Communication for Improving Policy Computation in Distributed POMDPs

Ranjit Nair, Milind Tambe
Computer Science Dept.
Univ. of Southern California
{nair,tambe}@usc.edu

Maayan Roth
Robotics Institute
Carnegie Mellon University
mroth@andrew.cmu.edu

Makoto Yokoo
Dept. of Intelligent Systems
Kyushu University
yokoo@is.kyushu-u.ac.jp

Abstract

Distributed **P***artially* **O***bservable* **M***arkov* **D***ecision* **P***roblems (POMDPs) are emerging as a popular approach for modeling multiagent teamwork where a group of agents work together to jointly maximize a reward function. Since the problem of finding the optimal joint policy for a distributed POMDP has been shown to be NEXP-Complete if no assumptions are made about the domain conditions, several locally optimal approaches have emerged as a viable solution. However, the use of communicative actions as part of these locally optimal algorithms has been largely ignored or has been applied only under restrictive assumptions about the domain. In this paper, we show how communicative acts can be explicitly introduced in order to find locally optimal joint policies that allow agents to coordinate better through synchronization achieved via communication. Furthermore, the introduction of communication allows us to develop a novel* compact *policy representation that results in savings of both space and time which are verified empirically. Finally, through the imposition of constraints on communication such as not going without communicating for more than K steps, even greater space and time savings can be obtained.*

1. Introduction

Multiagent systems are increasingly being applied to domains like disaster rescue where the performance is linked to critical metrics like loss of property and human life [8]. Moreover, in these realistic domains, the agents in the team need to work together in the presence of uncertainty arising from various sources like partial observability, imperfect sensing, failure of agents, etc. Distributed **P**artially **O**bservable **M**arkov **D**ecision **P**roblems (POMDPs) are emerging as a popular approach for modeling multiagent teamwork in such domains where a groups of agents work together to jointly maximize a reward function [7, 2, 12, 4, 9].

Unfortunately as shown by Bernstein *et al.*, the problem of finding the optimal joint policy for a distributed POMDP is NEXP-Complete if no assumptions are made about the domain conditions[2]. There are two prominent approaches to dealing with this large complexity. The first approach involves approximating the domain, e.g., abstracting away from agents' interactions (transition independence) [1], or approximating to complete observability of local state [4], or assuming memory-less agents [3].

The second approach involves finding a locally optimal joint policy without approximating the domain. For instance, Peshkin *et al.* [10] use gradient descent search to find locally optimal finite-controllers with bounded memory. Their algorithm finds locally optimal policies from a limited subset of policies, with an infinite planning horizon. On the other hand, Nair *et al.* [7] develop an algorithm called "Joint Equilibrium-based Search for Policies" (JESP) which finds locally optimal policies from an unrestricted set of possible policies, with a finite planning horizon. 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. The work in this paper is described in the context of JESP but can potentially be applied to other distributed POMDP approaches.

One key challenge in finding locally (or globally) optimal policies for distributed POMDPs is as follows: the central planner, when generating a policy for one agent must reason explicitly about the possible observations that other agents are receiving – since other agents' observations will determine their actions, and hence the team's joint reward. Such reasoning about observations is a fundamental reason for intractability of distributed POMDPs. Furthermore, uncertainty about other agents' observations can hamper inter-agent coordination.

We take steps in addressing the above challenge by introducing communicative acts that not only improve agents' coordination and consequently their expected rewards, but also reduce the complexity of the computation of joint policies. The use of communication to facilitate policy computation has been largely ignored or has been applied only under restrictive assumptions about the domain [12, 4]. The synchronization achieved via communication allows us to develop a novel *compact* policy representation that results in savings of both space and time which are verified empirically. Finally, through the imposition of constraints on communication such as requiring that agents do not go for more than K steps without communication, even greater space and time savings can be obtained.

2. Model

We utilize the Markov Team Decision Problem (MTDP) [11] as a concrete illustration of a distributed POMDP model and extend it to include communication. Our approach can be applied to other distributed POMDP models [2, 12].

