Introducing Communication in Dis-POMDPs with Locality of Interaction

Makoto Tasaki [a,*], Yuichi Yabu [a], Yuki Iwanari [a], Makoto Yokoo [a], Janusz Marecki [b],
Pradeep Varakantham [c] and Milind Tambe [d]

[a] *Graduate School of ISEE, Kyushu University, Fukuoka, 819-0395 Japan, E-mail: tasaki@agent.,*
yabu@agent.is., iwanari@agent.is.,yokoo@is.kyusyu-u.ac.jp
[b] *IBM T.J. Watson Research Center, 1101 Kirchawan Road, Route 134, Yorktown Heights, NY 10598 E-mail:*
marecki@us.ibm.com
[c] *Carnegie Mellon University, Pittsburgh, PA 15213, E-mail: pradeepv@cs.cmu.edu*
[d] *University of Southern California, Los Angeles, CA 90089, E-mail: tambe@usc.edu*

Abstract. The Networked Distributed POMDPs (ND-POMDPs) can model multiagent systems in uncertain domains and has begun to scale-up the number of agents. However, prior work in ND-POMDPs has failed to address communication. Without communication, the size of a local policy at each agent within the ND-POMDPs grows exponentially in the time horizon. To overcome this problem, we extend existing algorithms so that agents periodically communicate their observation and action histories with each other. After communication, agents can start from new synchronized belief state. Thus, we can avoid the exponential growth in the size of local policies at agents. Furthermore, we introduce an idea that is similar to the Point-based Value Iteration algorithm to approximate the value function with a fixed number of representative points. Our experimental results show that we can obtain much longer policies than existing algorithms as long as the interval between communications is small.

Keywords: Multi-agent system, Distributed POMDPs, Communication

1. Introduction

Distributed Partially Observable Markov Decision Problems (Dis-POMDPs) are emerging as a popular approach for modeling sequential decision making in teams operating under uncertainty [1,10,5]. The uncertainty is due to the nondeterminism in the outcomes of actions and the limited observability of world states. Unfortunately, as shown by Bernstein *et al.* [1], the problem of finding an optimal joint policy for a distributed POMDP is NEXP-Complete if no assumptions are made about the domain conditions.

To address this significant computational complexity, Networked Distributed POMDPs (ND-POMDPs) [6], a model motivated by domains such as distributed sensor nets, distributed UAV teams, and distributed satel-lites, was introduced. These domains are character-ized by teams of agents coordinating with strong lo-cality in their interactions. For example, within a large distributed sensor net, only a small subset of sensor agents must coordinate to track targets. By exploiting the locality, LID-JESP [6] (locally optimal) and SPI-DER [11] (globally optimal), which are leading al-gorithms in this area, can scale-up in the number of agents. However, these approaches cannot handle run-time communication among agents. A consequence of this shortcoming is the exponential growth in the size of local policies. Recently, Mareki et.al. [4] devel-oped the FANS algorithm that utilizes a finite state ma-chine to represent each local policy. By using a finite state machine, the size of a local policy becomes fixed. However, run-time communication is not considered in [4].

---

*Corresponding author. E-mail: tasaki@agent.is.kyushu-u.ac.jp

To overcome this problem, we provide extensions to these algorithms called LID-JESP-Comm and SPIDER-Comm by introducing the run-time communication scheme presented in [5]. More specifically, agents periodically exchange observation and action histories with each other. Compared to other approaches such as [2,8,9], the advantage of using this scheme is that it allows the agents to build a new joint policy from a new synchronized belief state, i.e., instead of having one huge policy tree, an agent has multiple smaller policy trees.

Though this approach reduces the size of policies, it creates an exponential number of synchronized belief states after communication. To overcome this problem, we introduce an idea that resembles the Point-based Value Iteration (PBVI) algorithm [7] for single agent POMDPs. Instead of computing policies for all the synchronized belief states, we compute policies (and corresponding value vectors) only for a set of of representative belief points. Thus, we approximate the value function over the entire belief set by these value vectors, i.e., for any given belief point, we use the policy corresponding to the value vector that yields the highest value.

