Exploiting Locality of Interaction in Networked Distributed POMDPs

Yoonheui Kim[⋆], Ranjit Nair[†], Pradeep Varakantham[⋆], Milind Tambe[⋆], Makoto Yokoo[§]
[⋆]Computer Science Department,University of Southern California, CA 90089, USA,{yoonheuk,varakant,tambe}@usc.edu
[†]Automation and Control Solutions, Honeywell Laboratories, MN 55418, USA, ranjit.nair@honeywell.com
[§]Department of Intelligent Systems, Kyushu University, Fukuoka 812-8581, Japan,yokoo@is.kyushu-u.ac.jp

Abstract

In many real-world multiagent applications such as distributed sensor nets, a network of agents is formed based on each agent's limited interactions with a small number of neighbors. While distributed POMDPs capture the real-world uncertainty in multiagent domains, they fail to exploit such locality of interaction. Distributed constraint optimization (DCOP) captures the locality of interaction but fails to capture planning under uncertainty. In previous work, we presented a model synthesized from distributed POMDPs and DCOPs, called Networked Distributed POMDPs (ND-POMDPs). Also, we presented LID-JESP (locally interacting distributed joint equilibrium-based search for policies: a distributed policy generation algorithm based on DBA (distributed breakout algorithm). In this paper, we present a stochastic variation of the LID-JESP that is based on DSA (distributed stochastic algorithm) that allows neighboring agents to change their policies in the same cycle. Through detailed experiments, we show how this can result in speedups without a large difference in solution quality. We also introduce a technique called *hyper-link-based decomposition* that allows us to exploit locality of interaction further, resulting in faster run times for both LID-JESP and its stochastic variant without any loss in solution quality.

Introduction

Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are emerging as an important approach for multiagent teamwork. These models enable modeling more realistically the problems of a team's coordinated action under uncertainty. Unfortunately, as shown by Bernstein *et al.* (2000), the problem of finding the optimal joint policy for a general distributed POMDP is NEXP-Complete. Researchers have attempted two different approaches to address this complexity. First, they have focused on algorithms that sacrifice global optimality and instead focus on local optimality (Nair *et al.* 2003; Peshkin *et al.* 2000). Second, they have focused on restricted types of domains, e.g. with transition independence or collective observability (Becker *et al.* 2004). While these approaches have led to useful advances, the complexity of the distributed POMDP problem has limited most experiments to a central policy generator planning for just two agents. Further, these previous approaches have relied on a centralized planner that computes the policies for all the agents in an off-line manner.

Nair *et al.* (2005) presented third complementary approach called Networked Distributed POMDPs (ND-POMDPs), that is motivated by domains such as distributed sensor nets (Lesser, Ortiz, & Tambe 2003), distributed UAV teams and distributed satellites, where an agent team must coordinate under uncertainty, but agents have strong locality in their interactions. For example, within a large distributed sensor net, small subsets of sensor agents must coordinate to track targets. To exploit such local interactions, ND-POMDPs combine the planning under uncertainty of POMDPs with the local agent interactions of distributed constraint optimization (DCOP) (Modi *et al.* 2003; Yokoo & Hirayama 1996). DCOPs have successfully exploited limited agent interactions in multiagent systems, with over a decade of algorithm development. Distributed POMDPs benefit by building upon such algorithms that enable distributed planning, and provide algorithmic guarantees. DCOPs benefit by enabling (distributed) planning under uncertainty — a key DCOP deficiency in practical applications such as sensor nets (Lesser, Ortiz, & Tambe 2003). In that work, we introduced the LID-JESP algorithm that combines the JESP algorithm of Nair *et al.* (2003) and the *DBA* (Yokoo & Hirayama 1996) DCOP algorithm. LID-JESP thus combines the dynamic programming of JESP with the innovation that it uses off-line distributed policy generation instead of JESP's centralized policy generation.

This paper makes two novel contributions to the previous work on Networked POMDPs. First, we present a stochastic variation of the LID-JESP that is based on DSA (distributed stochastic algorithm) (Zhang *et al.* 2003) that allows neighboring agents to change their policies in the same cycle. Through detailed experiments, we show how this can result in speedups without a large difference in solution quality.

Second, we introduce a technique called *hyper-link-based decomposition* (HLD) that decomposes each agent's local optimization problem into loosely-coupled optimization problems for each hyper-link. This allows us to exploit locality of interaction further resulting in faster run times for both LID-JESP and its stochastic variant without any loss in solution quality.

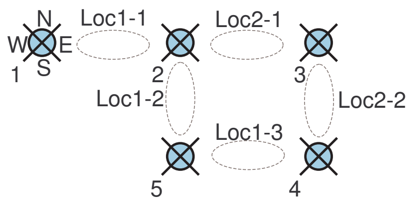


Figure 1: Sensor net scenario: If present, target1 is in Loc1-1, Loc1-2 or Loc1-3, and target2 is in Loc2-1 or Loc2-2.

Illustrative Domain

We describe an illustrative problem within the distributed sensor net domain, motivated by the real-world challenge in (Lesser, Ortiz, & Tambe 2003)[1]. 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 coordinate by scanning the same area simultaneously. We assume that there are two independent targets and that each target's movement is 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. Each agent incurs a cost for scanning whether the target is present or not, but no cost if it turns off.

As seen in this domain, each sensor interacts with only a limited number of neighboring sensors. For instance, sensors 1 and 3's scanning areas do not overlap, and cannot affect each other except indirectly via sensor 2. The sensors' observations and transitions are independent of each other's actions. Existing distributed POMDP algorithms are unlikely to work well for such a domain because they are not geared to exploit locality of interaction. Thus, they will have to consider all possible action choices of even non-interacting agents in trying to solve the distributed POMDP. Distributed constraint satisfaction and distributed constraint optimization (DCOP) have been applied to sensor nets but they cannot capture the uncertainty in the domain.

ND-POMDPs

We define an ND-POMDP for a group Ag of n agents as a tuple $\langle S, A, P, \Omega, O, R, b\rangle$, where $S = \times_{1\le i\le n} S_i \times S_u$ 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. Unaffectable state refers to that part of the world state that cannot be affected by the agents' actions, e.g. environmental factors like target locations that no agent can control. $A = \times_{1\le i\le n} A_i$ is the set of joint actions, where A_i is the set of action for agent i.

