Vt, we can also extract a form of the optimal policy, π_1. that maps observation histories into actions, as required by Equations 8 and 10.

Algorithm 2 presents the pseudo-code for our overall dynamic programming algorithm. Lines 1-6 generate all of the belief states reachable from a given initial belief state, $b(s) = \Pr(S^1 = s)$.. Since there is a possibly unique belief state for every seąuence of actions and observations by agent 1, there are O $\left((|A_1| \cdot |\Omega_1|)^T\right)$ reachable belief states. This reachability analysís uses our belief update procedure (Algorithm 3), which itself has time complexity $O(|S|^2|\Omega_2|^{t+1})$ when invoked on a belief state at time *t.* Thus, the overall reachability analysis phase has a time complexity of O$\left((|S|^2(|A_1| \cdot |\Omega_1| \cdot |\Omega_2|)^T\right)$. Lines 7-22 perform the heart of our dynamic programminq alqorithm,. which also has a time complexity of 0($\left(S^2(|A_1| \cdot |\Omega_1| \cdot |\Omega_2|)^T\right)$.. Lines 23-27 translate the resulting value functíon ínto an agent policy defined over observation sequences, as required by our algorithm (i.e., the π_2 argument). This last phase has a lower time and space complexity, O$(|S|^2|\Omega_1|^T \cdot |\Omega_2|^T)$, than our other two phases, since it considers only optimal actions for agent 1. Thus, the overall time complexity of our algorithm is O $\left(S^2(|A_1| \cdot |\Omega_1| \cdot |\Omega_2|)^T\right)$. The space complexity of the resulting value function and policy is essentially the product of the number of reachable belief states and the size of our belief state representation: O $\left(|S|(|A_1| \cdot |\Omega_1| \cdot |\Omega_2|)^T\right)$.

5.4 Piecewise Linearity and Convexity of Value Function

Algorithm 2 computes a value function over only those belief states that are reachable from a given initial belief state, which is a subset of all possible probability distributions over S^t and $\vec{\omega}_2^t$. To use dynamic programming over the entire set, we must show that our chosen value function is piecewise linear and convex (PWLC). Each agent is faced with solving a single agent POMDP is the policies of all other agents is fixed as shown in Section 5.2. Sondik [1971] showed that the value function for a single agent POMDP is PWLC. Hence the value function in Equation 5 is PWLC. Thus, in addition

Algorithm 2 OPTIMALPOLICYDP(b, π_2, T)

```
$reachable(0) \leftarrow \{b\}$
for $t \leftarrow 1$ to $T$ do
    for all $B^{t-1} \in reachable(t-1)$ do
        $reachable(t) \leftarrow \emptyset$
        for all $a_1 \in A_1, \omega_1 \in \Omega_1$ do
            $reachable(t) \overset{\cup}{\leftarrow}$ UPDATE($B^{t-1}, a_1, \omega_1$)
for $t \leftarrow T$ downto 1 do
    for all $B^t \in reachable(t)$ do
        $V_t(B^t) \leftarrow -\infty$
        for all $a_1 \in A_1$ do
            $V_t^{a_1}(B^t) \leftarrow 0$
            for all $s \in S, \vec{\omega}_2 \in \Omega_2^t$ do {Equation 8}
                $V_t^{a_1}(B^t) \overset{+}{\leftarrow} B^t(s, \vec{\omega}_2) \cdot R(s, \langle a_1, \pi_2(\vec{\omega}_2)\rangle)$
            if $t < T$ then {Compute future reward}
                for all $\omega_1 \in \Omega_1$ do
                    $prob \leftarrow 0$
                    for all $s^t, s^{t+1} \in S, \vec{\omega}_2 \in \Omega_2^t$ do
                        $act \leftarrow \langle a_1, \pi_2(\vec{\omega}_2)\rangle$
                        $prob \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2) \cdot P(s^t, act, s^{t+1}) \cdot$
                        $O_1(s^{t+1}, act, \omega_1)$
                    $V_t^{a_1}(B^t) \overset{+}{\leftarrow} prob \cdot$
                    $V_{t+1}($UPDATE$(B^t, a_1, \omega_1))$ {Equation 6}
            if $V_t^{a_1}(B^t) > V_t(B^t)$ then
                $V_t(B^t) \leftarrow V_t^{a_1}(B^t)$
for all $\vec{\omega}_1 \in \Omega_1^T$ do
    $B^0 \leftarrow b$
    for $t \leftarrow 1$ to $T$ do
        $B^t \leftarrow$ UPDATE$(B^{t-1}, \pi_1(\vec{\omega}_1[1 \ldots (t-1)]), \vec{\omega}_1[t])$
        $\pi_1(\vec{\omega}_1[1 \ldots t]) \leftarrow \arg\max_{a_1} V_t^{a_1}(B^t)$
return $\pi_1$
```

Algorithm 3 UPDATE(B^t, a_1, ω_1)

```
for all $s^{t+1} \in S, \vec{\omega}_2 \in \Omega_2^{t+1}$ do
    $B^{t+1}(s^{t+1}, \vec{\omega}_2) \leftarrow 0$
    $act \leftarrow \langle a_1, \pi_2(\vec{\omega}_2[1 \ldots t])\rangle$
    for all $s^t \in S$ do {Equation 10}
        $B^{t+1}(s^{t+1}, \vec{\omega}_2) \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2[1 \ldots t]) \cdot$
        $P(s^t, act, s^{t+1}) \cdot O_1(s^{t+1}, act, \omega_1) \cdot$
        $O_2(s^{t+1}, act, \vec{\omega}_2[t+1])$
    normalize $B^{t+1}(s^{t+1}, \vec{\omega}_1)$
return $B^{t+1}$
```

to supporting the more efficient dynamic programming of Algorithm 2, our novel choice of belief state space and value function can potentially support a dynamic programming algorithm over the entire continuous space of possible belief states.