We develop two new algorithms based on this idea, i.e., LID-JESP-Comm and SPIDER-Comm (extensions to LID-JESP and SPIDER respectively). Since communication introduces inter-dependencies among agent policies, these algorithms lose some of the merits of the original algorithms. In LID-JESP-Comm, to update the policy of an agent, we need to consider the policies of all the other agents. SPIDER-Comm cannot provide global optimality, because it requires the enumeration of all joint policies. Despite these disadvantages, our experimental results show that these algorithms can obtain much longer policies than existing algorithms within a reasonable amount of time.

2. Model: Networked Distributed POMDP

We follow the networked distributed POMDP (ND-POMDP) model [6] as a concrete description of a Dis-POMDP. It is defined for a group of n agents as tuple $\langle S, A, P, \Omega, O, R, b \rangle$, where $S = \times_{1 \leq i \leq n} S_i \times S_u$ is the set of world states. S_i refers to the set of local states of agent i and S_u is the set of unaffectable states. An unaffectable state represents the part of world states that cannot be affected by agent actions. $A = \times_{1 \leq i \leq n} A_i$ is the set of joint actions, where A_i is the set of actions for agent i.

ND-POMDP assumes transition independence, i.e., the transition function is defined as $P(s, a, s') = P_u(s_u, s'_u) \cdot \prod_{1 \leq i \leq n} P_i(s_i, s_u, a_i, s'_i)$, where $a = \langle a_1, \ldots, a_n \rangle$ is the joint action performed in state $s = \langle s_1, \ldots, s_n, s_u \rangle$ and $s' = \langle s'_1, \ldots, s'_n, s'_u \rangle$ is the resulting state. $\Omega = \times_{1 \leq i \leq n} \Omega_i$ is the set of joint observations where Ω_i is the set of observations for agent i. Observational independence is assumed in ND-POMDPs i.e., the joint observation function is defined as $O(s', a, \omega) = \prod_{1 \leq i \leq n} O_i(s'_i, s'_u, a_i, \omega_i)$. where s' is the world state that results from the agents performing a in the previous state, and ω is the observation received in state s'. Reward function R is defined as $R(s, a) = \sum_l R_l(s_{l1}, \ldots, s_{lr}, s_u, \langle a_{l1}, \ldots, a_{lr} \rangle)$, where each l could refer to any subgroup of agents and $r = |l|$. Based on the reward function, an interaction hypergraph is constructed. Hyper-link l exists between a subset of agents for all R_l that comprise R. The interaction hypergraph is defined as $G = (Ag, E)$, where agents Ag are the vertices and $E = \{l | l \subseteq Ag \wedge R_l$ is a component of $R\}$ are the edges. The distribution over the initial state b is defined as $b(s) = b_u(s_u) \cdot \prod_{1 \leq i \leq n} b_i(s_i)$, where b_u and b_i refer to distribution over the initial unaffectable and agent i's belief states, respectively. Each agent i chooses its actions based on its local policy π_i that maps its observation history to an action. The goal in ND-POMDP is to compute joint policy $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ that maximizes the team's expected reward over finite horizon T starting from belief state b.

3. Domain: Distributed Sensor Network

Distributed sensor networks are a large, important class of domains that motivate our work. This paper focuses on a set of target tracking problems that arise in certain types of sensor networks [6]. Figure 1 shows a specific problem instance within this type that consists of three sensors. Here, each sensor node can scan in one of four directions: North, South, East or West (see Figure 1). To track a target and obtain associated reward, two sensors with overlapping scanning areas must be coordinated by simultaneously scanning the same area. In Figure 1, to track a target in Loc 1, sensor 1 needs to scan 'East' and sensor 2 needs to scan 'West' simultaneously. We assume there exist two independent targets, whose movements are uncertain and unaffected by the sensor agents. Based on the area it is scanning, each sensor receives observations that can have false positives and false negatives. Sen-
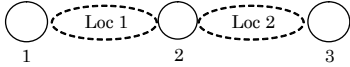
Fig. 1. A 3-chain sensor configuration

sors' observations and transitions are independent of each other's actions. Each agent incurs a scanning cost whether the target is present or not, but no cost if it is turned off. There is a high reward for successfully tracking a target.

These targets can be either enemies or friends of agents, i.e., a sensor network is trying to detect the incursion of enemy vehicles, or these agents want to localize friendly robots in an environment. Having an accurate probabilistic model of targets would be easier in a friendly environment. Also, co-learning of sensor agents and robot agents in a friendly environment would be an interesting research topic.

