Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies

Pradeep Varakantham, Janusz Marecki, Yuichi Yabu*, Milind Tambe, Makoto Yokoo*
University of Southern California, Los Angeles, CA 90089, {varakant, marecki, tambe}@usc.edu
* Dept. of Intelligent Systems, Kyushu University, Fukuoka, 812-8581 Japan, yokoo@is.kyushu-u.ac.jp

ABSTRACT

Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are a popular approach for modeling multi-agent systems acting in uncertain domains. Given the significant complexity of solving distributed POMDPs, particularly as we scale up the numbers of agents, one popular approach has focused on approximate solutions. Though this approach is efficient, the algorithms within this approach do not provide any guarantees on solution quality. A second less popular approach focuses on global optimality, but typical results are available only for two agents, and also at considerable computational cost. This paper overcomes the limitations of both these approaches by providing SPIDER, a novel combination of three key features for policy generation in distributed POMDPs: (i) it exploits agent interaction structure given a network of agents (i.e. allowing easier scale-up to larger number of agents); (ii) it uses a combination of heuristics to speedup policy search; and (iii) it allows quality guaranteed approximations, allowing a systematic tradeoff of solution quality for time. Experimental results show orders of magnitude improvement in performance when compared with previous global optimal algorithms.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence - Multi-agent Systems

General Terms

Algorithms, Theory

Keywords

Multi-agent systems, Partially Observable Markov Decision Process (POMDP), Distributed POMDP, Globally optimal solution

1. INTRODUCTION

Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are emerging as a popular approach for modeling sequential decision making in teams operating under uncertainty [9, 4, 1, 2, 13]. The uncertainty arises on account of nondeterminism in the outcomes of actions and because the world state may only be partially (or incorrectly) observable. Unfortunately, as shown by Bernstein *et al.* [3], the problem of finding the optimal joint policy for general distributed POMDPs is NEXP-Complete.

Researchers have attempted two different types of approaches towards solving these models. The first category consists of highly efficient approximate techniques, that may not reach globally optimal solutions [2, 9, 11]. The key problem with these techniques has been their inability to provide any guarantees on the quality of the solution. In contrast, the second less popular category of approaches has focused on a global optimal result [13, 5, 10]. Though these approaches obtain optimal solutions, they typically consider only two agents. Furthermore, they fail to exploit structure in the interactions of the agents and hence are severely hampered with respect to scalability when considering more than two agents.

To address these problems with the existing approaches, we propose approximate techniques that provide guarantees on the quality of the solution while focussing on a network of more than two agents. We first propose the basic SPIDER (Search for Policies In Distributed EnviRonments) algorithm. There are two key novel features in SPIDER: (i) it is a branch and bound heuristic search technique that uses a MDP-based heuristic function to search for an optimal joint policy; (ii) it exploits network structure of agents by organizing agents into a Depth First Search (DFS) pseudo tree and takes advantage of the independence in the different branches of the DFS tree. We then provide three enhancements to improve the efficiency of the basic SPIDER algorithm while providing guarantees on the quality of the solution. The first enhancement uses abstractions for speedup, but does not sacrifice solution quality. In particular, it initially performs branch and bound search on abstract policies and then extends to complete policies. The second enhancement obtains speedups by sacrificing solution quality, but within an input parameter that provides the tolerable expected value difference from the optimal solution. The third enhancement is again based on bounding the search for efficiency, however with a tolerance parameter that is provided as a percentage of optimal.

We experimented with the sensor network domain presented in Nair *et al.* [10], a domain representative of an important class of problems with networks of agents working in uncertain environments. In our experiments, we illustrate that SPIDER dominates an existing global optimal approach called GOA [10], the only known global optimal algorithm with demonstrated experimental results for more than two agents. Furthermore, we demonstrate that abstraction improves the performance of SPIDER significantly (while providing optimal solutions). We finally demonstrate a key feature of SPIDER: by utilizing the approximation enhancements it enables *principled* tradeoffs in run-time versus solution quality.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

2. DOMAIN: DISTRIBUTED SENSOR NETS

Distributed sensor networks are a large, important class of domains that motivate our work. This paper focuses on a set of target tracking problems that arise in certain types of sensor networks [6] first introduced in [10]. Figure 1 shows a specific problem instance within this type consisting of three sensors. Here, each sensor node can scan in one of four directions: North, South, East or West (see Figure 1). To track a target and obtain associated reward, two sensors with overlapping scanning areas must coordinate by scanning the same area simultaneously. In Figure 1, to track a target in Loc1-1, sensor1 needs to scan 'East' and sensor2 needs to scan 'West' simultaneously. Thus, sensors have to act in a coordinated fashion.

