Networked Distributed POMDPs: A Synergy of Distributed Constraint Optimization and POMDPs

Ranjit Nair Knowledge Services Group, Honeywell Laboratories, ranjit.nair@honeywell.com

Pradeep Varakantham, Milind Tambe Computer Science Dept., University of Southern California, {varakant,tambe}@usc.edu

Makoto Yokoo Dept. of Intelligent Systems, Kyushu University, 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 a distributed policy generation algorithm that performs local search.

1 Introduction

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. For instance, in distributed sensor nets, multiple sensor agents must coordinate with their neighboring agents in order to track individual targets moving through an area. In particular, we consider in this paper a problem motivated by the real-world challenge in [Lesser *et al.*, 2003]. Here, each sensor node can scan in one of four directions — North, South, East or West (see Figure 1), and to track a target, 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 but unaffected by the actions of the sensor agents. Additionally, each sensor receives observations only from the area it is scanning and this observation can have both false positives and false negatives. Each agent pays a cost for scanning whether the target is present or not but no cost if turned off. Existing distributed POMDP algorithms [Montemerlo *et al.*, 2004; Nair *et al.*, 2003; Becker *et al.*, 2004; Hansen *et al.*, 2004] although rich enough to capture the uncertainties in this domain, are unlikely to work well for such a domain because they are not geared to take advantage of the locality of interaction. As a result 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 these approaches cannot capture the uncertainty in the domain. Hence, we introduce the networked distributed POMDP (ND-POMDP) model, a hybrid of POMDP and DCOP, that can handle the uncertainties in the domain as well as take advantage of locality of interaction. Exploiting network structure enables us to present a novel algorithm for ND-POMDPs – a distributed policy generation algorithm that performs local search.

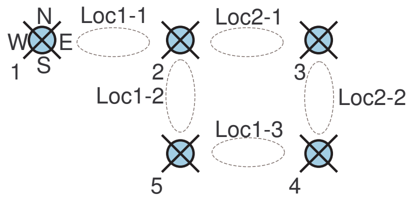


Figure 1: Sensor net scenario: If present, target1 is in Loc1-1, Loc1-2 or Loc1-3, and target2 is in Loc2-1 or Loc2-2.

2 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.* [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 $O_i(s_i, s_u, a_i, \omega_i) = \Pr(\omega_i|s_1, s_u, a_i)$.

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 $N_i = \{j \in Ag|j \ne i \wedge (\exists l \in E,\ i \in l \wedge j \in l)\}$. $S_{N_i} = \times_{j\in N_i} S_j$ 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 $b(s) = b_u(s_u)\cdot\prod_{1\le i\le n} b_i(s_i)$ 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 [Yokoo and Hirayama, 1996], an algorithm for distributed constraint satisfaction, to develop an algorithm for solving ND-POMDPs.

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

$$V_\pi[N_i] = \sum_{s_i, s_{N_i}, s_u} b_u(s_u) b_{N_i}(s_{N_i}) b_i(s_i) \sum_{l\in E\ s.t.\ i\in l} V^0_{\pi_l}(s_{l1}, \ldots, s_{lk}, s_u, \langle\rangle) \tag{1}$$

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 the following section.

3 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 [Yokoo and 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. It then evaluates its contribution to the global value function, from its initial belief state b with respect to its current policy and its neighbors' policy (function EVALUATE()). Agent i then tries to improve upon its current policy by calling function GETVALUE, which returns the value 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 its neighbors. If i's gain is greater than that one any of its neighbors, i changes its policy and sends its new policy to all its neighbors. This process of trying to improve the local policy is continued until termination, which is based on maintaining and exchanging a counter that counts the number of cycles where $gain_i = 0$. This is omitted from this article in order to simplify the presentation.

The algorithm 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 $e^t_i = \langle s^t_u, s^t_i, s^t_{N_i}, \vec{\omega}^t_{N_i}\rangle$. Given that the neighbors' policies are fixed, treating episode as the state, results in a single agent POMDP, where the transition function and observation function can be defined as follows:

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

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

The function GETVALUE() returns the optimal policy for agent i given the above definitions of transition and observation functions and the policies of N_i. In this paper, GETVALUE() was implemented via value iteration but any other single agent POMDP algorithm could have been used.