We assume a *transition independent* distributed POMDP model, where the transition function is defined as $P(s,a,s') = P_u(s_u,s'_u)\cdot\prod_{1\le i\le n} P_i(s_i,s_u,a_i,s'_i)$, 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. Agent i's transition function is defined as $P_i(s_i,s_u,a_i,s'_i) = \Pr(s'_i|s_i,s_u,a_i)$ and the unaffectable transition function is defined as $P_u(s_u,s'_u) = \Pr(s'_u|s_u)$. Becker *et al.* (2004) also relied on transition independence, and Goldman and Zilberstein (2004) introduced the possibility of uncontrollable state features. In both works, the authors assumed that the state is *collectively observable*, an assumption that does not hold for our domains of interest.

$\Omega = \times_{1\le i\le n}\Omega_i$ is the set of joint observations where Ω_i is the set of observations for agents i. We make an assumption of *observational independence*, i.e., we define the joint observation function as $O(s,a,\omega) = \prod_{1\le i\le n} O_i(s_i,s_u,a_i,\omega_i)$, where $s = \langle s_1,\ldots,s_n,s_u\rangle$, $a = \langle a_1,\ldots,a_n\rangle$, $\omega = \langle\omega_1,\ldots,\omega_n\rangle$, and $O_i(s_i,s_u,a_i,\omega_i) = \Pr(\omega_i|s_i,s_u,a_i)$.

The reward function, R, is defined as $R(s,a) = \sum_l R_l(s_{l1},\ldots,s_{lk},s_u,\langle a_{l1},\ldots,a_{lk}\rangle)$, where each l could refer to any sub-group of agents and $k = |l|$. In the sensor grid example, the reward function is expressed as the sum of rewards between sensor agents that have overlapping areas ($k = 2$) and the reward functions for an individual agent's cost for sensing ($k = 1$). Based on the reward function, we construct an *interaction hypergraph* where a hyper-link, l, exists between a subset of agents for all R_l that comprise R. *Interaction hypergraph* is defined as $G = (Ag, E)$, where the agents, Ag, are the vertices and $E = \{l | l \subseteq Ag \wedge R_l$ is a component of $R\}$ are the edges. *Neighborhood* of i is defined as $N_i = \{j \in Ag | j \neq i \wedge (\exists l \in E,\ i \in l \wedge j \in l)\}$. $S_{N_i} = \times_{j\in N_i} S_j$ refers to the states of i's neighborhood. Similarly we define $A_{N_i} = \times_{j\in N_i} A_j$, $\Omega_{N_i} = \times_{j\in N_i}\Omega_j$, $P_{N_i}(s_{N_i},a_{N_i},s'_{N_i}) = \prod_{j\in N_i} P_j(s_j,a_j,s'_j)$, and $O_{N_i}(s_{N_i},a_{N_i},\omega_{N_i}) = \prod_{j\in N_i} O_j(s_j,a_j,\omega_j)$.

b, the distribution over the initial state, is defined as $b(s) = b_u(s_u)\cdot\prod_{1\le i\le n} b_i(s_i)$ where b_u and b_i refer to the distributions over initial unaffectable state and i's initial state, respectively. We define $b_{N_i} = \prod_{j\in N_i} b_j(s_j)$. We assume that b is available to all agents (although it is possible to refine our model to make available to agent i only b_u, b_i and b_{N_i}). 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 a finite horizon T starting from b. π_i refers to the individual policy of agent i and is a mapping from the set of observation histories of i to A_i. π_{N_i} and π_l refer to the joint policies of the agents in N_i and hyper-link l respectively.

ND-POMDP can be thought of as an *n-ary* DCOP where the variable at each node is an individual agent's policy. The reward component R_l where $|l| = 1$ can be thought of as a local constraint while the reward component R_l where $l > 1$ corresponds to a non-local constraint in the constraint graph. In the next section, we push this analogy further by taking inspiration from the DBA algorithm (Yokoo & Hirayama 1996), an algorithm for distributed constraint satisfaction, to develop an algorithm for solving ND-POMDPs.

The following proposition (proved in (Nair *et al.* 2005)) shows that given a factored reward function and the assumptions of transitional and observational independence, the resulting value function can be factored as well into value functions for each of the edges in the interaction hypergraph.

Proposition 1 *Given transitional and observational independence and* $R(s,a) =$

[1]For simplicity, this scenario focuses on binary interactions. However, ND-POMDP and LID-JESP allow n-ary interactions.

$\sum_{l\in E} R_l(s_{l1},\ldots,s_{lk},s_u,\langle a_{l1},\ldots,a_{lk}\rangle)$,

$$V^t_\pi(s^t,\vec{\omega}^t) = \sum_{l\in E} V^t_{\pi_l}(s^t_{l1},\ldots,s^t_{lk},s^t_u,\vec{\omega}^t_{l1},\ldots\vec{\omega}^t_{lk}) \qquad (1)$$

where $V^t_\pi(s^t,\vec{\omega})$ is the expected reward from the state s^t and joint observation history $\vec{\omega}^t$ for executing policy π, and $V^t_{\pi_l}(s^t_{l1},\ldots,s^t_{lk},s^t_u,\vec{\omega}^t_{l1},\ldots\vec{\omega}^t_{lk})$ is the expected reward for executing π_l accruing from the component R_l.

We define *local neighborhood utility* of agent *i* as the expected reward for executing joint policy π accruing due to the hyper-links that contain agent *i*:

$$V_\pi[N_i] = \sum_{s_i,s_{N_i},s_u} b_u(s_u)\cdot b_{N_i}(s_{N_i})\cdot b_i(s_i)\cdot \sum_{l\in E\ s.t.\ i\in l} V^0_{\pi_l}(s_{l1},\ldots,s_{lk},s_u,\langle\rangle,\ldots,\langle\rangle) \qquad (2)$$

Proposition 2 Locality of interaction: *The local neighborhood utilities of agent i for joint policies π and π' are equal ($V_\pi[N_i] = V_{\pi'}[N_i]$) if $\pi_i = \pi'_i$ and $\pi_{N_i} = \pi'_{N_i}$.*

From the above Proposition (proved in (Nair *et al.* 2005)), we conclude that increasing the local neighborhood utility of agent *i* cannot reduce the local neighborhood utility of agent *j* if $j \notin N_i$. Hence, while trying to find best policy for agent *i* given its neighbors' policies, we do not need to consider non-neighbors' policies. This is the property of *locality of interaction* that is used in later sections.