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. The sensors' observations and transitions are independent of each other's actions e.g.the observations that sensor1 receives are independent of sensor2's actions. Each agent incurs a cost for scanning whether the target is present or not, but no cost if it turns off. Given the sensors' observational uncertainty, the targets' uncertain transitions and the distributed nature of the sensor nodes, these sensor nets provide a useful domains for applying distributed POMDP models.

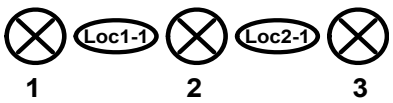

Figure 1: A 3-chain sensor configuration

3. BACKGROUND

3.1 Model: Network Distributed POMDP

The ND-POMDP model was introduced in [10], motivated by domains such as the sensor networks introduced in Section 2. It is defined as the 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.

ND-POMDP assumes *transition independence*, 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.

$\Omega = \times_{1\leq i\leq n}\Omega_i$ is the set of joint observations where Ω_i is the set of observations for agents i. *Observational independence* is assumed in ND-POMDPs i.e., the joint observation function is defined as $O(s, a, \omega) = \prod_{1\leq i\leq n} O_i(s_i, s_u, a_i, \omega_i)$, where $s = \langle s_1, \ldots, s_n, s_u\rangle$ is the world state that results from the agents performing $a = \langle a_1, \ldots, a_n\rangle$ in the previous state, and $\omega = \langle\omega_1, \ldots, \omega_n\rangle \in \Omega$ is the observation received in state s. This implies that each agent's observation depends only on the unaffectable state, its local action and on its resulting local state.

The reward function, R, is defined as $R(s, a) = \sum_l R_l(s_{l1}, \ldots, s_{lr}, s_u, \langle a_{l1}, \ldots, a_{lr}\rangle)$, where each l could refer to any sub-group of agents and $r = |l|$. Based on the reward function, an *interaction hypergraph* is constructed. A hyper-link, l, exists between a subset of agents for all R_l that comprise R. The *interaction hypergraph* is defined as $G = (Ag, E)$, where the agents, Ag, are the vertices and $E = \{l | l \subseteq Ag \wedge R_l \text{ is a component of } R\}$ are the edges.

The initial belief state (distribution over the initial state), b, is defined as $b(s) = b_u(s_u) \cdot \prod_{1\leq i\leq n} b_i(s_i)$, where b_u and b_i refer to the distribution over initial unaffectable state and agent i's initial belief state, respectively. The goal in ND-POMDP is to compute the joint policy $\pi = \langle\pi_1, \ldots, \pi_n\rangle$ that maximizes team's expected reward over a finite horizon T starting from the belief state b.

An ND-POMDP is similar to an *n-ary* Distributed Constraint Optimization Problem (DCOP)[8, 12] where the variable at each node represents the policy selected by an individual agent, π_i with the domain of the variable being the set of all local policies, Π_i. 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.

3.2 Algorithm: Global Optimal Algorithm (GOA)

In previous work, GOA has been defined as a global optimal algorithm for ND-POMDPs [10]. We will use GOA in our experimental comparisons, since GOA is a state-of-the-art global optimal algorithm, and in fact the only one with experimental results available for networks of more than two agents. GOA borrows from a global optimal DCOP algorithm called DPOP[12]. GOA's message passing follows that of DPOP. The first phase is the UTIL propagation, where the utility messages, in this case values of policies, are passed up from the leaves to the root. Value for a policy at an agent is defined as the sum of best response values from its children and the joint policy reward associated with the parent policy. Thus, given a policy for a parent node, GOA requires an agent to iterate through all its policies, finding the best response policy and returning the value to the parent — while at the parent node, to find the best policy, an agent requires its children to return their best responses to each of its policies. This UTIL propagation process is repeated at each level in the tree, until the root exhausts all its policies. In the second phase of VALUE propagation, where the optimal policies are passed down from the root till the leaves.