Given a team of n agents, an MTDP [11] is defined as a tuple: $\langle S, A, P, \Omega, O, R\rangle$. S is a finite set of world states $\{s_1, \ldots, s_m\}$. $A = \times_{1\leq i\leq n} A_i$, 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 $\langle a_1, \ldots, a_n\rangle$. $\Omega = \times_{1\leq i\leq n}\Omega_i$ 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. 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.

Agents start with the same probability distribution over the start state. However, they will receive different observations. As a result, although agent i knows the policies of other agents, considerable uncertainty can arise about what other agents are going to do, since the actions of other agents depend on the observation histories of these agents, which are not accessible to agent i. If agents communicate and exchange their observation histories, such uncertainty disappears. However, communication requires certain cost. To act optimally, an agent needs to estimate the benefit of communication and should communicate only when the benefit exceeds the communication cost.

Hence, we extend MTDP by introducing a new communicative action, *Sync*, that can be initiated by any agent. Unlike other models, like COM-MTDP [11] and Dec_POMDP_Com [4], where there are alternate communication and action phases, we do not assume a separate communication phase. In a particular epoch an agent can either choose to communicate or act. This assumption models the missed opportunity cost that occurs when the agents communicate instead of acting. However, it is not central to this paper and the algorithms can be easily modified for models with alternating communication and action phases.

If one agent initiates a *Sync*, the other agents are forced to communicate with it, ignoring the action that they would otherwise have performed. On performing a *Sync*, or on receiving notification that another agent has performed a *Sync*, the agents exchange all their observation histories since the last *Sync* action. Given that the agents know each other's policies, a *Sync* action results in all agents knowing exactly what action each agent will perform in the next epochin, thus preventing miscoordination.

We assume that the entire process of one agent initiating a *Sync* followed by all the agents sharing their observation histories takes place in a single epoch. At a finer granularity, we can think of every domain-level action as having two phases – an arbitrarily small "wait-for-interrupt" phase followed by an action phase. Similarly, a *Sync* action is comprised of a "send-interrupt", followed by a communication. If an agent receives a *Sync* interrupt during its "wait-for-interrupt" phase, it is forced to communicate instead of acting. We further assume that *Sync* never fails and has no effect on the world state.

3. Domain

For illustrative purposes, we consider a multiagent adaptation of the classic tiger problem used in illustrating single-agent POMDPs[5]. 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. The state, S, takes values $\{SL, SR\}$, indicating the door behind which the tiger is present. In the initial state, the tiger is equally likely to be behind each door. 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 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 function, O_1 or O_2, shown in Table 1, are identical and 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 independently receives the observation HL with probability 0.85 and HR with probability 0.15.

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

Table 1. Observation function for each agent

If either agent opens the door behind which the tiger is present, they are both attacked (equally) by the tiger (see Table 2). However, the injury sustained if they jointly opened the door to the tiger is less severe than if only one agent opens the door. Similarly, if both agents open the door to the riches, the amount of wealth received is twice what they would have received if only one of them opened that door. The agents incur a small cost for performing the $`Listen'$ action.

Clearly, acting jointly is beneficial (e.g. $A_1 = A_2 = `OpenLeft'$) because the agents receive more riches and sustain less damage by acting together. However, because the agents receive independent observations and cannot share these observations without explicit communication, to act optimally, each agent must consider all possible observation histories of the other agent to determine which action its teammate is likely to perform. This can result in mis-coordinated actions, such as one agent opening the left door while the other agent opens the right door. In order to reduce the likelihood of mis-coordination, we introduce the *Sync* action, allowing the agents to share their observation histories.

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
<Sync,*>	-2	-2
<*,Sync>	-2	-2

Table 2. Reward function

In keeping with the constraints of real-world communication, the *Sync* action is given a cost, $SyncCost$, which is incurred every time an agent chooses to communicate.