Previous Work

LID-JESP

The locally optimal policy generation algorithm called LID-JESP (Locally interacting distributed joint equilibrium search for policies) is based on DBA (Yokoo & Hirayama 1996) and JESP (Nair *et al.* 2003). In this algorithm (see Algorithm 1), each agent tries to improve its policy with respect to its neighbors' policies in a distributed manner similar to DBA. Initially each agent *i* starts with a random policy and exchanges its policies with its neighbors (lines 3-4). It then computes its local neighborhood utility (see Equation 2) with respect to its current policy and its neighbors' policies. Agent *i* then tries to improve upon its current policy by calling function GETVALUE (see Algorithm 3), which returns the local neighborhood utility of agent *i*'s best response to its neighbors' policies. This algorithm is described in detail below. Agent *i* then computes the gain (always ≥ 0 because at worst GETVALUE will return the same value as *prevVal*) that it can make to its local neighborhood utility, and exchanges its gain with its neighbors (lines 8-11). If *i*'s gain is greater than any of its neighbors' gain[2], *i* changes its policy (FINDPOLICY) and sends its new policy to all its neighbors. This process of trying to improve the local neighborhood utility is continued until termination. Termination detection is based on using a termination counter to count the number of cycles where $gain_i$ remains $= 0$. If its gain is greater than zero the termination counter is reset. Agent *i* then exchanges its termination counter with its neighbors and set its counter to the minimum of its counter and its neighbors' counters. Agent *i* will terminate if its termination counter equals the diameter of the interaction hypergraph.

[2]The function $\mathbf{argmax}_j$ disambiguates between multiple j corresponding to the same max value by returning the lowest j.

Algorithm 1 LID-JESP(i, ND-POMDP)

```
 1: Compute interaction hypergraph and $N_i$
 2: $d \leftarrow$ diameter of hypergraph, $terminationCtr_i \leftarrow 0$
 3: $\pi_i \leftarrow$ randomly selected policy, $prevVal \leftarrow 0$
 4: Exchange $\pi_i$ with $N_i$
 5: while $terminationCtr_i < d$ do
 6:     for all $s_i, s_{N_i}, s_u$ do
 7:         $B^0_i(\langle s_u, s_i, s_{N_i}, \langle\rangle\rangle) \leftarrow b_u(s_u)\cdot b_i(s_i)\cdot b_{N_i}(s_{N_i})$
 8:         $prevVal \overset{+}{\leftarrow} B^0_i(\langle s_u, s_i, s_{N_i}, \langle\rangle\rangle)\cdot$ EVALUATE$(i, s_i, s_u, s_{N_i}, \pi_i, \pi_{N_i}, \langle\rangle, \langle\rangle, 0, T)$
 9:     $gain_i \leftarrow$ GETVALUE$(i, B^0_i, \pi_{N_i}, 0, T) - prevVal$
10:     if $gain_i > 0$ then $terminationCtr_i \leftarrow 0$
11:     else $terminationCtr_i \overset{+}{\leftarrow} 1$
12:     Exchange $gain_i$,$terminationCtr_i$ with $N_i$
13:     $terminationCtr_i \leftarrow \mathbf{min}_{j\in N_i\cup\{i\}} terminationCtr_j$
14:     $maxGain \leftarrow \max_{j\in N_i\cup\{i\}} gain_j$
15:     $winner \leftarrow \mathrm{argmax}_{j\in N_i\cup\{i\}} gain_j$
16:     if $maxGain > 0$ and $i = winner$ then
17:         FINDPOLICY$(i, b, \langle\rangle, \pi_{N_i}, 0, T)$
18:         Communicate $\pi_i$ with $N_i$
19:     else if $maxGain > 0$ then
20:         Receive $\pi_{winner}$ from $winner$ and update $\pi_{N_i}$
21: return $\pi_i$
```

Algorithm 2 EVALUATE$(i, s^t_i, s^t_u, s^t_{N_i}, \pi_i, \pi_{N_i}, \vec{\omega}^t_i, \vec{\omega}^t_{N_i}, t, T)$

```
1: $a_i \leftarrow \pi_i(\vec{\omega}^t_i)$, $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}^t_{N_i})$
2: $val \leftarrow \sum_{l\in E} R_l\left(s^t_{l1},\ldots,s^t_{lk},s^t_u,a_{l1},\ldots,a_{lk}\right)$
3: if $t < T-1$ then
4:     for all $s^{t+1}_i, s^{t+1}_{N_i}, s^{t+1}_u$ do
5:         for all $\omega^{t+1}_i, \omega^{t+1}_{N_i}$ do
6:             $val \overset{+}{\leftarrow} P_u(s^t_u, s^{t+1}_u)\cdot P_i(s^t_i, s^t_u, a_i, s^{t+1}_i)\cdot P_{N_i}(s^t_{N_i}, s^t_u, a_{N_i}, s^{t+1}_{N_i})\cdot O_i(s^{t+1}_i, s^{t+1}_u, a_i, \omega^{t+1}_i)\cdot O_{N_i}(s^{t+1}_{N_i}, s^{t+1}_u, a_{N_i}, \omega^{t+1}_{N_i})\cdot$ EVALUATE$(i, s^{t+1}_i, s^{t+1}_u, s^{t+1}_{N_i}, \pi_i, \pi_{N_i}, \left\langle\vec{\omega}^t_i, \omega^{t+1}_i\right\rangle, \left\langle\vec{\omega}^t_{N_i}, \omega^{t+1}_{N_i}\right\rangle, t+1, T)$
7: return $val$
```

Finding Best Response

The algorithm, GETVALUE, for computing the best response is a dynamic-programming approach similar to that used in JESP. Here, we define an *episode* of agent *i* at time *t* as $e^t_i = \left\langle s^t_u, s^t_i, s^t_{N_i}, \vec{\omega}^t_{N_i}\right\rangle$. Treating episode as the state, results in a single agent POMDP, where the transition function and observation function can be defined as:

$$P'(e^t_i, a^t_i, e^{t+1}_i) = P_u(s^t_u, s^{t+1}_u)\cdot P_i(s^t_i, s^t_u, a^t_i, s^{t+1}_i)\cdot P_{N_i}(s^t_{N_i}, s^t_u, a^t_{N_i}, s^{t+1}_{N_i})\cdot O_{N_i}(s^{t+1}_{N_i}, s^{t+1}_u, a^t_{N_i}, \omega^{t+1}_{N_i})$$

$$O'(e^{t+1}_i, a^t_i, \omega^{t+1}_i) = O_i(s^{t+1}_i, s^{t+1}_u, a^t_i, \omega^{t+1}_i)$$