4. Existing Algorithms

4.1. LID-JESP

The locally optimal policy generation algorithm called LID-JESP (Locally interacting distributed joint search for policies) is based on DBA [12,3] and JESP [5]. In LID-JESP, each agent starts from its initial, randomly selected policy. By fixing the local policies of its neighbors, each agent tries to find the way to improve its local policy. Then, one agent who can maximally improve the expected reward within the neighbors can change its local policy. Agents repeat this process until no agent can find a better policy. This process of trying to improve the local neighborhood utility is done in a distributed manner similar to DBA.

More specifically, each agent i starts with a random policy and exchanges its policies with its neighbors. It then computes its local neighborhood utility with respect to its current policy and its neighbors' policies. The local neighborhood utility of agent i is defined as the expected reward for executing joint policy π accruing due to the hyper-links that contain agent i. Agent i then tries to improve upon its current policy by computing the local neighborhood utility of agent i's best response to its neighbors' policies. Agent i then computes the gain that it can make to its local neighborhood utility, and exchanges its gain with its neighbors. If i's gain is greater than any of its neighbors' gain, i changes its policy and sends its new policy to all its neighbors. This process of trying to improve the local

neighborhood utility is continued until the joint policies reach an equilibrium.

4.2. SPIDER

In this subsection, we briefly describe SPIDER [11]. In essence, SPIDER performs a depth-first branch & bound search procedure using an admissible heuristic function, while exploiting the locality of agent interactions. By utilizing an admissible heuristic function, SPIDER can obtain the upper bounds on the expected values of policies and prune a search space that cannot be an optimal solution. Also, agents are organized into a Depth First Search (DFS) tree (i.e., pseudo tree) that allow links between ancestors and children. In a DFS tree, agents in different branches can be considered independent, given the policies of ancestors are fixed.

In Figure 2, we show a snapshot of search trees in the SPIDER algorithm. The middle agent in Figure 1 is the root of the tree. Each agent is assigned a policy with $T = 2$. Each rounded-edge rectangle (search tree node) indicates a partial/complete joint policy, a rectangle (internal to a rounded-edge rectangle) indicates an agent and the ovals (internal to a rectangle) show its policy.

A heuristic or actual expected value for a joint policy is indicated in the top right corner of the rounded rectangle. If the number is underlined, the actual expected value of the joint policy is provided. SPIDER begins with no policy assigned to any of the agents (shown in level 1 of the search tree). Level 2 of the search tree indicates that the joint policies are sorted based on upper bounds computed for the root agent's policies. Level 3 shows one SPIDER search node with a complete joint policy (a policy assigned to each agent). The expected value for this joint policy is used to prune the nodes in level 2 (those with upper bounds < 234). When creating policies for each non-leaf agent i, SPIDER potentially performs two steps:

STEP 1 Obtaining upper bounds and sorting In this step, agent i computes the upper bounds on the expected values of the joint policies corresponding to each of its policies and the fixed ancestor policies. The upper bounds (heuristic value) is calculated assuming that agents that do not have their local policies assigned yet will search for their optimal policies assuming full observability, i.e., using MDP policy search All the policies of agent i are then sorted based on these upper bounds in descending order.

STEP 2 Exploring and pruning Exploring implies computing the best response joint policy that corresponds to the fixed ancestor policies of agent i. This is performed by iterating through all policies of agent i and summing two quantities for each policy: (i) the best response for all of i's children; (ii) the expected value obtained by i for fixed policies of ancestors. Pruning refers to avoiding the exploration of all policies at agent i using the current best expected value as *threshold*. A policy need not be explored if its upper bound is less than the *threshold*. For example, if the best response policies from the leaf agents yield an actual expected value of 240, a policy with upperbound 232 is pruned (see Figure 2).
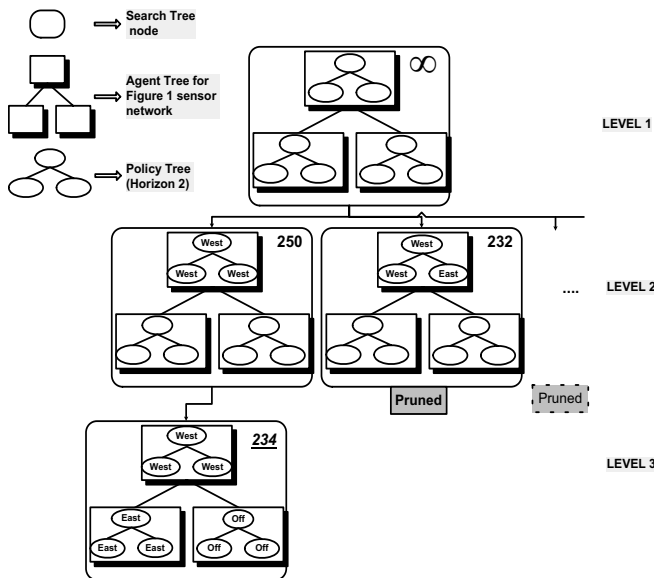