GOA takes advantage of the local interactions in the interaction graph, by pruning out 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 iterates through all the relevant joint policy evaluations, this algorithm yields a globally optimal solution.

4. SPIDER

As mentioned in Section 3.1, an ND-POMDP can be treated as a DCOP, where the goal is to compute a joint policy that maximizes the overall joint reward. The brute-force technique for computing an optimal policy would be to examine the expected values for all possible joint policies. The key idea in SPIDER is to avoid computation of expected values for the entire space of joint policies, by utilizing upper bounds on the expected values of policies and the interaction structure of the agents.

Akin to some of the algorithms for DCOP [8, 12], SPIDER has a pre-processing step that constructs a DFS tree corresponding to the given interaction structure. Note that these DFS trees are pseudo trees [12] that allow links between ancestors and children. We employ the Maximum Constrained Node (MCN) heuristic used in the DCOP algorithm, ADOPT [8], however other heuristics (such as MLSP heuristic from [7]) can also be employed. MCN heuristic tries to place agents with more number of constraints at the top of the tree. This tree governs how the search for the optimal joint pol-

icy proceeds in SPIDER. The algorithms presented in this paper are easily extendable to hyper-trees, however for expository purposes, we assume binary trees.

SPIDER is an algorithm for centralized planning and distributed execution in distributed POMDPs. In this paper, we employ the following notation to denote policies and expected values:
$Ancestors(i) \Rightarrow$ agents from i to the $root$ (not including i).
$Tree(i) \Rightarrow$ agents in the sub-tree (not including i) for which i is the root.
$\pi^{root+} \Rightarrow$ joint policy of all agents.
$\pi^{i+} \Rightarrow$ joint policy of all agents in $Tree(i) \cup i$.
$\pi^{i-} \Rightarrow$ joint policy of agents that are in $Ancestors(i)$.
$\pi_i \Rightarrow$ policy of the ith agent.
$\hat{v}[\pi_i, \pi^{i-}] \Rightarrow$ upper bound on the expected value for π^{i+} given π_i and policies of ancestor agents i.e. π^{i-}.
$\hat{v}_j[\pi_i, \pi^{i-}] \Rightarrow$ upper bound on the expected value for π^{i+} from the jth child.
$v[\pi_i, \pi^{i-}] \Rightarrow$ expected value for π_i given policies of ancestor agents, π^{i-}.
$v[\pi^{i+}, \pi^{i-}] \Rightarrow$ expected value for π^{i+} given policies of ancestor agents, π^{i-}.
$v_j[\pi^{i+}, \pi^{i-}] \Rightarrow$ expected value for π^{i+} from the jth child.

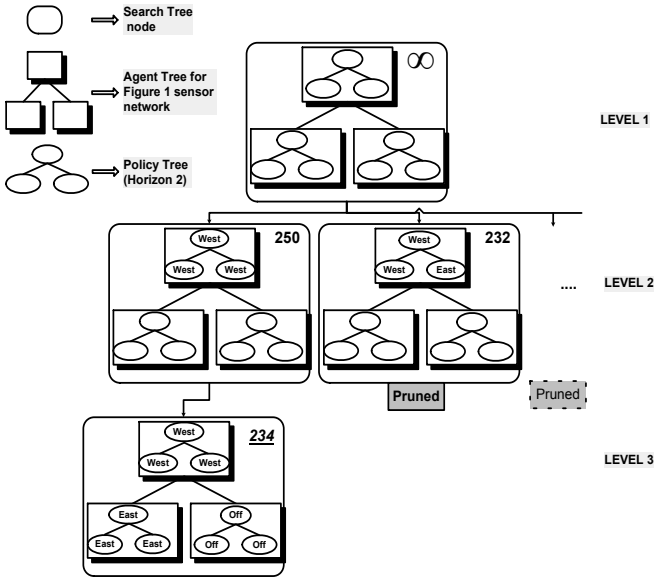

Figure 2: Execution of SPIDER, an example

4.1 Outline of SPIDER

SPIDER is based on the idea of branch and bound search, where the nodes in the search tree represent partial/complete joint policies. Figure 2 shows an example search tree for the SPIDER algorithm, using an example of the three agent chain. Before SPIDER begins its search we create a DFS tree (i.e. pseudo tree) from the three agent chain, with the middle agent as the root of this tree. SPIDER exploits the structure of this DFS tree while engaging in its search. Note that in our example figure, each agent is assigned a policy with T=2. Thus, each rounded rectange (search tree node) indicates a partial/complete joint policy, a rectangle indicates an agent and the ovals internal to an agent show its policy. Heuristic or actual expected value for a joint policy is indicated in the top right corner of the rounded rectangle. If the number is italicized and underlined, it implies that the actual expected value of the joint policy is provided.