4. JESP

As shown by Bernstein *et al.* [2] the complexity of the decision problem corresponding to finding the globally optimal policy for a distributed POMDP is NEXP-complete if no assumptions are made about the domain conditions. Given this high complexity, locally optimal approaches [10, 3, 7] have emerged as viable solutions. In this paper, we concentrate on "JESP" (Joint Equilibrium-Based Search for Policies) [7], an approach where the solution obtained is a Nash equilibrium. Algorithm 1 describes the JESP approach for 2 agents. The algorithm can be easily modified for n agents. The key idea is to find the policy that maximizes the joint expected reward for one agent at a time, keeping the policies of the other agent 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. DP-JESP is a dynamic programming approach for finding the optimal policy for a single agent relative to the fixed policies of its n-1 teammates(line 4). We briefly describe the DP-JESP algorithm in the following sub-section before introducing communication.

4.1. 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 searching the entire policy space [6, 5]. The key insight in the multiagent case is

Algorithm 1 DP-JESP()

```
1: $prev \leftarrow$ default joint policy, $prevVal \leftarrow$ value of $prev$, $conv \leftarrow 0$
2: while $conv \neq 2$ do
3:   for $i \leftarrow 1$ to $2$ do
4:     $val, \Pi_i \leftarrow$ OptimalPolicyDP$(b, \Pi_{(i+1)\text{Mod}2}, T)$
5:     if $val = prevVal$ then
6:       $conv \leftarrow conv + 1$
7:     else
8:       $prev \leftarrow \left\langle \Pi_i, \Pi_{(i+1)\text{Mod}2} \right\rangle$, $prevVal \leftarrow val$, $conv \leftarrow 0$
9:     if $conv = 2$ then
10:      break
11: return new
```

that if the policies of all other agents are fixed, then the free agent faces a complex but normal single-agent POMDP. However, a belief state that stores the distribution, $\Pr(s^t|\vec{\omega}^t)$, as in the single-agent POMDP case, is not a *sufficient statistic* because the agent must also reason about the action selection of the other agents and hence about the observation histories of the other agents. Thus, at each time t, agent 1 reasons about the 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 single-agent POMDP for agent 1 as follows:

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

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

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

$$B_1^t = \Pr(e_1^t|\vec{\omega}_1^t, \vec{a}_1^{t-1}, b) \quad (3)$$

In other words, when reasoning about an agent's policy in conjunction with its teammate agent, we maintain a distribution over e_1^t, rather than simply over the current state. However, since the agent does not know exactly what observations the other agent has received at run time, it will not be able to know precisely which action the other agent will execute. Hence, we introduce *Sync* action, allowing the agents to periodically synchronize, thus reducing the likelihood of mis-coordination.

5. Communicative DP-JESP

When one of the agents performs *Sync*, the agents share their observation histories with each other. The agents are now said to have a *synchronized belief state*. (The distribution over initial state is considered the first synchronized belief state). As described in the previous section, in order to reason about agent 1's policy in the context of agent 2, we maintain a distribution over e_1^t, rather than simply over the current state. When agents are in a *synchronized belief state*, they can discard their observation histories. Figure 1 shows a sample progression of belief states for agent 1 in the tiger domain. For instance, B_1^2, shows probability distributions over e_1^2. In $e_1^2 = (SL, (HR))$, (HR) is the history of agent 2's observations while SL is the current state. If the action specified in B_1^2 is *Sync*, then the agents reach a synchronized belief state and can get rid of their observation histories.

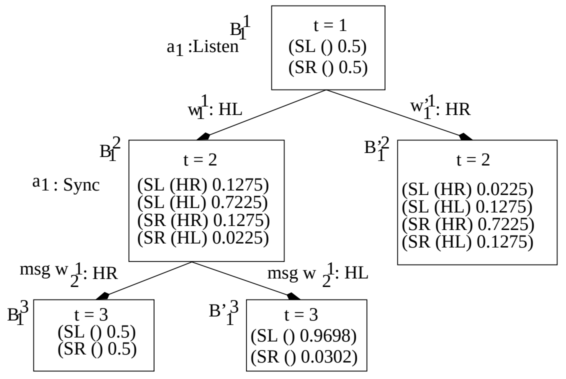


Figure 1. Trace of Tiger Scenario

Because agents can discard their observation histories after synchronization, we introduce a novel *compact* policy representation. We now refer to the policy of an agent 1 as Π_1. $\pi_1 = \Pi_1(b)$ refers to agent 1's sub-policy indexed by the last synchronized belief state, b and $\pi_1(\vec{\omega}_1, T-t)$ is the action agent 1 will perform at time t. Algorithm 8 shows how a policy using this representation can be obtained.