Fig. 2. Execution of SPIDER, an example

5. Communication in ND-POMDP

The basic idea of introducing the run-time communication scheme to ND-POMDPs is as follows. Agents periodically communicates their observation/action histories with each other. As a result, agents reach a new synchronized belief state and start a new policy. This corresponds to having multiple smaller policy trees rather than having a single huge policy tree. However, the number of new synchronized belief states grows exponentially. To overcome this problem, we

use a fixed number of representative points to approximate the values of new synchronized belief states.

More specifically, we introduce the run-time communication scheme presented in [5] to ND-POMDPs as follows.

- In the initial state, agents have a synchronized belief state. Each agent has a local plan for subsequent k steps[1].
- Each agent executes its local plan for k steps. Then, agents go through the *communication phase*.
- During the *communication phase*, agents communicate their observation/action histories with each other. By exchanging the observation and action histories with each other, they have common knowledge on the observation/action histories of all agents. Thus, they can update their beliefs and reach a new synchronized belief state.
- Each agent chooses a part of its policy that corresponds to that belief point

Thus, we use multiple small policy trees with a constant depth k instead of one huge policy tree whose size is exponential to the length of the time horizon.

However, the number of joint (small) policies grows exponentially to the length of the time horizon. To overcome this problem, we introduce an idea that resembles the Point-based Value Iteration (PBVI) algorithm [7] for single agent POMDPs. More specifically, we use a fixed number of *representative belief points* and compute the k-step optimal joint policy for each representative belief point. By using a fixed number of representative belief points, the obtained policy can be suboptimal. However, as shown in [7], we can bound the the difference between the obtained approximated policy and the optimal policy in a single agent POMDPs[2].

Let us assume we fix one particular k-step joint policy π. The expected reward of π starting from one particular belief state b is represented as a weighted linear combination of the expected reward for each state (Figure 3). More specifically, assume that possible states are $\{s_1, s_2, \ldots, s_N\}$ and a belief state

---

[1]For simplicity, we assume one communication phase occurs exactly once after k non-communication steps. Extending the algorithms to the cases where one communication phase occurs *at least* once within k steps is rather straightforward.

[2]On the other hand, our proposed methods, i.e., LID-JESP-Comm and SPIDER-Comm, utilize other approximation methods. Therefore, we cannot bound the difference between the obtained approximated joint policy and the optimal joint policy.
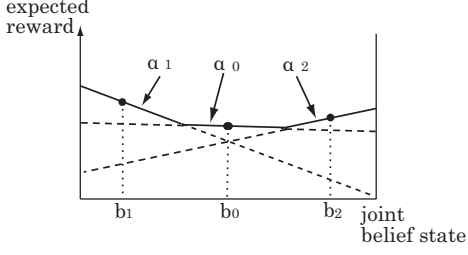
Fig. 3. Value function and α vectors

Fig. 4. Policy obtained by LID-JESP-Comm or SPIDER-Comm

$b = \langle b(s_1), b(s_2), \ldots, b(s_N) \rangle$. The expected reward for joint policy π starting from b, denoted as $V^\pi(b)$, can be represented as:

$$b(s_1) * V^\pi(\langle 1, 0, \ldots \rangle) + b(s_2) * V^\pi(\langle 0, 1, \ldots \rangle)$$

$$+ \ldots + b(s_N) * V^\pi(\langle 0, \ldots, 1 \rangle)$$

Here, we call the vector $\langle V^\pi(\langle 1, 0, \ldots \rangle), V^\pi(\langle 0, 1, \ldots \rangle) \ldots, V^\pi(\langle 0, \ldots, 1 \rangle) \rangle$ as α vector. The expected reward starting from belief state b is obtained by calculating the inner product of the belief state and the α vector. Since the optimal reward of the entire belief space is obtained by taking the maximal value for all possible joint polices, it is clear that the optimal reward satisfies piece-wise linear, convex (PWLC) property.

