Networked Distributed POMDPs: DCOP-Inspired Distributed POMDPs

Ranjit Nair[1], Pradeep Varakantham[2], Milind Tambe[2], and Makoto Yokoo[3]

[1] Honeywell Laboratories,Minneapolis, MN 55418
ranjit.nair@honeywell.com
[2] University of Southern California, Los Angeles, CA 90089
{varakant,tambe}@usc.edu
[3] Kyushu University, 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. This paper present a new model synthesized from distributed POMDPs and DCOPs, called Networked Distributed POMDPs (ND-POMDPs). Exploiting network structure enables us to present two novel algorithms for ND-POMDPs: a distributed policy generation algorithm that performs local search and a systematic policy search that is guaranteed to reach the global optimal.

1 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 [1–3]. Unfortunately, as shown by Bernstein *et al.* [4], the problem of finding the optimal joint policy for a general distributed POMDP is NEXP-Complete. This complexity has fuelled a lot of criticism that distributed POMDPs cannot be applied to large scale realistic problems. 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 [1, 5]. Second, they have focused on restricted types of domains, e.g. with transition independence or collective observability [3]. 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.

This paper introduces a third complementary approach called Networked Distributed POMDPs (ND-POMDPs), that is motivated by domains such as distributed sensor nets [6], 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) [7, 8]. 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 [6].

Taking inspiration from DCOP algorithms, we provide two algorithms for ND-POMDPs. First, the LID-JESP algorithm combines the existing JESP algorithm of Nair *et al.* [1] and the *DBA* [8] DCOP algorithm. LID-JESP thus combines the dynamic programming of JESP with the innovation that it is uses off-line distributed policy generation instead of JESP's centralized policy generation. Second, we present a more systematic policy search that is guaranteed to reach the global optimal on tree-structured agent-interaction graphs; and illustrate that by exploiting properties from constraint literature, it can guarantee optimality in general. Finally, by empirically comparing the performance of the two algorithms with benchmark algorithms that do not exploit network structure, we illustrate the gains in efficiency made possible by exploiting network structure in ND-POMDPs.

2 Illustrative Domain

We describe an illustrative problem within the distributed sensor net domain, motivated by the real-world challenge in [6][4]. 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 effect 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.

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

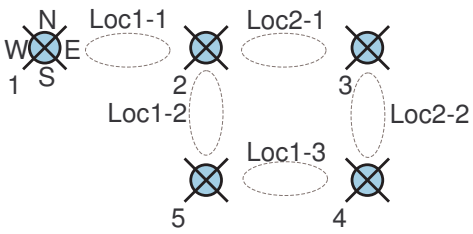

Fig. 1. Sensor net scenario: If present, target1 is in Loc1-1, Loc1-2 or Loc1-3, and target2 is in Loc2-1 or Loc2-2.

3 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 $S = \times_{1\leq i\leq n} S_i \times S_u$ 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. 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. $A = \times_{1\leq i\leq n} A_i$ 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 $P(s, a, s') = P_u(s_u, s'_u) \cdot \prod_{1\leq i\leq n} P_i(s_i, s_u, a_i, s'_i)$, 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. 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.* [3] also relied on transition independence, and Goldman and Zilberstein [9] 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.

$\Omega = \times_{1\leq i\leq n} \Omega_i$ 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 $O(s, a, \omega) = \prod_{1\leq i\leq n} O_i(s_i, s_u, a_i, \omega_i)$, 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 $O_i(s_i, s_u, a_i, \omega_i) = \Pr(\omega_i|s_1, s_u, a_i)$.

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}, Ω_{N_i}, P_{N_i} and O_{N_i}.

b, the distribution over the initial state, 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 the distributions over initial unaffectable state and over 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 $\pi = \langle \pi_i, \ldots, \pi_n \rangle$ 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 [8], an algorithm for distributed constraint satisfaction, to develop an algorithm for solving ND-POMDPs.

The following proposition 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* $R(s, a) = \sum_{l \in E} R_l(s_{l1}, \ldots, s_{lk}, s_u, \langle a_{l1}, \ldots, a_{lk} \rangle)$,

$$V_\pi^t(s^t, \boldsymbol{\omega}^t) = \sum_{l \in E} V_{\pi_l}^t(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, \boldsymbol{\omega}_{l1}^t, \ldots \boldsymbol{\omega}_{lk}^t) \tag{1}$$

where $V_\pi^t(s^t, \boldsymbol{\omega})$ *is the expected reward from the state* s^t *and joint observation history* $\boldsymbol{\omega}^t$ *for executing policy* π*, and* $V_{\pi_l}^t(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, \boldsymbol{\omega}_{l1}^t, \ldots \boldsymbol{\omega}_{lk}^t)$ *is the expected reward for executing* π_l *accruing from the component* R_l.