Having fixed the policy of agent 2, the optimal policy for agent 1 can be computed using Algorithm 2.

Algorithm 2 OptimalPolicyDP(b, Π_2, T)

```
1: $val \leftarrow$ GetValue$(b, b, \Pi_2, 1, T)$
2: initialize $\Pi_1$
3: FindPolicy$(b, b, \langle\rangle, \Pi_2, 1, T)$
4: return $val$,$\Pi_1$
```

Following the model of the single-agent value-iteration algorithm, central to our dynamic program is the value function over a T-step finite horizon. The value function, $V_t(B^t, b)$ (see Algorithm 3), represents the expected reward that the team will receive, starting from the most recent synchronized belief state, b, and with agent 1 following an optimal policy from the t-th step onwards. 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 the value function over possible action choices:

$$V_t(B_1^t, b) = \max_{a_1 \in A_1} V_t^{a_1}(B_1^t, b) \tag{4}$$

Algorithm 3 GETVALUE(B^t, b, Π_2, t, T)

```
1:  if $t > T$ then
2:      return 0
3:  if $V_t(B^t, b)$ is already recorded then
4:      return $V_t(B^t, b)$
5:  $best \leftarrow -\infty$
6:  for all $a_1 \in A \cup \{Sync\}$ do
7:      $value \leftarrow$ GETVALUEACTION($B^t, a_1, b, \Pi_2, t, T$)
8:      record $V_t^{a_1}(B^t, b)$ as value
9:      if $value > best$ then
10:         $best \leftarrow value$
11: record $V_t(B^t, b)$ as $best$
12: return $best$
```

The function, $V_t^{a_1}$, can be computed using Algorithm 4. This procedure can be defined separately for the cases of $a_1 = Sync$ (lines 1-7) and for all other domain actions (lines 8-24). If $a_1 = Sync$:

$$\begin{aligned} V_t^{a_1=Sync}(B_1^t, b) &= SyncCost + \sum_{\vec{\omega}_2} \Pr(\vec{\omega}_2 | B_1^t) \\ &\quad \cdot V_{t+1}(b', b'), \\ &\quad \text{where b' is the belief state after msg } \vec{\omega}_2 \end{aligned} \tag{5}$$

The quantity $\Pr(\vec{\omega}_2|B_1^t)$ refers to the probability of receiving a message $\vec{\omega}_2$ from agent 2 as a result of a *Sync*. This value and the resulting synchronized belief state b' is obtained using Algorithm 5.

In the case where $a_1 \neq Sync$, it is possible that agent 2 chooses to perform a *Sync*. Taking this into consideration while defining the action value function for the case where the action $a_1 \neq Sync$, we get:

$$\begin{aligned} V_t^{a_1 \neq Sync}(B_1^t, b) &= V_t^{a_1 \neq Sync, a_2 = Sync}(B_1^t, b) \\ &\quad + V_t^{no\ Sync}(B_1^t, b) \end{aligned} \tag{6}$$

The first term in Equation 6 is the vlaue obtained when agent 2 does a *Sync*, thus over-riding agent 1's action while the second term is the value obtained when neither agent performs *Sync*. When agent 2 does a *Sync*, the value function is:

$$\begin{aligned} V_t^{a_1 \neq Sync, a_2 = Sync}(B_1^t, b) &= \sum_{\vec{\omega}_2 \ s.t.\ \pi_2(\vec{\omega}_2) = Sync} \Pr(\vec{\omega}_2 | B_1^t) \\ &\quad \cdot (SyncCost + V_{t+1}(b', b')), \\ &\quad \text{where b' is the belief state after msg } \vec{\omega}_2 \end{aligned} \tag{7}$$

Algorithm 6 is used for computing the probability that agent 2 communicates the message $\vec{\omega}_2$, $\Pr(\vec{\omega}_2|B_1^t)$), and the synchronized belief state b' that results from this communication.