We approximate this optimal reward for the entire belief space (value function) using these α vectors of representative belief points (Figure 3).

5.1. ND-POMDP-Comm Algorithm (the mechanism)

Next, we describe the details of algorithm in ND-POMDP with communication. We employ the following notation to denote the policies and the expected values:

$\pi^* \Rightarrow$ optimal joint policy of all agents.
$\pi^{i,*} \Rightarrow$ joint policy computed before searching for the policy of agent i.
$\pi^{j+} \Rightarrow$ joint policy of agents searched for after j.
$\pi_i \Rightarrow$ local policy of agent i.
$v[\vec{\alpha}, b] \Rightarrow$ the expected value for $\vec{\alpha}$ given belief state b.
$\hat{v}[\pi^{i,*} || \pi_i] \Rightarrow$ upper bound on the expected value given $\pi^{i,*}$ and π_i.
$B \Rightarrow$ the set of representative points.

We need to find a joint policy for each representative point after each communication phase. If there are $|B|$ representative points and c communication phases, we need to find $c|B|$ joint policies for belief points after communication and one joint policy for the initial belief state.
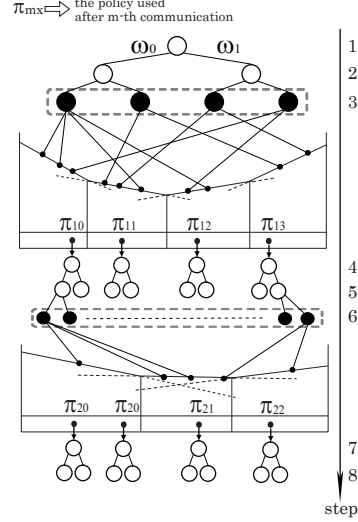
Figure 4 shows the local policy given $k = 2$. First, our algorithm computes the joint policy for each of the representative points after the last communication phase, i.e., the joint policy for time steps 7-8 (Figure 4). This results in three policies: π_{20}, π_{21}, and π_{22}. Our algorithm computes the α-vectors for these joint policies.

Next, it computes a joint policy for time steps 4-6. A rectangle (represented by dashed lines) indicates the communication phase and lines from filled circles indicate the transitions to synchronized belief states after communication. The policies generated are π_{10}, π_{11}, π_{12}, and π_{13}. The algorithm computes the α-vectors for these joint policies. Finally, it determines the joint policy for the initial belief state.

Algorithm 1 provides the pseudo code for ND-POMDP with communication. This algorithm outputs a joint policy π^*. $CommPhase$ represents the number of communication phases. In line 2, a set of representative belief points is generated using the method described in the next subsection. Then, a joint policy is calculated for each representative belief point $b \in B$, and the obtained joint policy is stored in $\pi^*[b, CommPhase]$ (lines 5-7). In each action phase, FINDPOLICY function finds a joint policy and its α-vector, based on the extension of LID-JESP-Comm or SPIDER-Comm described in Sections 5.3 and 5.4.

5.2. Belief Point Selection

The way to choose representative belief points can affect the solution quality. We consider the following

Algorithm 1 ND-POMDP-Comm($k, CommPhase$)

1: initialize $\vec{\alpha}^*, \pi^* \leftarrow null$
2: $B \leftarrow$ BeliefExpansion(b_{init})
3: **while** $CommPhase \geq 0$ **do**
4:   **for all** $b \in B$ **do**
5:     $\langle \pi^*[b, CommPhase], \vec{\alpha} \rangle \leftarrow$
        FINDPOLICY($b, root, null, -\infty, k, \vec{\alpha}^*$)
6:     $\vec{\alpha}^* \leftarrow \vec{\alpha}$
7:     $CommPhase = CommPhase - 1$
8: **return** π^*

---

two methods. We assume that initial belief state b_{init} is always included in representative belief points B.

Random Belief Selection (RA) In this method, we sample belief points from uniform distribution over the entire belief space.

Stochastic Simulation with Exploratory Action (SSEA) This method is based on the algorithm presented in [7]. We gradually expand B by adding new reachable belief points after k actions and communication. More specifically, we stochastically run k actions in the forward trajectory from the belief points already in B and obtain several candidates. From these candidates, we select belief points that improve the worst-case density, i.e., we choose the point farthest from any point already in B.