A multiagent belief state for an agent i given the distribution over the initial state, $b(s)$ is defined as:

$$B_i^t(e_i^t) = \Pr(s_u^t, s_i^t, s_{N_i}^t, \vec{\omega}_{N_i}^t | \vec{\omega}_i^t, \vec{a}_i^{t-1}, b)$$

The initial multiagent belief state for agent i, B_i^0, can be computed from b as follows:

$$B_i^0(\langle s_u, s_i, s_{N_i}, \langle\rangle\rangle) \leftarrow b_u(s_u) \cdot b_i(s_i) \cdot b_{N_i}(s_{N_i})$$

We can now compute the value of the best response policy via GETVALUE using the following equation (see Algorithm 3):

$$V_i^t(B_i^t) = \max_{a_i \in A_i} V_i^{a_i,t}(B_i^t) \tag{3}$$

Algorithm 3 GETVALUE$(i, B_i^t, \pi_{N_i}, t, T)$

```
1: if $t \geq T$ then return 0
2: if $V_i^t(B_i^t)$ is already recorded then return $V_i^t(B_i^t)$
3: $best \leftarrow -\infty$
4: for all $a_i \in A_i$ do
5:     $value \leftarrow$ GETVALUEACTION$(i, B_i^t, a_i, \pi_{N_i}, t, T)$
6:     record $value$ as $V_i^{a_i,t}(B_i^t)$
7:     if $value > best$ then $best \leftarrow value$
8: record $best$ as $V_i^t(B_i^t)$
9: return $best$
```

The function, $V_i^{a_i,t}$, can be computed using GETVALUE-ACTION(see Algorithm 4) as follows:

$$V_i^{a_i,t}(B_i^t) = \sum_{e_i^t} B_i^t(e_i^t) \sum_{l \in E \ s.t. \ i \in l} R_l(s_{l1}, \ldots, s_{lk}, s_u, \langle a_{l1}, \ldots, a_{lk}\rangle) + \sum_{\omega_i^{t+1} \in \Omega_1} \Pr(\omega_i^{t+1} | B_i^t, a_i) \cdot V_i^{t+1}\left(B_i^{t+1}\right) \tag{4}$$

B_i^{t+1} is the belief state updated after performing action a_i and observing ω_i^{t+1} and is computed using the function UPDATE (see Algorithm 5). Agent i's policy is determined from its value function $V_i^{a_i,t}$ using the function FINDPOLICY (see Algorithm 6).

Stochastic LID-JESP (SLID-JESP)

One of the criticisms of LID-JESP is that if an agent is the winner (maximum reward among its neighbors), then its precludes its neighbors from changing their policies too in that cycle. In addition, it will sometimes prevent its neighbor's neighbors (and may be their neighbors and so on) from changing their policies in that cycle even if they are actually independent. For example, consider three agents, a, b, c, arranged in a chain such that $gain_a > gain_b > gain_c$. In this situation, only a changes its policy is that cycle. However, c should have been able to change its policy too because it does not depend on a. This realization that LID-JESP allows limited parallelism led us to come up with a stochastic version of LID-JESP, SLID-JESP (Algorithm 7).

The key difference between LID-JESP and SLID-JESP is that in SLID-JESP if an agent i can improve its local neighborhood utility (i.e. $gain_i > 0$), it will do so with probability p, a predefined threshold probability (see lines 14-17). Note,

Algorithm 4 GETVALUEACTION$(i, B_i^t, a_i, \pi_{N_i}, t, T)$

```
1: $value \leftarrow 0$
2: for all $e_i^t = \left\langle s_u^t, s_i^t, s_{N_i}^t, \vec{\omega}_{N_i}^t \right\rangle$ s.t. $B_i^t(e_i^t) > 0$ do
3:     $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}_{N_i}^t)$
4:     $reward \leftarrow \sum_{l \in E} R_l\left(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, a_{l1}, \ldots, a_{lk}\right)$
5:     $value \overset{+}{\leftarrow} B_i^t(e_i^t) \cdot reward$
6: if $t < T - 1$ then
7:     for all $\omega_i^{t+1} \in \Omega_i$ do
8:         $B_i^{t+1} \leftarrow$ UPDATE$(i, B_i^t, a_i, \omega_i^{t+1}, \pi_{N_i})$
9:         $prob \leftarrow 0$
10:        for all $s_u^t, s_i^t, s_{N_i}^t$ do
11:            for all $e_i^{t+1} = \left\langle s_u^{t+1}, s_i^{t+1}, s_{N_i}^{t+1}, \left\langle \vec{\omega}_{N_i}^t, \omega_{N_i}^{t+1} \right\rangle \right\rangle$ s.t. $B_i^{t+1}(e_i^{t+1}) > 0$ do
12:                $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}_{N_i}^t)$
13:                $prob \overset{+}{\leftarrow} B_i^t(e_i^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i^{t+1}) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i}^{t+1})$
14:        $value \overset{+}{\leftarrow} prob \cdot$ GETVALUE$(i, B_i^{t+1}, \pi_{N_i}, t+1, T)$
15: return $value$
```

Algorithm 5 UPDATE$(i, B_i^t, a_i, \omega_i^{t+1}, \pi_{N_i})$

```
1: for all $e_i^{t+1} = \left\langle s_u^{t+1}, s_i^{t+1}, s_{N_i}^{t+1}, \left\langle \vec{\omega}_{N_i}^t, \omega_{N_i}^{t+1} \right\rangle \right\rangle$ do
2:     $B_i^{t+1}(e_i^{t+1}) \leftarrow 0$, $a_{N_i} \leftarrow \pi_{N_i}(\vec{\omega}_{N_i}^t)$
3:     for all $s_u^t, s_i^t, s_{N_i}^t$ do
4:         $B_i^{t+1}(e_i^{t+1}) \overset{+}{\leftarrow} B_i^t(e_i^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i^{t+1}) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i}^{t+1})$
5: normalize $B_i^{t+1}$
6: return $B_i^{t+1}$
```

that unlike LID-JESP, an agent's decision to change its policy does not depend on its neighbors' gain messages. However, the agents need to communicate their gain messages to their neighbors for termination detection.

Since there has been no change to the termination detection approach and the way gain is computed, the following proposition from LID-JESP(see (Nair *et al.* 2005)) hold for SLID-JESP as well.

Proposition 3 *SLID-JESP will terminate within d (= diameter) cycles iff agent are in a local optimum.*

This shows that the SLID-JESP will terminate if and only if agents are in a local optimum.