SPIDER begins with no policy assigned to any of the agents (shown in the level 1 of the search tree). Level 2 of the search tree indicates that the joint policies are sorted based on upper bounds computed for root agent's policies. Level 3 shows one SPIDER search node with a complete joint policy (a policy assigned to each of the agents). The expected value for this joint policy is used to prune out the nodes in level 2 (the ones with upper bounds < 234)

When creating policies for each non-leaf agent i, SPIDER potentially performs two steps:

1. **Obtaining upper bounds and sorting**: In this step, agent i computes upper bounds on the expected values, $\hat{v}[\pi_i, \pi^{i-}]$ of the joint policies π^{i+} corresponding to each of its policy π_i and fixed ancestor policies. An MDP based heuristic is used to compute these upper bounds on the expected values. Detailed description about this MDP heuristic is provided in Section 4.2. All policies of agent i, Π_i are then sorted based on these upper bounds (also referred to as heuristic values henceforth) in descending order. Exploration of these policies (in step 2 below) are performed in this descending order. As indicated in the level 2 of the search tree (of Figure 2), all the joint policies are sorted based on the heuristic values, indicated in the top right corner of each joint policy. The intuition behind sorting and then exploring policies in descending order of upper bounds, is that the policies with higher upper bounds could yield joint policies with higher expected values.

2. **Exploration and Pruning**: Exploration implies computing the best response joint policy $\pi^{i+,*}$ corresponding to fixed ancestor policies of agent i, π^{i-}. This is performed by iterating through all policies of agent i i.e. Π_i and summing two quantities for each policy: (i) the best response for all of i's children (obtained by performing steps 1 and 2 at each of the child nodes); (ii) the expected value obtained by i for fixed policies of ancestors. Thus, exploration of a policy π_i yields actual expected value of a joint policy, π^{i+} represented as $v[\pi^{i+}, \pi^{i-}]$. The policy with the highest expected value is the best response policy.

Pruning refers to avoiding exploring all policies (or computing expected values) at agent i by using the current best expected value, $v^{max}[\pi^{i+}, \pi^{i-}]$. Henceforth, this $v^{max}[\pi^{i+}, \pi^{i-}]$ will be referred to as *threshold*. A policy, π_i need not be explored if the upper bound for that policy, $\hat{v}[\pi_i, \pi^{i-}]$ is less than the *threshold*. This is because the expected value for the best joint policy attainable for that policy will be less than the threshold.

On the other hand, when considering a leaf agent, SPIDER computes the best response policy (and consequently its expected value) corresponding to fixed policies of its ancestors, π^{i-}. This is accomplished by computing expected values for each of the policies (corresponding to fixed policies of ancestors) and selecting the highest expected value policy. In Figure 2, SPIDER assigns best response policies to leaf agents at level 3. The policy for the left leaf agent is to perform action "East" at each time step in the policy, while the policy for the right leaf agent is to perform "Off" at each time step. These best response policies from the leaf agents yield an actual expected value of 234 for the complete joint policy.

Algorithm 1 provides the pseudo code for SPIDER. This algorithm outputs the best joint policy, $\pi^{i+,*}$ (with an expected value greater than *threshold*) for the agents in $Tree(i)$. Lines 3-8 compute the best response policy of a leaf agent i, while lines 9-23 computes the best response joint policy for agents in $Tree(i)$. This best response computation for a non-leaf agent i includes: (a) Sorting of policies (in descending order) based on heuristic values on line 11; (b) Computing best response policies at each of the children for fixed policies of agent i in lines 16-20; and (c) Maintaining