5.3. LID-JESP with Communication

LID-JESP with Communication (LID-JESP-Comm) performs the following procedure:

(i) For each representative point, we find the joint equilibrium policy (where each policy of an agent is the best response for other agents' policies) for k steps after the last communication using LID-JESP [6].

(ii) Then, for each representative point, we find the joint equilibrium policy for k steps after the second to the last communication. For the current k steps, we need only the policies of neighbors to evaluate the expected reward. On the other hand, to evaluate the expected reward after communication, we consider the policies of non-neighbors and obtain the probability distribution of the new synchronized belief states. For each new synchronized belief state, we use the best expected reward for the joint policies obtained in (i).

(iii) Then, we find the joint equilibrium policy for k steps after the third to the last communication, and so on.

5.4. SPIDER with Communication

Next, we describe the details of SPIDER with Communication (SPIDER-Comm). SPIDER can obtain global optimal joint policies by exploiting the locality of agent interaction. However, communication phase invalidates the locality in interaction that original SPIDER was relying on. In essence, agents on different hyperlinks are independent without communication, but they become interdependent after communication. More specifically, a new synchronized belief state (and the expected reward after communication) depends on all agents' policies. In SPIDER-Comm, we utilize a greedy method i.e., when finding a best response policy for agent i in the DFS tree, we don't enumerate the combinations of the joint policies of different subtrees, while we enumerate the combinations within a subtree. Thus, although the SPIDER-Comm cannot guarantee to find the global optimal joint policy, it can utilize the locality of interaction and obtain a reasonable policy within a reasonable amount of time.

Algorithm 2 provides a pseudo code for procedure **FINDPOLICY** for SPIDER-Comm, which finds a joint policy and its α-vector. First, we store all possible local policies in Π_i (line 2). If i is a leaf agent, the local policies of all agents in its subtree are already assigned. SPIDER-Comm obtains an exact value for the subtree (and ancestors) and new synchronized belief states after communication (assuming default policies are used by the agents whose policies are not assigned yet), and chooses the best one (lines 3-9). On the other hand, if i is not a leaf agent, SPIDER-Comm performs the following procedure: (a) sorts policies in descending order based on heuristic values (line 12), (b) recursively calls **FINDPOLICY** for the next agent and calculates the best response policies for each local policy of agent i as long as the heuristic evaluation of the policy is better than the solution found so far (line 17), (c) maintains the threshold, the best solution found so far (lines 18-21).

5.4.1. Heuristic Function

In SPIDER-Comm, we need to construct a heuristic function that estimates the expected reward for the current k steps and after communication.

In [11], the MDP heuristic function is introduced. More specifically, the subtree of agents is a Dis-POMDP in itself. Thus, we can construct a centralized MDP corresponding to the (subtree) Dis-POMDP and obtain the expected value of the optimal policy for this centralized MDP. The advantage of the MDP heuristic

Algorithm 2 FINDPOLICY($b, i, \pi^{i,*}, threshold, k, \vec{\alpha}^*$)

1:   $\vec{\hat{\alpha}} \leftarrow null, \hat{\pi}^* \leftarrow null$
2:   $\Pi_i \leftarrow$ GET-ALL-POLICIES(k, A_i, Ω_i)
3:   **if** IS-LEAF(i) **then**
4:     **for all** $\pi_i \in \Pi_i$ **do**
5:       $\vec{\alpha}_i \leftarrow$ GETVECTOR$(i, \pi_i, \pi^{i,*}, \vec{\alpha}^*)$
6:       **if** $v\left[\vec{\alpha}_i, b\right] > threshold$ **then**
7:         $\hat{\pi}^* \leftarrow \pi_i$
8:         $threshold \leftarrow v[\vec{\alpha}_i, b]$
9:         $\vec{\hat{\alpha}} \leftarrow \vec{\alpha}_i$
10: **else**
11:    $children \leftarrow$ CHILDREN(i)
12:    $\hat{\Pi}_i \leftarrow$ UPPER-BOUND-SORT$(b, i, \Pi_i, \pi^{i,*}, \vec{\alpha}^*)$
13:    **for all** $\pi_i \in \hat{\Pi}_i$ **do**
14:      **if** $\hat{v}[\pi^{i,*}||\pi_i] < threshold$ **then**
15:        Go to line 22
16:      **for all** $j \in children$ **do**
17:        $\langle \pi^{j+}, \vec{\alpha}_i \rangle \leftarrow$
           FINDPOLICY$(b, j, \pi^{i,*}||\pi_i, threshold, k, \vec{\alpha}^*)$
18:        **if** $v[\vec{\alpha}_i, b] > threshold$ **then**
19:          $\hat{\pi}^* \leftarrow \pi_i||\pi^{j+}$
20:          $threshold \leftarrow v[\vec{\alpha}_i, b]$
21:          $\vec{\hat{\alpha}} \leftarrow \vec{\alpha}_i$
22: **return** $\langle \hat{\pi}^*, \vec{\hat{\alpha}} \rangle$