Hyper-link-based Decomposition (HLD)

Proposition 1 and Equation 2 show that the value function and the local neighborhood utility function can both be decomposed into components for each hyper-link in the *interaction hypergraph*. The Hyper-link-based Decomposition (HLD) technique exploits this decomposability for speeding up the algorithms EVALUATE and GETVALUE.

We introduce the following definitions to ease the description of HLD. Let $E_i = \{l | l \in E \wedge i \in l\}$ be the subset of hyper-

Algorithm 6 FINDPOLICY$(i, B_i^t, \vec{\omega}_i{}^t, \pi_{N_i}, t, T)$

```
1: $a_i^* \leftarrow \text{argmax}_{a_i} V_i^{a_i,t}(B_i^t)$, $\pi_i(\vec{\omega}_i{}^t) \leftarrow a_i^*$
2: if $t < T - 1$ then
3:     for all $\omega_i^{t+1} \in \Omega_i$ do
4:         $B_i^{t+1} \leftarrow$ UPDATE$(i, B_i^t, a_i^*, \omega_i^{t+1}, \pi_{N_i})$
5:         FINDPOLICY$(i, B_i^{t+1}, \langle \vec{\omega}_i{}^t, \omega_i^{t+1} \rangle, \pi_{N_i}, t+1, T)$
6: return
```

Algorithm 7 SLID-JESP$(i, \text{ND-POMDP}, p)$

```
0: {lines 1-4 same a LID-JESP}
5: while $terminationCtr_i < d$ do {lines 6-13 same as LID-JESP}
14:     if RANDOM() $< p$ and $gain_i > 0$ then
15:         FINDPOLICY$(i, b, \langle\rangle, \pi_{N_i}, 0, T)$
16:         Communicate $\pi_i$ with $N_i$
17:     Receive $\pi_j$ from all $j \in N_i$ that changed their policies
18: return $\pi_i$
```

links that contain agent i. Note that $N_i = \cup_{l \in E_i} l - \{i\}$, i.e. the neighborhood of i contains all the agents in E_i except agent i itself. $S_l = \times_{j \in l} S_j$ refers to the states of agents in link l. Similarly, $A_l = \times_{j \in l} A_j$, $\Omega_l = \times_{j \in l} \Omega_l$, $P_l(s_l, a_l, s_l') = \prod_{j \in l} P_j(s_j, a_j, s_j')$, and $O_l(s_l, a_l, \omega_l) = \prod_{j \in l} O_j(s_j, a_j, \omega_j)$. Further, we define $b_l = \prod_{j \in l} b_j(s_j)$, where b_j is the distribution over agent j's initial state. Using the above definitions, we can rewrite Equation 2 as:

$$V_\pi[N_i] = \sum_{l \in E_i} \sum_{s_l, s_u} b_u(s_u) \cdot b_l(s_l) \cdot V_{\pi_l}^0(s_l, s_u, \langle\rangle, \ldots, \langle\rangle) \quad (5)$$

EVALUATE-HLD (Algorithm 9) is used to compute the local neighborhood utility of a hyperlink l (inner loop of Equation 8). When the joint policy is completely specified, the expected reward from each hyper-link can be computed independently (as in EVALUATE-HLD). However, when trying to find the optimal best response, we cannot optimize on each hyper-link separately since in any belief state, an agent can perform only one action. The optimal best response in any belief state is the action that maximizes the sum of the expected rewards on each of its hyper-links.

The algorithm, GETVALUE-HLD, for computing the best response is a modification of the GETVALUE function that attempts to exploit the decomposability of the value function without violating the constraint that the same action must be applied to all the hyper-links in a particular belief state. Here, we define an *episode* of agent i for a hyper-link l at time t as $e_{il}^t = \langle s_u^t, s_l^t, \vec{\omega}_{l-\{i\}}^t \rangle$. Treating episode as the state, the transition and observation functions can be defined as:

$$\begin{aligned} P_{il}'(e_{il}^t, a_i^t, e_{il}^{t+1}) =& P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l^t, s_l^{t+1}) \\ & \cdot O_{l-\{i\}}(s_{l-\{i\}}^{t+1}, s_u^{t+1}, a_{l-\{i\}}^t, \omega_{l-\{i\}}^{t+1}) \\ O_{il}'(e_i^{t+1}, a_i^t, \omega_i^{t+1}) =& O_i(s_i^{t+1}, s_u^{t+1}, a_i^t, \omega_i^{t+1}) \end{aligned}$$

where $a_{l-\{i\}}^t = \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$. We can now define the multiagent belief state for an agent i with respect to hyper-link $l \in E_i$ as:

$$B_{il}^t(e_{il}^t) = \Pr(s_u^t, s_l^t, \vec{\omega}_{l-\{i\}}^t | \vec{\omega}_i^t, \vec{a}_i^{t-1}, b)$$

We redefine the multiagent belief state of agent i as :

$$B_i^t(e_i^t) = \{B_{il}^t(e_{il}^t) | l \in E_i\}$$

We can now compute the value of the best response policy using the following equation:

$$V_i^t(B_i^t) = \max_{a_i \in A_i} \left(\sum_{l \in E_i} V_{il}^{a_i,t}(B_i^t) \right) \quad (6)$$

The value of the best response policy for the link l can be computed as follows:

$$V_{il}^t(B_i^t) = V_{il}^{a_i^*,t}(B_i^t) \quad (7)$$

where $a_i^* = \arg\max_{a_i \in A_i} \left(\sum_{l \in E_i} V_{il}^{a_i,t}(B_i^t) \right)$. The function GETVALUE-HLD (see Algorithm 10) computes the term $V_{il}^t(B_i^t)$ for all links $l \in E_i$.

The function, $V_{il}^{a_i,t}$, can be computed as follows:

$$\begin{aligned} V_{il}^{a_i,t}(B_i^t) =& \sum_{e_{il}^t} B_{il}^t(e_{il}^t) \cdot R_l(s_l, s_u, a_l) \\ & + \sum_{\omega_i^{t+1} \in \Omega_1} \Pr(\omega_i^{t+1} | B_i^t, a_i) \cdot V_{il}^{t+1}\left(B_i^{t+1}\right) \quad (8) \end{aligned}$$

The function GETVALUEACTION-HLD(see Algorithm 11) computes the above value for all links l. B_i^{t+1} is the belief state updated after performing action a_i and observing ω_i^{t+1} and is computed using UPDATE-HLD (see Algorithm 12). Agent i's policy is determined from its value function $V_i^{a_i,t}$ using FINDPOLICY-HLD (see Algorithm 13).