In the case where neither agent 1 nor agent 2 performs a *Sync*, the value function is defined as follows:

$$\begin{aligned} V_t^{no\ Sync}(B_1^t, b) = &\sum_{e^t = \langle s^t, \vec{\omega}_2 \rangle \ s.t.\ \pi_2(\vec{\omega}_2) \neq Sync} B^t(e^t) \cdot \left(R\left(s^t, \langle a_1, \pi_2(\vec{\omega}_2, T - t)\rangle\right)\right. \\ &\left. + \sum_{\omega_1^{t+1} \in \Omega_1} \Pr(\omega_1^{t+1} | B_1^t, a_1) \cdot V_{t+1}\left(B_1^{t+1}\right)\right) \end{aligned} \tag{8}$$

The first term (computed in lines 13-16 of Algorithm 4) in equation 8 refers to the expected immediate reward, while the second term (computed in lines 17-24 of Algorithm 4) 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} and is computed using Algorithm 7.

Finally, Algorithm 8 shows us how we can convert the value function from Equation 4 into the corresponding policy in the compact representation.

5.1. Communication with constraints

The algorithm presented in Section 5 shows how to find the locally optimal joint policy with no restrictions imposed on how often the agents should communicate. In the current sub-section, we impose constraints on the communication policy, forcing the agents to not go more than K steps without communicating. Thus no policy $\pi_1 \in \Pi_1$ can be indexed by an observation history of length greater than K. In computing the value function too, no episode can have a observation history of length greater than K. Thus by fixing K, we can limit the space requirements of the program.

Such constraints on K can be imposed by making small changes to the Algorithm 3 so that a count is maintained of the number of steps without a *Sync*. Section 6 shows detailed empirical results to justify the use of such constraints by highlighting the savings in run time and the increase in the complexity of the problems that can be solved.

6. Experimental Results

For the experiments in this section, we considered the scenario in Section 3. For our very first experiment,

Algorithm 4 GetValueAction(B^t, a, b, Π_2, t, T)

```
1: if $a = \text{Sync}$ then
2:     $value \leftarrow SyncCost$
3:     $reachable(B^t) \leftarrow \text{ComUpdate}(B^t)$
4:     for all $\langle B^{t+1}, prob \rangle \in reachable(B^t)$ do
5:         $value \overset{+}{\leftarrow} prob \cdot \text{GetValue}(B^{t+1}, B^{t+1}, \Pi_2, t+1, T)$
6:     return $value$
7: else
8:     $\pi_2 \leftarrow \Pi_2(b)$
9:     $value \leftarrow 0$
10:    $reachable(B^t) \leftarrow \text{ComUpdateOther}(B^t, \pi_2)$
11:    for all $\langle B^{t+1}, prob \rangle \in reachable(B^t)$ do
12:        $value \overset{+}{\leftarrow} prob \cdot [SyncCost + \text{GetValue}(B^{t+1}, B^{t+1}, \Pi_2, t+1, T)]$
13:    for all $e^t = \langle s^t, \vec{\omega}_2 \rangle$ s.t. $B^t(e^t) > 0$ do
14:        $a_2 \leftarrow \pi_2(\vec{\omega}_2, T-t)$
15:        if $a_2 \neq \text{Sync}$ then
16:            $value \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2) \cdot R\left(s^t, \langle a_1, \pi_2(\vec{\omega}_2, T-t) \rangle\right)$
17:    for all $\omega_1 \in \Omega_1$ do
18:        $B^{t+1} \leftarrow \text{Update}(B^t, a, \omega_1, \pi_2)$
19:        $prob \leftarrow 0$
20:        for all $s^t, e^{t+1} = \langle s^{t+1}, \vec{\omega}_2 \rangle$ s.t. $B^{t+1}(e^{t+1}) > 0$ do
21:            $a_2 \leftarrow \pi_2(\vec{\omega}_2, T-t)$
22:            if $a_2 \neq \text{Sync}$ then
23:                $prob \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2) \cdot P(s^t, \langle a_1, a_2 \rangle, s^{t+1}) \cdot O_1(s^{t+1}, \langle a_1, a_2 \rangle, \omega_1)$
24:        $value \overset{+}{\leftarrow} prob \cdot \text{GetValue}(B^{t+1}, b, \Pi_2, t+1, T)$
25:    return $value$
```