6 Experimental Results

In this section, we perform an empirical comparison of the algorithms described in Sections 4 and 5 using the Tiger Scenario (Sec Section 3) in terms of time and performance. Figure 2, shows the results of running the globally optimal algorithm and the Exhaustive JESP algorithm for two different reward functions (Tables 3 and 4. Finding the globally optimal policy is extremely slow and is doubly exponential in the finite horizon, T and so we evaluate the algorithms only for finite horizons of 2 and 3. We ran the JESP algorithm for 3 different randomly selected initial policy settings and compared the performance of the algorithms in terms of the number of policy evaluations (on Y-axis using log scale) that were necessary. As can be seen from this figure, for T — 2 the JESP algorithm requires much fewer evaluations to arrive at an equilibrium. The difference in the run times of the globally optimal algorithm and the JESP algorithm is even more apparent when T = 3. Here the globally optimal algorithm performed > 4 million policy evaluations while the JESP algorithm did < 7000 evaluations. For the reward function A, JESP succeeded in finding the globally optimal policies for both T = 2 (expected reward = -4) and 3 (expected reward = —6). However, this is not always the case. Using reward function B for T = 2, the JESP algorithm sometimes settles on a locally optimal policy (expected reward = -4) that is different from the globally optimal policy (expected reward = 20). However, when random restarts are used, the globally optimal reward can be obtained.

Based on Figure 2, we can conclude that the exhaustive JESP algorithm performs better than an exhaustive search for the globally optimal policy but can some times settle on a policy that is only locally optimal. This could be sufficient for problems where the difference between the locally optimal policy's value and the globally optimal policy's value is small and it is imperative that a policy be found quickly. Alternatively, the JESP algorithm could be altered so that it doesn't get stuck in a local optimum via random restarts.

Table 5 compares presents experimental results from comparison of exhaustive JESP with our dynamic programming approach (DP-JESP). These results, also from the tiger domain, show run-time in milliseconds (ms) for the two algorithms with increasing horizon. DP-JESP is seen to obtain significant speedups over exhaustive-JESP. For time horizon of 2 and 3 DP-JESP run time is essentially 0 ms, compared to the significant run times of Exhaustive-JESP. As we increased the horizon to > 3, we could not run exhaustive-JESP at all; while DP-JESP could be easily run up to horizon of 7.

7 Summary and Conclusion

With the growing importance of decentralized POMDPs in the multiagents arena, for both design and analysis, it is critical to develop efficient algorithms for generating joint poli-

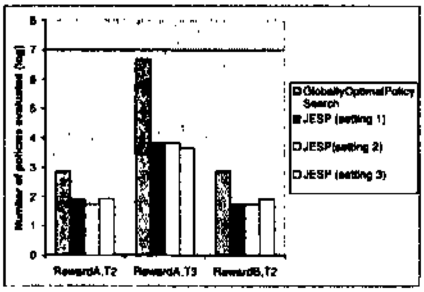

Figure 2: Evaluation Results

Method	2	3	4	5	6	7
Exhaustivc-JESP	10	317,800				
DP-JESP	0	0	20	110	1,360	30,030

Table 5: Run time(ms) for various T with Pentium 4, 2.0GHz, 1GB memory, Linux Redhat 7.1, Allegro Common Lisp 6.0

cies. Yet, there is a significant lack of such efficient algorithms. There are three novel contributions in this paper to address this shortcoming. First, given the complexity of the exhaustive policy search algorithm — doubly exponential in the number of agents and time — we describe a class of algorithms called "Joint Equilibrium-based Search for Policies" (JESP) that search for a local optimum rather than a global optimum. In particular, we provide detailed algorithms for Exhaustive JESP and dynamic programming JESP(DP-JESP). Second, we provide complexity analysis for DP-JESP, which illustrates a potential for exponential speedups over exhaustive JESP. We have implemented all of our algorithms, and empirically verified the significant speedups they provide. Third, we provide a proof that the value function for individual agents is piece-wise linear and convex (PWLC) in their belief states. This key result could pave the way to a new family of algorithms that operate over continuous belief states, increasing the range of applications that can be attacked via decentralized POMDPs, and is now a major issue for our future work.

Acknowledgments

We thank Piotr Gmytrasiewicz for discussions related to the paper. This research was supported by NSF grant #0208580 and DARPA award no. F30602-98-2-0108.

References

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

[Boutilier, 1996] C. Boutilier. Planning, learning & coordination in multiagent decision processes. In *TARK-96,* 1996.

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

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

[Kaelbling et al., 1998] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence,* 101, 1998.

[Monahan, 1982] G. Monahan. A survey of partially observable markov decision processes: Theory, models and algorithms. *Management Science,* 101(1): 1-16, January 1982.

[Papadimitriou and Tsitsiklis, 1987] C. Papadimitriou and J. Tsitsiklis. Complexity of markov decision processes. *Mathematics ofOperatios Research,* 12(3):441-450, 1987.

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

[Pynadath and Tambe, 2002] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *JAIR,* 2002.

[Sondik, 1971] Edward J. Sondik. The optimal control of partially observable markov processes. *Ph.D. Thesis, Stanford,* 1971.

[Xuan et al., 2001] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multiagent cooperation. In *Agents-01, 2001.*