Proof: Proposition holds for $t = T - 1$ (no future reward). Assume it holds for $t = \tau$ where $1 \leq \tau < T - 1$. Thus,

$$V_\pi^\tau(s^\tau, \boldsymbol{\omega}^\tau) = \sum_{l \in E} V_{\pi_l}^\tau(s_{l1}^\tau, \ldots, s_{lk}^\tau, s_u^\tau, \boldsymbol{\omega}_{l1}^\tau, \ldots \boldsymbol{\omega}_{lk}^\tau)$$

We introduce the following abbreviations:

$$p_i^t \stackrel{\triangle}{=} P_i(s_i^t, s_u^t, \pi_i(\boldsymbol{\omega}_i^t), s_i^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, \pi_i(\boldsymbol{\omega}_i^t), \omega_i^t)$$
$$p_u^t \stackrel{\triangle}{=} P_i(s_u^t, s_u^{t+1})$$

$$r_l^t \stackrel{\triangle}{=} R_l(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, \pi_{l1}(\boldsymbol{\omega}_{l1}^t), \ldots, \pi_{lk}(\boldsymbol{\omega}_{lk}^t))$$
$$v_l^t \stackrel{\triangle}{=} V_{\pi_l}^t(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, \boldsymbol{\omega}_{l1}^t, \ldots \boldsymbol{\omega}_{lk}^t)$$

We show that proposition holds for $t = \tau - 1$,

$$V_\pi^{\tau-1}(s^{\tau-1}, \boldsymbol{\omega}^{\tau-1}) = \sum_{l \in E} r_l^{\tau-1} + \sum_{s^\tau, \omega^\tau} p_u^{\tau-1} p_1^{\tau-1} \dots p_n^{\tau-1} \sum_{l \in E} v_l^\tau$$
$$= \sum_{l \in E} (r_l^{\tau-1} + \sum_{s_{l1}^\tau, \dots, s_{lk}^\tau, s_u^\tau, \omega_{l1}^\tau, \dots, \omega_{lk}^\tau} p_{l1}^{\tau-1} \dots p_{lk}^{\tau-1} p_u^{\tau-1} v_l^\tau) = \sum_{l \in E} v_l^{\tau-1} \quad \square$$

Local neighborhood utility of agent i as the expected reward accruing due to the hyper-links that contain agent i:

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

Proposition 2. Locality of interaction: *The local neighborhood utility $V_\pi[N_i] = V_{\pi'}[N_i]$ if $\pi_i = \pi_i'$ and $\pi_{N_i} = \pi_{N_i}'$.*

Proof sketch: Equation 2 sums over $l \in E$ such that $i \in l$, and hence any change of the policy of an agent $j \notin i \cup N_i$ cannot affect $V_\pi[N_i]$. Thus, any such policy assignment, π' that has different policies for only non-neighborhood agents, has equal value as $V_\pi[N_i]$. $\square$

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.

4 Locally Optimal Policy Generation

The locally optimal policy generation algorithm called LID-JESP (Locally interacting distributed joint equilibrium search for policies) is based on the DBA algorithm [8] and JESP [1]. 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[5], 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

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

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 becomes equal to the diameter of the interaction hypergraph.