The reason why HLD will reduce the run time for finding the best response is that the optimal value function is computed for each link separately. This reduction in runtime is borne out by our complexity analysis and experimental results as well.

Algorithm 8 LID-JESP-HLD$(i, \text{ND-POMDP})$

```
0: {lines 1-4 same a LID-JESP}
5: while $terminationCtr_i < d$ do
6:     for all $s_u$ do
7:         for all $l \in E_i$ do
8:             for all $s_l \in S_l$ do
9:                 $B_{il}^0(\langle s_u, s_l, \langle\rangle \rangle) \leftarrow b_u(s_u) \cdot b_l(s_l)$
10:                $prevVal \overset{+}{\leftarrow} B_{il}^0(\langle s_u, s_l, \langle\rangle \rangle) \cdot$
                   EVALUATE-HLD$(l, s_l, s_u, \pi_l, \langle\rangle, 0, T)$
11:    $gain_i \leftarrow$ GETVALUE-HLD$(i, B_i^0, \pi_{N_i}, 0, T) - prevVal$
12:    if $gain_i > 0$ then $terminationCtr_i \leftarrow 0$
13:    else $terminationCtr_i \overset{+}{\leftarrow} 1$
14:    Exchange $gain_i$,$terminationCtr_i$ with $N_i$
15:    $terminationCtr_i \leftarrow \min_{j \in N_i \cup \{i\}} terminationCtr_j$
16:    $maxGain \leftarrow \max_{j \in N_i \cup \{i\}} gain_j$
17:    $winner \leftarrow \text{argmax}_{j \in N_i \cup \{i\}} gain_j$
18:    if $maxGain > 0$ and $i = winner$ then
19:        FINDPOLICY-HLD$(i, B_i^0, \langle\rangle, \pi_{N_i}, 0, T)$
20:        Communicate $\pi_i$ with $N_i$
21:    else if $maxGain > 0$ then
22:        Receive $\pi_{winner}$ from $winner$ and update $\pi_{N_i}$
23: return $\pi_i$
```

Algorithm 9 EVALUATE-HLD$(l, s_l^t, s_u^t, \pi_l, \vec{\omega}_l^t, t, T)$

```
1: $a_l \leftarrow \pi_l(\vec{\omega}_l^t)$
2: $val \leftarrow R_l\left(s_l^t, s_u^t, a_l\right)$
3: if $t < T-1$ then
4:     for all $s_l^{t+1}, s_u^{t+1}$ do
5:         for all $\omega_l^{t+1}$ do
6:             $val \overset{+}{\leftarrow} P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l, s_l^{t+1}) \cdot O_l(s_l^{t+1}, s_u^{t+1}, a_l, \omega_l^{t+1}) \cdot \text{EVALUATE-HLD}\left(l, s_l^{t+1}, s_u^{t+1}, \pi_l, \left\langle \vec{\omega}_l^t, \omega_l^{t+1} \right\rangle, t+1, T\right)$
7: return $val$
```

Algorithm 10 GETVALUE-HLD$(i, B_i^t, \pi_{N_i}, t, T)$

```
1: if $t \geq T$ then return 0
2: if $V_{il}^t(B_i^t)$ is already recorded $\forall l \in E_i$ then return $[V_{il}^t(B_i^t)]_{l \in E_i}$
3: $bestSum \leftarrow -\infty$
4: for all $a_i \in A_i$ do
5:     $value \leftarrow$ GETVALUEACTION-HLD$(i, B_i^t, a_i, \pi_{N_i}, t, T)$
6:     $valueSum \leftarrow \sum_{l \in E_i} value[l]$
7:     record $valueSum$ as $V_i^{a_i,t}(B_i^t)$
8:     if $valueSum > bestSum$ then $best \leftarrow value, bestSum \leftarrow valueSum$
9: for all $l \in E_i$ do
10:     record $best[l]$ as $V_{il}^t(B_i^t)$
11: return $best$
```

Algorithm 11 GETVALUEACTION-HLD$(i, B_i^t, a_i, \pi_{N_i}, t, T)$

```
1: for all $l \in E_i$ do
2:     $value[l] \leftarrow 0$
3:     for all $e_{il}^t = \left\langle s_u^t, s_l^t, \vec{\omega}_{l-\{i\}}^t \right\rangle$ s.t. $B_{il}^t(e_{il}^t) > 0$ do
4:         $a_{l-\{i\}} \leftarrow \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$
5:         $value[l] \overset{+}{\leftarrow} B_{il}^t(e_{il}^t) \cdot R_l\left(s_l^t, s_u^t, a_l\right)$
6: if $t < T-1$ then
7:     for all $\omega_i^{t+1} \in \Omega_i$ do
8:         for all $l \in E_i$ do
9:             $B_{il}^{t+1} \leftarrow$ UPDATE-HLD$(i, l, B_{il}^t, a_i, \omega_i^{t+1}, \pi_{l-\{i\}})$
10:            $prob[l] \leftarrow 0$
11:            for all $s_u^t, s_l^t$ do
12:                for all $e_{il}^{t+1} = \left\langle s_u^{t+1}, s_l^{t+1}, \left\langle \vec{\omega}_{l-\{i\}}^t, \omega_{l-\{i\}}^{t+1} \right\rangle \right\rangle$ s.t. $B_{il}^{t+1}(e_{il}^{t+1}) > 0$ do
13:                    $a_{l-\{i\}} \leftarrow \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$
14:                    $prob[l] \overset{+}{\leftarrow} B_{il}^t(e_{il}^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l, s_l^{t+1}) \cdot O_l(s_l^{t+1}, s_u^{t+1}, a_l, \omega_l^{t+1})$
15:        $futureValue \leftarrow$ GETVALUE-HLD$(i, B_i^{t+1}, \pi_{N_i}, t+1, T)$
16:        for all $l \in E_i$ do
17:            $value[l] \overset{+}{\leftarrow} prob[l] \cdot futureValue[l]$
18: return $value$
```