Algorithm 1 SPIDER$(i, \pi^{i-}, threshold)$

1: $\pi^{i+,*} \leftarrow null$
2: $\Pi_i \leftarrow$ GET-ALL-POLICIES $(horizon, A_i, \Omega_i)$
3: **if** IS-LEAF(i) **then**
4: **for all** $\pi_i \in \Pi_i$ **do**
5: $v[\pi_i, \pi^{i-}] \leftarrow$ JOINT-REWARD (π_i, π^{i-})
6: **if** $v[\pi_i, \pi^{i-}] > threshold$ **then**
7: $\pi^{i+,*} \leftarrow \pi_i$
8: $threshold \leftarrow v[\pi_i, \pi^{i-}]$
9: **else**
10: $children \leftarrow$ CHILDREN (i)
11: $\hat{\Pi}_i \leftarrow$ UPPER-BOUND-SORT(i, Π_i, π^{i-})
12: **for all** $\pi_i \in \hat{\Pi}_i$ **do**
13: $\tilde{\pi}^{i+} \leftarrow \pi_i$
14: **if** $\hat{v}[\pi_i, \pi^{i-}] < threshold$ **then**
15: Go to line 12
16: **for all** $j \in children$ **do**
17: $jThres \leftarrow threshold - v[\pi_i, \pi^{i-}] - \Sigma_{k \in children, k \neq j} \hat{v}_k[\pi_i, \pi^{i-}]$
18: $\pi^{j+,*} \leftarrow$ SPIDER$(j, \pi_i \parallel \pi^{i-}, jThres)$
19: $\tilde{\pi}^{i+} \leftarrow \tilde{\pi}^{i+} \parallel \pi^{j+,*}$
20: $\hat{v}_j[\pi_i, \pi^{i-}] \leftarrow v[\pi^{j+,*}, \pi_i \parallel \pi^{i-}]$
21: **if** $v[\tilde{\pi}^{i+}, \pi^{i-}] > threshold$ **then**
22: $threshold \leftarrow v[\tilde{\pi}^{i+}, \pi^{i-}]$
23: $\pi^{i+,*} \leftarrow \tilde{\pi}^{i+}$
24: **return** $\pi^{i+,*}$

Algorithm 2 UPPER-BOUND-SORT(i, Π_i, π^{i-})

1: $children \leftarrow$ CHILDREN (i)
2: $\hat{\Pi}_i \leftarrow null$ /* Stores the sorted list */
3: **for all** $\pi_i \in \Pi_i$ **do**
4: $\hat{v}[\pi_i, \pi^{i-}] \leftarrow$ JOINT-REWARD (π_i, π^{i-})
5: **for all** $j \in children$ **do**
6: $\hat{v}_j[\pi_i, \pi^{i-}] \leftarrow$ UPPER-BOUND$(i, j, \pi_i \parallel \pi^{i-})$
7: $\hat{v}[\pi_i, \pi^{i-}] \stackrel{+}{\leftarrow} \hat{v}_j[\pi_i, \pi^{i-}]$
8: $\hat{\Pi}_i \leftarrow$ INSERT-INTO-SORTED $(\pi_i, \hat{\Pi}_i)$
9: **return** $\hat{\Pi}_i$

best expected value, joint policy in lines 21-23.

Algorithm 2 provides the pseudo code for sorting policies based on the upper bounds on the expected values of joint policies. Expected value for an agent i consists of two parts: value obtained from ancestors and value obtained from its children. Line 4 computes the expected value obtained from ancestors of the agent (using JOINT-REWARD function), while lines 5-7 compute the heuristic value from the children. The sum of these two parts yields an upper bound on the expected value for agent i, and line 8 of the algorithm sorts the policies based on these upper bounds.

4.2 MDP based heuristic function

The heuristic function quickly provides an upper bound on the expected value obtainable from the agents in $Tree(i)$. The sub-tree of agents is a distributed POMDP in itself and the idea here is to construct a centralized MDP corresponding to the (sub-tree) distributed POMDP and obtain the expected value of the optimal policy for this centralized MDP. To reiterate this in terms of the agents in DFS tree interaction structure, we assume full observability for the agents in $Tree(i)$ and for fixed policies of the agents in $\{Ancestors(i) \cup i\}$, we compute the joint value $\hat{v}[\pi^{i+}, \pi^{i-}]$.