Algorithm 5 ComUpdate(B^t)

```
1: $reachable(B^t) \leftarrow \emptyset$
2: for all $e^t = \langle s^t, \vec{\omega}_2 \rangle$ s.t. $B^t(e^t) > 0$ do
3:     $prob(\vec{\omega}_2) \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2)$
4: for all $\vec{\omega}_2$ do
5:     for all $e^t = \langle s^t, \vec{\omega}_2 \rangle$ s.t. $B^t(e^t) > 0$ do
6:         $s^{t+1} \leftarrow s^t$
7:         $B^{t+1}(s^{t+1}, \langle\rangle) \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2)$
8:     normalize $B^{t+1}$
9:     $reachable(B^t) \overset{\cup}{\leftarrow} \langle B^{t+1}, prob(\vec{\omega}_2) \rangle$
10: return $reachable(B^t)$
```

Algorithm 6 ComUpdateOther(B^t, π_2)

```
1: $reachable(B^t) \leftarrow \emptyset$
2: for all $e^t = \langle s^t, \vec{\omega}_2 \rangle$ s.t. $B^t(e^t) > 0$ do
3:     $prob(\vec{\omega}_2) \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2)$
4: for all $\vec{\omega}_2$ do
5:     if $\pi_2(\vec{\omega}_2, T-t) = \text{Sync}$ then
6:         for all $e^t = \langle s^t, \vec{\omega}_2 \rangle$ s.t. $B^t(e^t) > 0$ do
7:             $s^{t+1} \leftarrow s^t$
8:             $B^{t+1}(s^{t+1}, \langle\rangle) \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2)$
9:     normalize $B^{t+1}(s^{t+1}, \langle\rangle)$
10:    $reachable(B^t) \overset{\cup}{\leftarrow} \langle B^{t+1}, prob(\vec{\omega}_2) \rangle$
11: return $reachable(B^t)$
```

Algorithm 7 Update(B^t, a, ω_1, π_2)

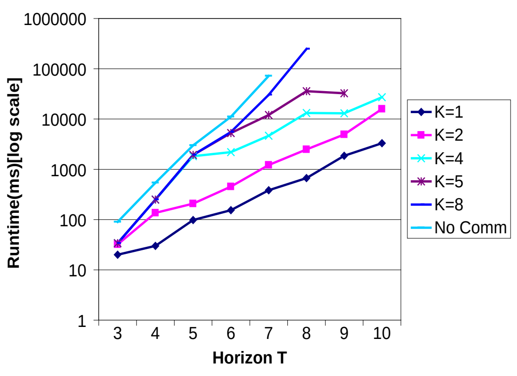
```
1: for all $e^{t+1} = \langle s^{t+1}, \langle \vec{\omega}_2, \omega_2 \rangle \rangle$ do
2:     $B^{t+1}(e^{t+1}) \leftarrow 0$
3:     $a_2 \leftarrow \pi_2(\vec{\omega}_2, T-t)$
4:     if $a_2 \neq \text{Sync}$ then
5:         for all $s^t \in S$ do
6:             $B^{t+1}(e^{t+1}) \overset{+}{\leftarrow} B^t(s^t, \vec{\omega}_2) \cdot P(s^t, \langle a_1, a_2 \rangle, s^{t+1}) \cdot O_1(s^{t+1}, \langle a_1, a_2 \rangle, \omega_1) \cdot O_2(s^{t+1}, \langle a_1, a_2 \rangle, \omega_2)$
7: normalize $B^{t+1}$
8: return $B^{t+1}$
```