is that it is admissible, i.e., it never under-estimates the optimal value. Thus, the SPIDER is guaranteed to find an optimal joint policy.

However, if we assume the subtree is solved by a centralized MDP (in which the current state is fully observable), we cannot estimate the new synchronized belief state after communication. Thus, we assign default policies to agents whose policies are not assigned yet and estimate the new synchronized belief state after communication assuming these agents use the default policies. We can use these default policies also for evaluating the expected reward for the current k steps. In this case, the heuristic function is no longer admissible, but it can prune more nodes and the run-time can be reduced. We will evaluate this trade-off in the next section.

6. Experimental Results

Our experiments were conducted on the example of the sensor network domain described in Section 2. We use three different topologies of sensors shown in Figure 5. Figure 5 (i) shows the example where there are three agents and two targets. Target 1 is either absent or in Loc1, and target 2 is either absent or in Loc2. Thus, there are 4 unaffectable states. Each agent can perform
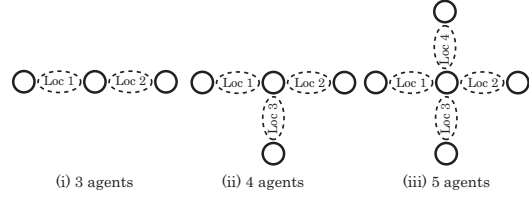

(i) 3 agents   (ii) 4 agents   (iii) 5 agents

Fig. 5. Sensor net configurations

Table 1

Run time (msec) /expected value for SPIDER and SPIDER-Comm ($T = 3$)

	SPIDER	SPIDER-Comm
runtime [sec]	20.80	0.39
value	141.90	87.05

turnOff, scanEast, or scanWest. Agents receive $+45$ as an immediate reward for finding target 1, $+35$ for finding target 2, and -5 for failing to find any target. Figure 5 (ii) shows the example where there are four agents and three targets, and (iii) shows the example where there are five agents and four targets.

We have compared two alternative methods for selecting representative points, i.e., RA or SSEA. We found that SSEA dominates RA, especially when the number of representative points is small. Thus, we use SSEA for selecting representative points in the following experiments.

Table 1 shows the runtime and the expected reward of the obtained joint policy of SPIDER and SPIDER-Comm for a sensor network with three sensors ($T = 3$ and $k = 1$). We cannot run SPIDER for larger T, since the size/number of local policies grow exponentially. By introducing communication, the runtime is drastically reduced. In fact, SPIDER takes 20.8 sec, while SPIDER-Comm takes only 0.39 sec. The expected rewards are 141.9 and 87.05, respectively. The expected reward of SPIDER-Comm is about 66% compared with SPIDER, since SPIDER spends all of three steps for executing actions, while SPIDER-Comm spends one step for communication.

Next, we evaluate the runtime and expected reward of SPIDER-Comm and LED-JESP-Comm. Figure 6 (a) provides runtime comparisons between SPIDER-Comm and LID-JESP-Comm that for $k = 2$ and $c = 1$ (c is the number of communications). In Figure 6, SPIDER-Comm (Default policy) indicates that SPIDER-Comm uses default policies both for the heuristic function for the current k steps and
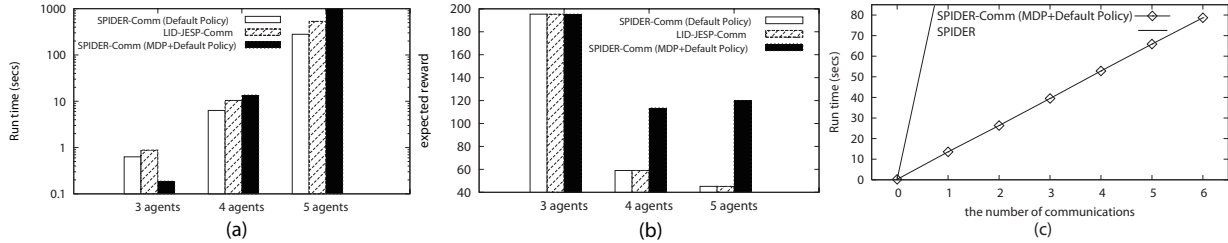
Fig. 6. (a)Runtime of SPIDER-Comm and LID-JESP-Comm, (b)expected reward of SPIDER-Comm and LID-JESP-Comm, and (c)runtime of SPIDER and SPIDER-Comm by increasing the number of communications