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

Compute interaction hypergraph and N_i
$d \leftarrow$ diameter of hypergraph, $terminationCtr_i \leftarrow 0$
$\pi_i \leftarrow$ randomly selected policy, $prevVal \leftarrow 0$
Exchange π_i with N_i
while $terminationCtr_i < d$ **do**
  for all s_i, s_{N_i}, s_u **do**
    $prevVal \stackrel{+}{\leftarrow} b_u(s_u) \cdot b_i(s_i) \cdot b_{N_i}(s_{N_i}) \cdot \text{EVALUATE}(i, s_i, s_u, s_{N_i}, \pi_i, \pi_{N_i}, \langle\rangle, \langle\rangle, 0, T)$
  $gain_i \leftarrow \text{GETVALUE}(i, b, \pi_{N_i}, 0, T) - prevVal$
  if $gain_i > 0$ **then** $terminationCtr_i \leftarrow 0$
  else $terminationCtr_i \stackrel{+}{\leftarrow} 1$
  Exchange $gain_i$,$terminationCtr_i$ with N_i
  $terminationCtr_i \leftarrow \mathbf{min}_{j \in N_i \cup \{i\}} terminationCtr_j$
  $maxGain \leftarrow \max_{j \in N_i \cup \{i\}} gain_j$
  $winner \leftarrow \text{argmax}_{j \in N_i \cup \{i\}} gain_j$
  if $maxGain > 0$ **and** $i = winner$ **then**
    $\text{FINDPOLICY}(i, b, \langle\rangle, \pi_{N_i}, 0, T)$
    Communicate π_i with N_i
  else if $maxGain > 0$ **then**
    Receive π_{winner} from $winner$ and update π_{N_i}
return π_i

Algorithm 2 $\text{EVALUATE}(i, s_i^t, s_u^t, s_{N_i}^t, \pi_i, \pi_{N_i}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_{N_i}, t, T)$

$a_i \leftarrow \pi_i(\boldsymbol{\omega}_i)$, $a_{N_i} \leftarrow \pi_{N_i}(\boldsymbol{\omega}_{N_i})$
$val \leftarrow \sum_{l \in E} R_l\left(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, a_{l1}, \ldots, a_{lk}\right)$
if $t < T - 1$ **then**
  for all $s_i^{t+1}, s_{N_i}^{t+1}, s_u^{t+1}$ **do**
    for all ω_i, ω_{N_i} **do**
      $val \stackrel{+}{\leftarrow} P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i}) \cdot \text{EVALUATE}(i, s_i^{t+1}, s_u^{t+1}, s_{N_i}^{t+1}, \pi_i, \pi_{N_i}, \langle \boldsymbol{\omega}_i, \omega_i \rangle, \langle \boldsymbol{\omega}_{N_i}, \omega_{N_i} \rangle, t+1, T)$
return val

4.1 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

$e_i^t = \langle s_u^t, s_i^t, s_{N_i}^t, \boldsymbol{\omega}_{N_i}^t \rangle$. Treating episode as the state, results in a single agent POMDP, where the transition function and observation function can be defined as:

$$\begin{aligned} P'(e_i^t, a_i^t, e_i^{t+1}) =& P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i^t, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, \\ & s_u^t, a_{N_i}^t, s_{N_i}^{t+1}) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}^t, \omega_{N_i}^{t+1}) \\ O'(e_i^{t+1}, a_i^t, \omega_i^{t+1}) =& O_i(s_i^{t+1}, s_u^{t+1}, a_i^t, \omega_i^{t+1}) \end{aligned}$$

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

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

We can now compute the best response using the following equation (see Algorithm 3):

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

The function, $V^{a_i,t}$, can be computed using Algorithm 4 as follows:

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

B_i^{t+1} is the belief state updated after performing action a_i and observing ω_i^{t+1} and is computed using Algorithm 5.

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

```
if t ≥ T then return 0
if V_t(B^t) is already recorded then return V_t(B^t)
best ← −∞
for all a_i ∈ A_i do
    value ← GETVALUEACTION(i, B^t, a_i, π_{N_i}, t, T)
    record value as V_t^{a_i}(B^t)
    if value > best then best ← value
record best as V_t(B^t)
return best
```

4.2 Correctness Results

Proposition 3. *When applying LID-JESP, the global utility is strictly increasing until local optimum is reached.*

Proof sketch By construction, only non-neighboring agents can modify their policies in the same cycle. Agent i chooses to change its policy if it can improve upon its local