Algorithm 8 FindPolicy($B^t, b, \vec{\omega_1}, \Pi_2, t, T$)

```
1: if $t > T$ then
2:     return
3: $a^* \leftarrow \arg\max_{a_1} V^t_{a_1}(B^t, b)$
4: $\pi_1 \leftarrow \Pi_1(b)$
5: $\pi_1(\vec{\omega_1}, T-t) \leftarrow a^*$
6: if $a^* = Sync$ then
7:     $B^{t+1} \leftarrow ComUpdate(B^t)$
8:     FindPolicy($B^{t+1}, B^{t+1}, \langle\rangle, \Pi_2, t+1, T$)
9: else
10:    $\pi_2 \leftarrow \Pi_2(b)$
11:    $B^{t+1} \leftarrow ComUpdateOther(B^t, \pi_2)$
12:    FindPolicy($B^{t+1}, B^{t+1}, \langle\rangle, \Pi_2, t+1, T$)
13:    for all $\omega_1 \in \Omega_1$ do
14:        $B^{t+1} \leftarrow Update(B^t, a^*, \omega_1, \pi_2)$
15:        FindPolicy($B^{t+1}, b, \langle \vec{\omega_1}, \omega_1 \rangle, \Pi_2, t+1, T$)
16: $\Pi_1(b) \leftarrow \pi_1$
```

we compared the run times for finding a locally optimal policy for various time horizons using various different communication policies. In Figure 2(a), X-axis shows the time horizon T, while the Y-axis shows the run time in milliseconds on a logarithmic scale. When K is equal to the time horizon, T, the communication is said to be unrestricted. As can be seen by comparing *No Comm* (No Communication) with unrestricted communication ($K = T$), introducing communication results in substantial savings in terms of run time. For instance at $K = T = 7$, there is more than a two-fold speedup for unrestricted communication over *No Comm.* These speed-ups become even more significant when we impose a constraint that the agents cannot go for more than K epochs without communicating. Also note that the gradients of the curves in Figure 2(a) are different, so the differences become exponentially large as T increases. Figure 2(b) shows that the run time (Y-axis) for finding the joint policy increases as K is increased(X-axis) for $T = 7$. Thus, we find that the run time increases as the amount of communication is reduced. The speed-ups in terms of run time are more pronounced for low K, e.g. at $T = 7$, there is a 37-fold speed-up for $K = 3$ over *No Comm.* All the run time values in Figure 2 are averaged over five runs to account for any variability in the run time environment.

Introduction of communication, also has a significant impact on memory requirements. This is evidenced by the fact that the program runs out of memory for $T > 7$ using *No Comm* but can run for $T = 10$ using $K = 4$.



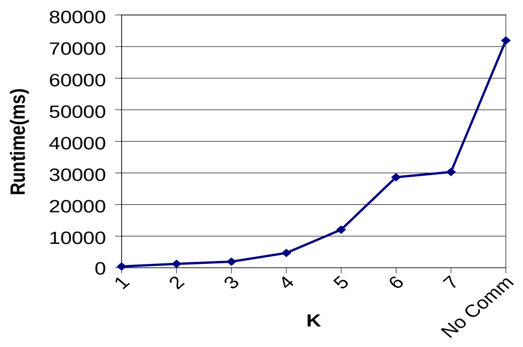


Figure 2. Run time(ms) for finding optimal joint policy for different K and no communication with Athlon, 1.8GHz, 4GB memory, Linux Redhat 7.3, Allegro Common Lisp 6.2, a) (top) varying T (log scale), b) (bottom) T=7

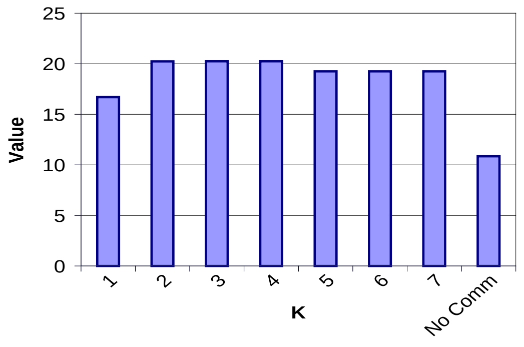
In our next experiment, we tried to determine the impact that the communication strategy has on the value of the joint policy that the program settles on. As explained in Section 4, the value of the joint policy at equilibrium depends on the starting policy that the JESP loop starts with. In Figure 3, we show the value obtained for various communication policies over a finite horizon of 7 using two different starting policies. Figure 3(a) has a policy with *Sync* when $t = K+1, 2K+2, 3K+3$, etc. and *Listen* at all other decision epoch as the default starting policy, while Figure 3(b) uses a policy which we call *reasonable* default policy. The only difference between the two default policies is that in the *reasonable* policy the agent will choose to open a door after synchronization if it believe that the tiger is behind the other door with probability greater than some threshold value(set at 5/6 for this experiment). The first thing to notice in Figures 3(a) and 3(b) is that the value of the policy is increased through the introduction of communication. This is because there is less mis-coordination arising from situations where agents open different doors or one opens the wrong door while the other agent performs *Listen*. We would expect that as K is reduced the value of the policy found should reduce too since the communication is more contrained with lower K. However, this is not always the case as evidenced in Figure 3(a), where the value of the joint policy found actually drops slightly when K is increased. This is because having a bigger constraint on communication (lower K) can sometimes cause the search to terminate at a different higher equilibrium.