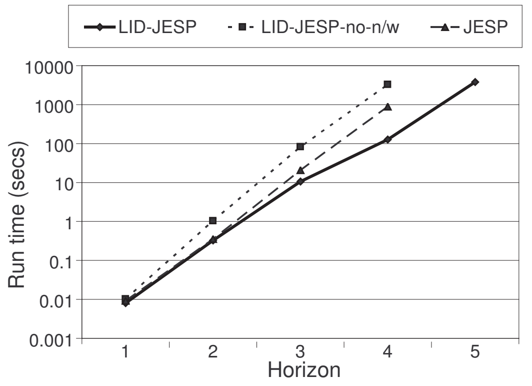
4 Experimental Results

For our experiments, we ran the LID-JESP algorithm on the sensor domain (see Figure 1). The first benchmark

Algorithm 1 LID-JESP($Agent\ i$)

```
1: $\pi_i \leftarrow$ randomly selected policy, $prevVal \leftarrow 0$
2: Exchange $\pi_i$ with $N_i$
3: while termination not detected do
4:   for all $s_i, s_{N_i}, s_u$ do
5:     $prevVal \overset{+}{\leftarrow} b_u(s_u) \cdot b_i(s_i) \cdot b_{N_i}(s_{N_i}) \cdot$ $\text{EVALUATE}(Agent\ i, s_i, s_u, s_{N_i}, \pi_i, \pi_{N_i}, \langle\rangle, \langle\rangle, 0, T)$
6:   $gain_i \leftarrow \text{GETVALUE}(Agent\ i, b, \pi_{N_i}, 0, T) - prevVal$
7:   Exchange $gain_i$ with $N_i$
8:   $maxGain \leftarrow \mathbf{max}_{j \in N_i \cup \{i\}} gain_j$
9:   $winner \leftarrow \mathbf{argmax}_{j \in N_i \cup \{i\}} gain_j$
10:  if $maxGain > 0$ and $i = winner$ then
11:    initialize $\pi_i$
12:    $\text{FINDPOLICY}(Agent\ i, b, \langle\rangle, \pi_{N_i}, 0, T)$
13:    Communicate $\pi_i$ with $N_i$
14:  else if $maxGain > 0$ then
15:    Receive $\pi_{winner}$ from $winner$ and update $\pi_{N_i}$
16: return $\pi_i$
```

(JESP) is Nair *et al.*'s JESP algorithm [2003], which uses a centralized processor to find a locally optimal joint policy and does not consider the *interaction graph*. The second benchmark (LID-JESP-no-nw) is LID-JESP with a fully connected *interaction graph*. Figure 2(a) shows the run time in seconds on a logscale on the Y-axis for increasing finite horizon T on the X-axis, while Figure 2(b) shows the value of policy found on the Y-axis and increasing finite horizon T on the X-axis. All run times and values were obtained by averaging 5 runs, each with different randomly chosen starting policies . As seen in Figure 2(b), the values obtained for LID-JESP, JESP and LID-JESP-no-nw are quite similar, although LID-JESP and LID-JESP-no-nw often converged on a higher local optima than JESP. In comparing the run times, it should be noted that LID-JESP outperforms LID-JESP-no-nw and JESP (which could not be run for $T > 4$ within 10000 secs).



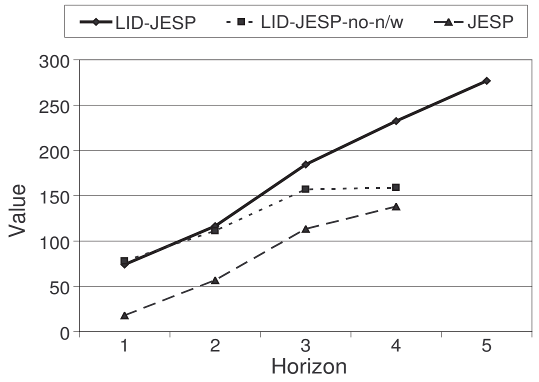


Figure 2: 5 agent F-config. (a) Run time (secs), (b) Value

5 Acknowledgments

This material is based upon work supported by the DARPA/IPTO COORDINATORs program and the Air Force Research Laboratoryunder 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 *et al.*, 2004] R. Becker, S. Zilberstein, V. Lesser, and C.V. Goldman. Solving transition independent decentralized Markov decision processes. *JAIR*, 22:423–455, 2004.

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

[Hansen *et al.*, 2004] E.A. Hansen, D.S. Bernstein, and S. Zilberstein. Dynamic Programming for Partially Observable Stochastic Games. In *AAAI*, 2004.

[Lesser *et al.*, 2003] V. Lesser, C. Ortiz, and M. Tambe. *Distributed sensor nets: A multiagent perspective.* Kluwer academic publishers, 2003.

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

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

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