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

$value \leftarrow 0$
for all $e^t = \left\langle s_u^t, s_i^t, s_{N_i}^t, \boldsymbol{\omega}_{N_i} \right\rangle$ s.t. $B^t(e^t) > 0$ **do**
  $a_{N_i} \leftarrow \pi_{N_i}(\boldsymbol{\omega}_{N_i})$
  $reward \leftarrow \sum_{l \in E} R_l \left(s_{l1}^t, \ldots, s_{lk}^t, s_u^t, a_{l1}, \ldots, a_{lk}\right)$
  $value \overset{+}{\leftarrow} B^t(e^t) \cdot reward$
if $t < T - 1$ **then**
  for all $\omega_i \in \Omega_i$ **do**
    $B^{t+1} \leftarrow$ UPDATE$(i, B^t, a, \omega_i, \pi_{N_i})$
    $prob \leftarrow 0$
    for all $s_u^t, s_i^t, s_{N_i}^t$ **do**
      for all $e^{t+1} = \left\langle s_u^{t+1}, s_i^{t+1}, s_{N_i}^{t+1}, \langle \boldsymbol{\omega}_{N_i}, \omega_{N_i} \rangle \right\rangle$ s.t. $B^{t+1}(e^{t+1}) > 0$ **do**
        $a_{N_i} \leftarrow \pi_{N_i}(\boldsymbol{\omega}_{N_i})$
        $prob \overset{+}{\leftarrow} B^t(e^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i})$
    $value \overset{+}{\leftarrow} prob \cdot$ GETVALUE$(i, B^{t+1}, \pi_{N_i}, t+1, T)$
return $value$

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

for all e^{t+1} **do**
  $B^{t+1}(e^{t+1}) \leftarrow 0$, $a_{N_i} \leftarrow \pi_{N_i}(\boldsymbol{\omega}_{N_i})$
  for all $s^t \in S$ **do**
    $B^{t+1}(e^{t+1}) \overset{+}{\leftarrow} B^t(e^t) \cdot P_u(s_u^t, s_u^{t+1}) \cdot P_i(s_i^t, s_u^t, a_i, s_i^{t+1}) \cdot P_{N_i}(s_{N_i}^t, s_u^t, a_{N_i}, s_{N_i}^{t+1}) \cdot O_i(s_i^{t+1}, s_u^{t+1}, a_i, \omega_i) \cdot O_{N_i}(s_{N_i}^{t+1}, s_u^{t+1}, a_{N_i}, \omega_{N_i})$
normalize B^{t+1}
return B^{t+1}

Algorithm 6 FINDPOLICY$(i, B^t, \boldsymbol{\omega_i}, \pi_{N_i}, t, T)$

1: $a^* \leftarrow \text{argmax}_{a_i} V^t_{a_i}(B^t)$, $\pi_i(\boldsymbol{\omega_i}) \leftarrow a^*$
2: **if** $t < T - 1$ **then**
3: **for all** $\omega_i \in \Omega_i$ **do**
4: $B^{t+1} \leftarrow$ UPDATE$(i, B^t, a^*, \omega_i, \pi_{N_i})$
5: FINDPOLICY$(i, B^{t+1}, \langle \boldsymbol{\omega_i}, \omega_i \rangle, \pi_{N_i}, t+1, T)$
6: **return**

neighborhood utility $V_\pi[N_i]$. From Equation 2, increasing $V_\pi[N_i]$ results in an increase in global utility. By locality of interaction, if an agent $j \notin i \cup N_i$ changes its policy to improve its local neighborhood utility, it will not affect $V_\pi[N_i]$ but will increase global utility. Thus with each cycle global utility is strictly increasing until local optimum is reached. □

Proposition 4. *LID-JESP will terminate within d (= diameter) cycles iff agent are in a local optimum.*

Proof: Assume that in cycle c, agent i terminates ($terminationCtr_i = d$) but agents are not in a local optimum. In cycle $c - d$, there must be at least one agent j who can improve, i.e., $gain_j > 0$ (otherwise, agents are in a local optimum in cycle $c - d$ and no agent can improve later). Let d_{ij} refer to the shortest path distance between agents i and j. Then, in cycle $c - d + d_{ij}$ ($\leq c$), $terminationCtr_i$ must have been set to 0. However, $terminationCtr_i$ increases by at most one in each cycle. Thus, in cycle c, $terminationCtr_i \leq d - d_{ij}$. If $d_{ij} \geq 1$, in cycle c, $terminationCtr_i < d$. Also, if $d_{ij} = 0$, i.e., in cycle $c - d$, $gain_i > 0$, then in cycle $c - d + 1$, $terminationCtr_i = 0$, thus, in cycle c, $terminationCtr_i < d$. In either case, $terminationCtr_i \neq d$. By contradiction, if LID-JESP terminates then agents must be in a local optimum.

In the reverse direction, if agents reach a local optimum, $gain_i = 0$ henceforth. Thus, $terminationCtr_i$ is never reset to 0 and is incremented by 1 in every cycle. Hence, after d cycles, $terminationCtr_i = d$ and agents terminate. □

Proposition 3 shows that the agents will eventually reach a local optimum and Proposition 4 shows that the LID-JESP will terminate if and only if agents are in a local optimum. Thus, LID-JESP will correctly find a locally optimum and will terminate.