Through the first two experiments we can conclude that introducing communication often results in an improvement in value as well as savings in space and time. In addition by imposing constraints on how often communication must be performed, we can obtain further improvements in terms of space and time although this might be at the expense of expected value.

In our third experiment (see Figure 4), we varied the cost of synchronization (X-axis) and determined the optimal value of K (Y-axis). We defined the optimal value of K as the value of K such that increasing K further does not yield any increase in value. As seen in the figure, as synchronization becomes more and more expensive the optimal K value increases.

7. Conclusion

Distributed POMDPs provide a rich framework to model uncertainties and utilities in complex multiagent domains, leading to significant recent research in applying these models in multiagent systems. The prohibitive computational complexity in finding optimal policies has led to a two pronged approach in the field: (i) restrict the POMDP model and find a truly optimal policy for this restricted model; (ii) maintain the rich model, but focus on local rather than global optimality. While significant research has focused on the first approach, including detailed investigation of communication[12, 4], we pursue the second approach. The key contribution of this paper is to investigate the impact of communication within the completely general distributed POMDP framework. We show how communicative acts can be explicitly introduced in or-



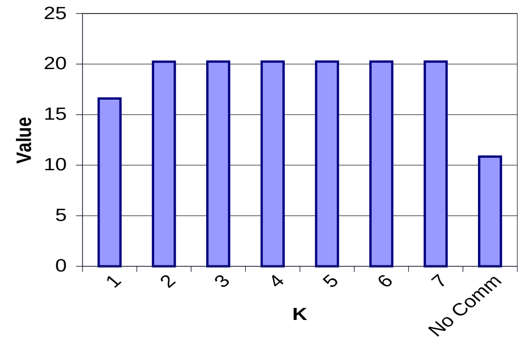


Figure 3. Value of joint optimal policy starting for T=7 with, a) (top) all listen default policy , b) (bottom)reasonable default policy

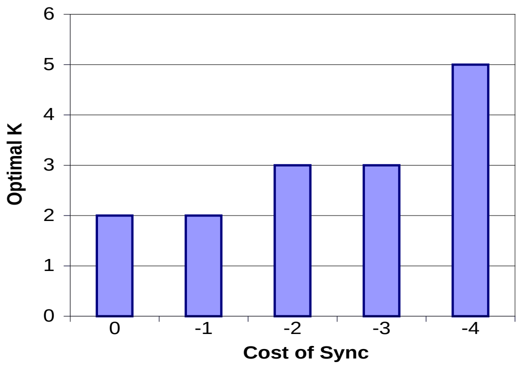


Figure 4. Optimal K for different costs of Sync ($T = 7$)

der to find locally optimal joint policies that allow agents to coordinate better through synchronization achieved via communication. Furthermore, the introduction of communication allows us to develop a novel *compact* policy representation that results in savings of both space and time which are analyzed theoretically and verified empirically. Finally, through the imposition of constraints on communication such as requiring communication at least once every K steps, even greater space and time savings can be obtained.

Acknowledgments

This research was supported by NSF grant #0208580.

References

[1] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Transition-Independent Decentralized Markov Decision Processes. In *AAMAS*, 2003.

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

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

[4] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *AAMAS*, 2003.

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

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

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

[8] R. Nair, M. Tambe, and S. Marsella. Team formation for reformation in multiagent domains like robocuprescue. In *RoboCup Symposium*, 2002.

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

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

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

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