We use the following notation for presenting the equations for computing upper bounds/heuristic values (for agents i and k):
Let E^{i-} denote the set of links between agents in $\{Ancestors(i) \cup i\}$ and $Tree(i)$, E^{i+} denote the set of links between agents in $Tree(i)$. Also, if $l \in E^{i-}$, then l_1 is the agent in $\{Ancestors(i) \cup i\}$ and l_2 is the agent in $Tree(i)$, that l connects together. We first compact the standard notation:

$$
\begin{aligned}
o_k^t &\stackrel{\triangle}{=} O_k(s_k^{t+1}, s_u^{t+1}, \pi_k(\vec{\omega}_k^t), \omega_k^{t+1}) \\
p_k^t &\stackrel{\triangle}{=} P_k(s_k^t, s_u^t, \pi_k(\vec{\omega}_k^t), s_k^{t+1}) \cdot o_k^t \\
p_u^t &\stackrel{\triangle}{=} P(s_u^t, s_u^{t+1}) \\
s_l^t &= \left\langle s_{l_1}^t, s_{l_2}^t, s_u^t \right\rangle ; \omega_l^t = \left\langle \omega_{l_1}^t, \omega_{l_2}^t \right\rangle \\
r_l^t &\stackrel{\triangle}{=} R_l(s_l^t, \pi_{l_1}(\vec{\omega}_{l_1}^t), \pi_{l_2}(\vec{\omega}_{l_2}^t)) \\
v_l^t &\stackrel{\triangle}{=} V_{\pi_l}^t(s_l^t, s_u^t, \vec{\omega}_{l_1}^t, \vec{\omega}_{l_2}^t)
\end{aligned}
\tag{1}
$$

Depending on the location of agent k in the agent tree we have the following cases:

$$
\begin{aligned}
&\text{IF } k \in \{Ancestors(i) \cup i\}, \hat{p}_k^t \stackrel{\triangle}{=} p_k^t, \\
&\text{IF } k \in Tree(i), \hat{p}_k^t \stackrel{\triangle}{=} P_k(s_k^t, s_u^t, \pi_k(\vec{\omega}_k^t), s_k^{t+1}) \\
&\text{IF } l \in E^{i-}, \hat{r}_l^t \stackrel{\triangle}{=} \max_{\{a_{l_2}\}} R_l(s_l^t, \pi_{l_1}(\vec{\omega}_{l_1}^t), a_{l_2}) \\
&\text{IF } l \in E^{i+}, \hat{r}_l^t \stackrel{\triangle}{=} \max_{\{a_{l_1}, a_{l_2}\}} R_l(s_l^t, a_{l_1}, a_{l_2})
\end{aligned}
\tag{2}
$$

The value function for an agent i executing the joint policy π^{i+} at time $\eta - 1$ is provided by the equation:

$$
\begin{aligned}
V_{\pi^{i+}}^{\eta-1}(s^{\eta-1}, \vec{\omega}^{\eta-1}) &= \sum_{l \in E^{i-}} v_l^{\eta-1} + \sum_{l \in E^{i+}} v_l^{\eta-1} \\
\text{where } v_l^{\eta-1} &= r_l^{\eta-1} + \sum_{\omega_l^\eta, s^\eta} p_{l_1}^{\eta-1} p_{l_2}^{\eta-1} p_u^{\eta-1} v_l^\eta
\end{aligned}
\tag{3}
$$

Algorithm 3 UPPER-BOUND (i, j, π^{j-})

1: $val \leftarrow 0$
2: **for all** $l \in E^{j-} \cup E^{j+}$ **do**
3: **if** $l \in E^{j-}$ **then** $\pi_{l_1} \leftarrow \phi$
4: **for all** s_l^0 **do**
5: $val \stackrel{+}{\leftarrow} startBel[s_l^0] \cdot$ UPPER-BOUND-TIME $(i, s_l^0, j, \pi_{l_1}, \langle\rangle)$
6: **return** val