5 Global Optimal Algorithm (GOA)

The global optimal algorithm (GOA) exploits network structure in finding the optimal policy for a distributed POMDP. Unlike LID-JESP, at present it requires binary interactions, i.e. edges linking two nodes. We start with a description of GOA applied to tree-structured interaction graphs, and then discuss its application to graphs with cycles. In trees, value for a policy at an agent is the sum of best response values from its children and the joint policy reward associated with the parent policy. Thus, given a fixed policy for a parent node, GOA requires an agent to iterate through all its policies, finding the best policy and returning the value to the parent — where to find the best

policy, an agent requires its children to return their best responses to each of its policies. An agent also stores the sum of best response values from its children, to avoid recalculation at the children. This process is repeated at each level in the tree, until the root exhausts all its policies. This method helps GOA take advantage of the interaction graph and prune unnecessary joint policy evaluations (associated with nodes not connected directly in the tree). Since the interaction graph captures all the reward interactions among agents and as this algorithm goes through all the joint policy evaluations possible with the interaction graph, this algorithm yields an optimal solution.

Algorithm 7 provides the pseudo code for the global optimal algorithm at each agent. This algorithm is invoked with the procedure call GO-JOINTPOLICY($root, \langle\rangle, no$). Line 8 iterates through all the possible policies, where as lines 20-21 work towards calculating the best policy over this entire set of policies using the value of the policies calculated in Lines 9-19. Line 21 stores the values of best response policies obtained from the children. Lines 22-24 starts the termination of the algorithm after all the policies are exhausted at the root. Lines 1-4 propagate the termination message to lower levels in the tree, while recording the best policy, π_i^*.

Algorithm 7 GO-JOINTPOLICY($i, \pi_j, terminate$)

1: **if** $terminate =$ yes **then**
2: $\quad \pi_i^* \leftarrow bestResponse\{\pi_j\}$
3: $\quad$ **for all** $k \in children_i$ **do**
4: $\quad\quad$ GO-JOINTPOLICY(k, π_i^*,yes)
5: $\quad$ **return**
6: $\Pi_i \leftarrow$ enumerate all possible policies
7: $bestPolicyVal \leftarrow$ -∞,$j \leftarrow$ parent(i)
8: **for all** $\pi_i \in \Pi_i$ **do**
9: $\quad jointPolicyVal \leftarrow 0, childVal \leftarrow 0$
10: $\quad$ **if** $i \neq$ root **then**
11: $\quad\quad$ **for all** s_i, s_j, s_u **do**
12: $\quad\quad\quad jointPolicyVal \overset{+}{\leftarrow} b_i(s_i) \cdot b_{N_i}(s_{N_i}) \cdot b_u(s_u) \cdot$ EVALUATE($i, s_i, s_u, s_j, \pi_i, \pi_j, \langle\rangle, \langle\rangle, 0, T$)
13: $\quad$ **if** $bestChildValMap\{\pi_i\} \neq$ null **then**
14: $\quad\quad jointPolicyVal \overset{+}{\leftarrow} bestChildValMap\{\pi_i\}$
15: $\quad$ **else**
16: $\quad\quad$ **for all** $k \in children_i$ **do**
17: $\quad\quad\quad childVal \overset{+}{\leftarrow}$ GO-JOINTPOLICY(k, π_i,no)
18: $\quad\quad bestChildValMap\{\pi_i\} \leftarrow childVal$
19: $\quad\quad jointPolicyVal \overset{+}{\leftarrow} childVal$
20: $\quad$ **if** $jointPolicyVal > bestPolicyVal$ **then**
21: $\quad\quad bestPolicyVal \leftarrow jointPolicyVal, \pi_i^* \leftarrow \pi_i$
22: **if** $i = root$ **then**
23: $\quad$ **for all** $k \in children_i$ **do**
24: $\quad\quad$ GO-JOINTPOLICY(k, π_i^*,yes)
25: **if** $i \neq root$ **then** $bestResponse\{\pi_j\} = \pi_i^*$
26: **return** $bestPolicyVal$

By using cycle-cutset algorithms [10], GOA can be applied to interaction graphs containing cycles. These algorithms are used to identify a cycle-cutset, i.e., a subset of agents, whose deletion makes the remaining interaction graph acyclic. After identifying the cutset, joint policies for the cutset agents are enumerated, and then for each of them, we find the best policies of remaining agents using GOA.