Algorithm 12 UPDATE-HLD$(i, l, B_{il}^t, a_i, \omega_i^{t+1}, \pi_{l-\{i\}})$

```
1: for all $e_{il}^{t+1} = \left\langle s_u^{t+1}, s_l^{t+1}, \left\langle \vec{\omega}_{l-\{i\}}^t, \omega_{l-\{i\}}^{t+1} \right\rangle \right\rangle$ do
2:     $B_{il}^{t+1}(e_{il}^{t+1}) \leftarrow 0$
3:     $a_{l-\{i\}} \leftarrow \pi_{l-\{i\}}(\vec{\omega}_{l-\{i\}}^t)$
4:     for all $s_u^t, s_l^t$ do
5:         $B_{il}^{t+1}(e_{il}^{t+1}) \overset{+}{\leftarrow} B_{il}^t(e_{il}^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_l(s_l^t, s_u^t, a_l, s_l^{t+1}) \cdot O_l(s_l^{t+1}, s_u^{t+1}, a_l, \omega_l^{t+1})$
6: normalize $B_{il}^{t+1}$
7: return $B_{il}^{t+1}$
```

Algorithm 13 FINDPOLICY-HLD$(i, B_i^t, \vec{\omega}_i^{\,t}, \pi_{N_i}, t, T)$

```
1: $a_i^* \leftarrow \text{argmax}_{a_i} V_i^{a_i,t}(B_i^t)$
2: $\pi_i(\vec{\omega}_i^{\,t}) \leftarrow a_i^*$
3: if $t < T-1$ then
4:     for all $\omega_i^{t+1} \in \Omega_i$ do
5:         for all $l \in E_i$ do
6:             $B_{il}^{t+1} \leftarrow$ UPDATE-HLD$(i, l, B_{il}^t, a_i^*, \omega_i^{t+1}, \pi_{l-\{i\}})$
7:         FINDPOLICY-HLD$(i, B_i^{t+1}, \left\langle \vec{\omega}_i^{\,t}, \omega_i^{t+1} \right\rangle, \pi_{N_i}, t+1, T)$
8: return
```

Complexity Results

The complexity of the finding the optimal best response for agent i for JESP (using the dynamic programming(Nair *et al.* 2003)) is $O(|S|^2 \cdot |A_i|^T \cdot \prod_{j \in \{1 \ldots n\}} |\Omega_j|^T)$. Note that the complexity depends on the number world states $|S|$ and the number of possible observation histories of all the agents.

In contrast, the complexity of finding the optimal best response for i for LID-JESP (and SLID-JESP) is $O(\prod_{l \in E_i}[|S_u \times S_l|^2 \cdot |A_i|^T \cdot |\Omega_l|^T])$. It should be noted that in this case, the complexity depends on the number of states $|S_u|$, $|S_i|$ and $|S_{N_i}|$ and not on the number of states of any non-neighboring agent. Similarly, the complexity depends on only the number of observation histories of i and its neighbors and not those of all the agents. This highlights the reason for why LID-JESP and SLID-JESP are superior to JESP for problems where locality of interaction can be exploited.

The complexity for computing optimal best response for i in LID-JESP with HLD (and SLID-JESP with HLD) is $O(\Sigma_{l \in E_i}[|S_u \times S_l|^2 \cdot |A_i|^T \cdot |\Omega_l|^T])$. Key difference of note compared to the complexity expression for LID-JESP, is the replacement of product,$\prod$ with a sum,Σ. Thus, as number of neighbors increases, difference between the two approaches increases.

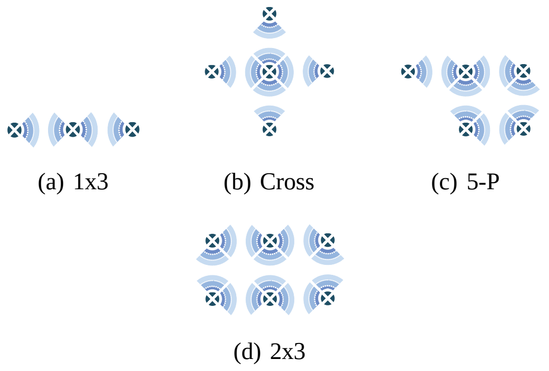
Since JESP is a centralized algorithm, the best response function is performed for each agent serially. LID-JESP and SLID-JESP (with and without HLD), in contrast, are distributed algorithms, where each agent can be run in parallel on a different processor, further alleviating the large complexity of finding the optimal best response.

Experimental Results

In this section, we performed comparison of the algorithms – LID-JESP, SLID-JESP, LID-JESP with HLD and SLID-JESP with HLD – in terms of value and runtime. We used four different topologies of sensors, shown in Figure 2, each with a different target movement scenario. With 2 targets moving in the environment, possible positions of targets are increased as the network grows and the number of unaffected states are increased accordingly. Figure 2(a) shows the example where there are 3 sensors arranged in a chain and the number of possible positions for each target is 1. In the cross topology, as in Figure 2(b), we considered 5 sensors with one sensor in the center surrounded by 4 sensors and 2 locations are possible for each target. In the example in Figure 2(c) with 5 sensors arranged in P shape, target1 and target2 can be at 2 and 3 locations respectively, thus leading to a total of 12 states. There are total 20 states for six sensors in example of Figure 2(d) with 4 and 3 locations for target1 and target2, respectively. As we assumed earlier, each target is independent of each other. Thus, total number of unaffected states are $(\prod_{targets}(\text{number of possible positions of each target} + 1))$. Due to the exponentially increasing runtime, the size of the network and time horizon is limited but is still significantly larger than those which have previously been demonstrated in distributed POMDPs. All experiments are started at random initial policies and averaged over five runs for each algorithm. We chose 0.9 as the threshold probability (p) for SLID-JESP which empirically gave a good result for most cases.

Figure 3 shows performance improvement of SLID-JESP and HLD in terms of runtime. In Figure 3, X-axis shows the time horizon T, while Y-axis shows the runtime in milliseconds on a logarithmic scale. In all cases of Figure 3, the line of SLID-JESP is lower than that of LID-JESP with and without HLD where the difference of two grows as the network grows. As in Figure 3(c) and Figure 3(d) the difference in runtime between LID-JESP and SLID-JESP is bigger than that in smaller network examples. The result that SLID-JESP always takes less time than LID-JESP is because in SLID-JESP, more agents change their policy in one cycle, and hence SLID-JESP tends to converge to a local optimum quickly. As for HLD, all the graphs shows that the use of Hyper-link-based decomposition clearly improved LID-JESP and SLID-JESP in terms of runtime. The improvement is more visible when the number of neighbors increases where HLD takes advantage of decomposition. For example, in Figure 3(b), by using HLD the runtime reduced by more than an order of magnitude for $T = 4$. In cross topology, the computation for the agent in the center which has 4 neighbors is a main bottleneck and HLD significantly reduces the computation by decomposition.