for estimating the belief states after communication. SPIDER-Comm (MDP+Default policy) indicates that SPIDER-Comm uses the MDP heuristic function for the current k steps and default policies for estimating the belief states after communication. The X-axis denotes the number of agents, while the Y-axis indicates the amount of time taken to compute the solution. SPIDER-Comm (MDP+Default policy) obtains runtime improvements over other methods in 3 agents configuration, while, in 4 and 5 agents configurations, SPIDER-Comm (Default policy) obtains runtime improvements over other methods. In Figure 6 (b), We evaluate the expected reward of SPIDER-Comm and LID-JESP-Comm in the same setting as Figure 6 (a). In 3 agents configuration, all methods obtain the same expected values. While, in 4 and 5 agents configurations, SPIDER-Comm (MDP+Default policy) obtains significantly better expected reward over other methods.

Finally, we evaluate the run-time of SPIDER and SPIDER-Comm (MDP+Default policy) by increasing the number of communications c for $k = 2$ in 4 agents configuration (Figure 6 (c)). When $c = 6$, the total time horizon is 20 (where the total time horizon T is equal to $c(k+1) + k$). When $c = 0$, agents find a joint policy for $T = 2$ without communication (note that the run-time for $c = 0$ is small but not zero).

We have obtained similar results for the run-time of other methods. We can see that our newly developed methods can obtain policies even if the length of the time horizon is large, as long as the interval between communications is small. For the original SPIDER, the maximal length of the time horizon is around 4, and for LID-JESP, the maximal length is around 6.

7. Conclusion

In this paper, we extended ND-POMDP so that agents can periodically communicate their observation

and action histories with each other, and developed two new algorithms: LID-JESP-Comm and SPIDER-Comm. To address the problem that the number of new synchronized belief states after communication will grow exponentially, we introduced an idea similar to the PBVI algorithm. Our experimental results show that these algorithms can obtain much longer policies than existing algorithms within a reasonable amount of time. Our future works include introducing a more flexible communication scheme, such as varying the interval between communications, introducing partial communications, etc.

References

[1] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov Decision Processes. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 32–37, 2000.

[2] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS-03)*, pages 137–144, 2003.

[3] K. Hirayama and M. Yokoo. Coordinated multi-agent local search. *Artificial Intelligence Journal*, 161(1-2):89–116, 2005.

[4] J. Marecki, T. Gupta, P. Varakantham, M. Tambe, and M. Yokoo. Not all agents are equal: scaling up distributed POMDPs for agent networks. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS-08)*, pages 485–492, 2008.

[5] R. Nair, M. Roth, M. Yokoo, and M. Tambe. Communication for improving policy computation in distributed POMDPs. In *Proceedings of the third International joint conference on Autonomous agents and Multi-agent Systems (AAMAS-04)*, pages 1096–1103, 2004.

[6] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed

constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 133–139, 2005.

[7] J. Pineau, G. Gordon, and S. Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 227:335–380, 2006.

[8] M. Roth, R. Simmons, and M. Veloso. Exploiting factored representations for decentralized execution in multiagent teams. In *Proceedings of the sixth International joint conference on Autonomous agents and Multi-agent Systems (AAMAS-07)*, pages 457–463, 2007.

[9] J. Shen, R. Becker, and V. Lesser. Agent interaction in distributed POMDPs and its implications on complexity. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS-06)*, pages 529–536, 2006.

[10] D. Szer and S. Z. Francois Charpillet. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 576–590, 2005.

[11] P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo. Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In *Proceedings of the sixth International joint conference on Autonomous agents and Multi-agent Systems (AAMAS-07)*, pages 822–829, May 2007.

[12] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 401–408, 1996.