6 Experimental Results

For our experiments, we use the sensor domain in Figure 1. We consider three different configurations of increasing complexity. The first configuration is a chain with 3 agents (sensors 1-3). Here target1 is either absent or in Loc1-1 and target2 is either absent or in Loc2-1 (4 unaffectable states). Each agent can perform either turnOff, scanEast or scanWest. Agents receive an observation, targetPresent or targetAbsent, based on the unaffectable state and its last action. The second configuration is a 4 agent chain (sensors 1-4). Here, target2 has an additional possible location, Loc2-2, giving rise to 6 unaffectable states. The number of individual actions and observations are unchanged. The 3rd configuration is the 5 agent P-configuration (named for the P shape of the sensor net) and is identical to Figure 1. Here, target1 can have two additional locations, Loc1-2 and Loc1-3, giving rise to 12 unaffectable states. We add a new action called scanVert for each agent to scan North and South. For each of these scenarios, we ran the LID-JESP algorithm. Our first benchmark, JESP, uses a centralized policy generator to find a locally optimal joint policy and does not consider the network structure of the interaction, while our second benchmark (LID-JESP-no-nw) is LID-JESP with a fully connected *interaction graph*. For 3 and 4 agent chains, we also ran the GOA algorithm.

Figure 2 compares the performance of the various algorithms for 3 and 4 agent chains and 5 agent P-configuration. Graphs (a), (b), (c) show the run time [6] in seconds on a logscale on Y-axis for increasing finite horizon T on X-axis. Run times for LID-JESP, JESP and LID-JESP-no-nw are averaged over 5 runs, each run with a different randomly chosen starting policy . For a particular run, all algorithms use the same starting policies. All three locally optimal algorithms show significant improvement over GOA in terms of run time with LID-JESP outperforming LID-JESP-no-nw and JESP by an order of magnitude (for high T) by exploiting *locality of interaction*. In graph (d), the values obtained using GOA for 3 and 4-Agent case ($T = 3$) are compared to the ones obtained using LID-JESP over 5 runs (each with a different starting policy) for $T = 3$. In this bar graph, the first bar represents value obtained using GOA, while other bars correspond to LID-JESP. This graph emphasizes the fact that with random restarts, LID-JESP converges to a higher local optima — such restarts are afforded given that GOA is orders of magnitude slower compared to LID-JESP.

Table 1 helps to better explain the reasons for the speed up of LID-JESP over JESP and LID-JESP-no-nw. LID-JESP allows more than one (non-neighboring) agent to change its policy within a cycle (W), LID-JESP-no-nw allows exactly one agent to change its policy in a cycle and in JESP, there are several cycles where no agent changes its policy. This allows LID-JESP to converge in fewer cycles (C) than LID-JESP-no-nw.

[6] Machine specs for all experiments: Intel Xeon 2.8 GHz processor, 2GB RAM, Linux Redhat 8.1

Although LID-JESP takes fewer cycles than JESP to converge, it required more calls to GETVALUE (G). However, each such call is cheaper owing to the locality of interaction. LID-JESP will out-perform JESP even more on multi-processor machines owing to its distributedness.