Figure 4 shows the values of each algorithm for different topologies. In Figure 4, X-axis shows the time horizon T, while Y-axis shows the value of team reward. There are only two lines in each graph because the values of the algorithm with HLD and without HLD are always the same because HLD only exploits independence between neighbors and doesn't affect the value of the resulting joint policy. The reward of LID-JESP is larger than that of SLID-JESP in three out of the four topologies that we tried. This suggests SLID-JESP's greedy approach to changing the joint policy causes it to converge to smaller local optima than LID-JESP in some cases. However, note that in Figure 4(a) SLID-JESP converges to a greater local optima than LID-JESP. This suggests that network topology greatly impacts the choice of whether to use LID-JESP or SLID-JESP. Further, the results of SLID-JESP vary in value for different threshold probabilities, however there is a consistent trend that the result is better when the threshold probability (p) is large. This trend means that in our domain, it is generally better to change policy if there is a visible gain.


(a) 1x3 (b) Cross (c) 5-P

(d) 2x3

Figure 2: Different sensor net configurations.

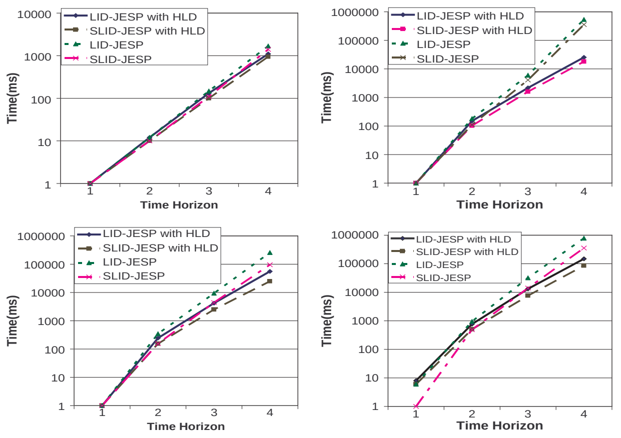


Figure 3: Runtime (ms) for (a) 1x3, (b) cross, (c) 5-P and (d) 2x3.

Summary and Related Work

In this paper, we presented a stochastic variation of the LID-JESP that is based on DSA (distributed stochastic algorithm) that allows neighboring agents to change their policies in the same cycle. Through detailed experiments, we showed how this can result in speedups without a large difference in solution quality. We also introduced a technique called *hyper-*

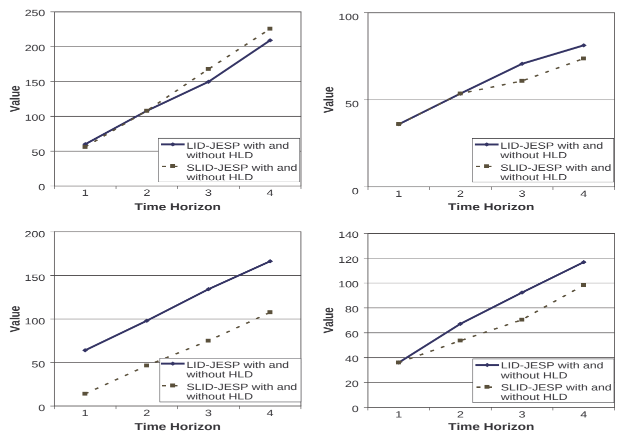


Figure 4: Value for (a) 1x3, (b) cross, (c) 5-P and (d) 2x3.

link-based decomposition (HLD) that allows us to exploit locality of interaction further, resulting in faster run times for both LID-JESP and its stochastic variant without any loss in solution quality.

Among related work, we have earlier discussed the relationship of our work to key DCOP and distributed POMDP algorithms, i.e., we synthesize new algorithms by exploiting their synergies. We now discuss some other recent algorithms for locally and globally optimal policy generation for distributed POMDPs. For instance, Hansen *et al.* (2004) present an exact algorithm for partially observable stochastic games (POSGs) based on dynamic programming and iterated elimination of dominant policies. Emery-Montemerlo *et al.* (2004) approximate POSGs as a series of one-step Bayesian games using heuristics to find the future discounted value for actions. We have earlier discussed Nair *et al.* (2003)'s JESP algorithm that uses dynamic programming to reach a local optimal. In addition, Becker *et al.*'s work (2004) on transition-independent distributed MDPs is related to our assumptions about transition and observability independence in ND-POMDPs. These are all centralized policy generation algorithms that could benefit from the key ideas in this paper — that of exploiting local interaction structure among agents to (i) enable distributed policy generation; (ii) limit policy generation complexity by considering only interactions with "neighboring" agents. Guestrin *et al.* (2002), present "coordination graphs" which have similarities to constraint graphs. The key difference in their approach is that the "coordination graph" is obtained from the value function which is computed in a centralized manner. The agents then use a distributed procedure for online action selection based on the coordination graph. In our approach, the value function is computed in a distributed manner. Dolgov and Durfee's algorithm (2004) exploits network structure in multiagent MDPs (not POMDPs) but assume that each agent tried to optimize its individual utility instead of the team's utility.

Acknowledgments

This material is based upon work supported by the DARPA/IPTO COORDINATORS program and the Air Force Research Laboratory under Contract No. FA8750–05–C–0030. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

References

Becker, R.; Zilberstein, S.; Lesser, V.; and Goldman, C. 2004. Solving transition independent decentralized Markov decision processes. *JAIR* 22:423–455.

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

Dolgov, D., and Durfee, E. 2004. Graphical models in local, asymmetric multi-agent markov decision processes. In *AAMAS*.

Goldman, C., and Zilberstein, S. 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. *JAIR* 22:143–174.

Guestrin, C.; Venkataraman, S.; and Koller, D. 2002. Context specific multiagent coordination and planning with factored MDPs. In *AAAI*.

Hansen, E.; Bernstein, D.; and Zilberstein, S. 2004. Dynamic Programming for Partially Observable Stochastic Games. In *AAAI*.

Lesser, V.; Ortiz, C.; and Tambe, M. 2003. *Distributed sensor nets: A multiagent perspective*. Kluwer.

Modi, P. J.; Shen, W.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. In *AAMAS*.

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

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

Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 133–139.

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

Yokoo, M., and Hirayama, K. 1996. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *ICMAS*.

Zhang, W.; Xing, Z.; Wang, G.; and Wittenberg, L. 2003. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS*.