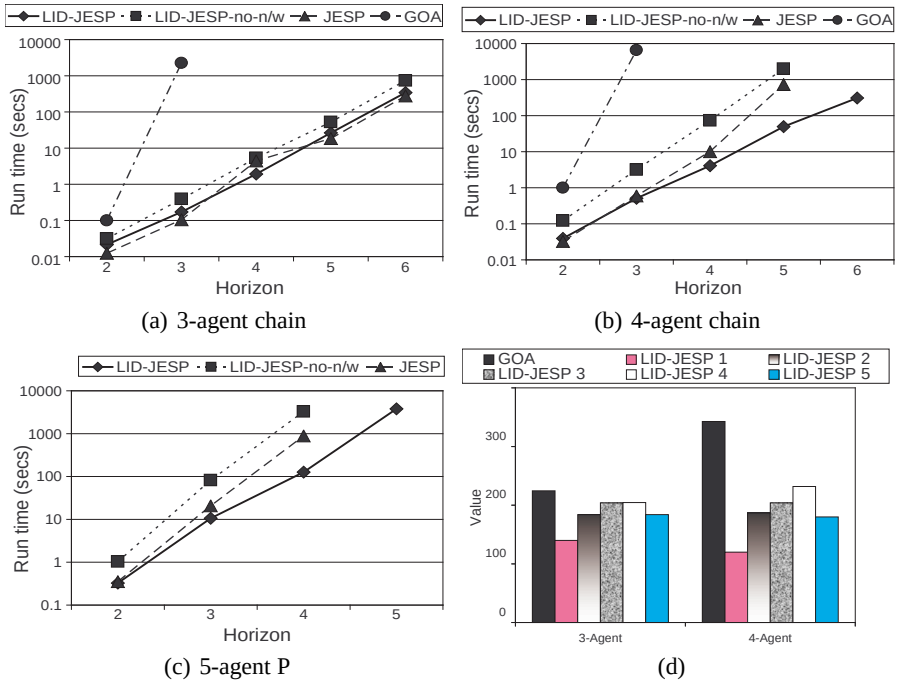

Fig. 2. Run times (a, b, c), and value (d).

7 Summary and Related Work

In a large class of applications, such as distributed sensor nets, distributed UAVs and satellites, a large network of agents is formed from each agent's limited interactions with a small number of neighboring agents. We exploit such network structure to present a new distributed POMDP model called ND-POMDP. Our distributed algorithms for ND-POMDPs exploit such network structure: the LID-JESP local search algorithm and GOA that is guaranteed to reach global optimal. Experimental results illustrate the significant run time gains of the two algorithms when compared with previous algorithms that are unable to exploit such structure.

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

Config.	Algorithm	C	G	W
4-chain	LID-JESP	3.4	13.6	1.412
	LID-JESP-no-nw	4.8	19.2	1
	JESP	7.8	7.8	0.436
5-P	LID-JESP	4.2	21	1.19
	LID-JESP-no-nw	5.8	29	1
	JESP	10.6	10.6	0.472

Table 1. Reasons for speed up. C: no. of cycles, G: no. of GETVALUE calls, W: no. of winners per cycle, for T=2.

globally optimal policy generation for distributed POMDPs. For instance, Hansen *et al.* [11] present an exact algorithm for partially observable stochastic games (POSGs) based on dynamic programming and iterated elimination of dominant policies. Emery-Montemerlo *et al.* [2] 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.* [1]'s JESP algorithm that uses dynamic programming to reach a local optimal. In addition, Becker *et al.*'s work [3] 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.* [12], 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 [13] 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.

8 Acknowledgments

This material is based upon work supported by DARPA/IPTO 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

1. Nair, R., Pynadath, D., Yokoo, M., Tambe, M., Marsella, S.: Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: IJCAI. (2003)
2. Montemerlo, R.E., Gordon, G., Schneider, J., Thrun, S.: Approximate solutions for partially observable stochastic games with common payoffs. In: AAMAS. (2004)

3. Becker, R., Zilberstein, S., Lesser, V., Goldman, C.: Solving transition independent decentralized Markov decision processes. JAIR **22** (2004) 423–455
4. Bernstein, D.S., Zilberstein, S., Immerman, N.: The complexity of decentralized control of MDPs. In: UAI. (2000)
5. Peshkin, L., Meuleau, N., Kim, K.E., Kaelbling, L.: Learning to cooperate via policy search. In: UAI. (2000)
6. Lesser, V., Ortiz, C., Tambe, M.: Distributed sensor nets: A multiagent perspective. Kluwer (2003)
7. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: An asynchronous complete method for distributed constraint optimization. In: AAMAS. (2003)
8. Yokoo, M., Hirayama, K.: Distributed breakout algorithm for solving distributed constraint satisfaction problems. In: ICMAS. (1996)
9. Goldman, C., Zilberstein, S.: Decentralized control of cooperative systems: Categorization and complexity analysis. JAIR **22** (2004) 143–174
10. Dechter, R.: Constraint Processing. Morgan Kaufman (2003)
11. Hansen, E., Bernstein, D., Zilberstein, S.: Dynamic Programming for Partially Observable Stochastic Games. In: AAAI. (2004)
12. Guestrin, C., Venkataraman, S., Koller, D.: Context specific multiagent coordination and planning with factored MDPs. In: AAAI. (2002)
13. Dolgov, D., Durfee, E.: Graphical models in local, asymmetric multi-agent markov decision processes. In: AAMAS. (2004)