Algorithm 4 UPPER-BOUND-TIME $(i, s_l^t, j, \pi_{l_1}, \vec{\omega}_{l_1}^t)$

1: $maxVal \leftarrow -\infty$
2: **for all** a_{l_1}, a_{l_2} **do**
3: **if** $l \in E^{i-}$ and $l \in E^{j-}$ **then** $a_{l_1} \leftarrow \pi_{l_1}(\vec{\omega}_{l_1}^t)$
4: $val \leftarrow$ GET-REWARD$(s_l^t, a_{l_1}, a_{l_2})$
5: **if** $t < \pi_i.horizon - 1$ **then**
6: **for all** $s_l^{t+1}, \omega_{l_1}^{t+1}$ **do**
7: $futVal \leftarrow p_u^t \hat{p}_{l_1}^t \hat{p}_{l_2}^t$
8: $futVal \stackrel{*}{\leftarrow}$ UPPER-BOUND-TIME$(s_l^{t+1}, j, \pi_{l_1}, \vec{\omega}_{l_1}^t \parallel \omega_{l_1}^{t+1})$
9: $val \stackrel{+}{\leftarrow} futVal$
10: **if** $val > maxVal$ **then** $maxVal \leftarrow val$
11: **return** $maxVal$

Upper bound on the expected value for a link is computed by modifying the equation 3 to reflect the full observability assumption. This involves removing the observational probability term for agents in $Tree(i)$ and maximizing the future value $\hat{v}_l^\eta$ over the actions of those agents (in $Tree(i)$). Thus, the equation for the

computation of the upper bound on a link l, is as follows:

$$\text{IF } l \in E^{i-}, \hat{v}_l^{\eta-1} = \hat{r}_l^{\eta-1} + \max_{a_{l_2}} \sum_{\omega_{l_1}^{\eta}, s_l^{\eta}} \hat{p}_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} p_u^{\eta-1} \hat{v}_l^{\eta}$$

$$\text{IF } l \in E^{i+}, \hat{v}_l^{\eta-1} = \hat{r}_l^{\eta-1} + \max_{a_{l_1}, a_{l_2}} \sum_{s_l^{\eta}} \hat{p}_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} p_u^{\eta-1} \hat{v}_l^{\eta}$$

Algorithm 3 and Algorithm 4 provide the algorithm for computing upper bound for child j of agent i, using the equations descirbed above. While Algorithm 4 computes the upper bound on a link given the starting state, Algorithm 3 sums the upper bound values computed over each of the links in $E^{i-} \cup E^{i+}$.

4.3 Abstraction

Algorithm 5 SPIDER-ABS$(i, \pi^{i-}, threshold)$

```
1: π^{i+,*} ← null
2: Π_i ← GET-POLICIES (<>, 1)
3: if IS-LEAF(i) then
4:    for all π_i ∈ Π_i do
5:       absHeuristic ← GET-ABS-HEURISTIC (π_i, π^{i-})
6:       absHeuristic ←* (timeHorizon − π_i.horizon)
7:       if π_i.horizon = timeHorizon and π_i.absNodes = 0 then
8:          v[π_i, π^{i-}] ← JOINT-REWARD (π_i, π^{i-})
9:          if v[π_i, π^{i-}] > threshold then
10:            π^{i+,*} ← π_i; threshold ← v[π_i, π^{i-}]
11:      else if v[π_i, π^{i-}] + absHeuristic > threshold then
12:         Π̂_i ← EXTEND-POLICY (π_i, π_i.absNodes + 1)
13:         Π_i ←+ INSERT-SORTED-POLICIES (Π̂_i)
14:      REMOVE(π_i)
15: else
16:   children ← CHILDREN (i)
17:   Π_i ← UPPER-BOUND-SORT(i, Π_i, π^{i-})
18:   for all π_i ∈ Π_i do
19:      π̃^{i+} ← π_i
20:      absHeuristic ← GET-ABS-HEURISTIC (π_i, π^{i-})
21:      absHeuristic ←* (timeHorizon − π_i.horizon)
22:      if π_i.horizon = timeHorizon and π_i.absNodes = 0 then
23:         if v̂[π_i, π^{i-}] < threshold and π_i.absNodes = 0 then
24:            Go to line 19
25:         for all j ∈ children do
26:            jThres ← threshold − v[π_i, π^{i-}] −
                        Σ_{k∈children, k≠j} v̂_k[π_i, π^{i-}]
27:            π^{j+,*} ← SPIDER(j, π_i || π^{i-}, jThres)
28:            π̃^{i+} ← π̃^{i+} || π^{j+,*}; v̂_j[π_i, π^{i-}] ← v[π^{j+,*}, π_i || π^{i-}]
29:         if v[π̃^{i+}, π^{i-}] > threshold then
30:            threshold ← v[π̃^{i+}, π^{i-}]; π^{i+,*} ← π̃^{i+}
31:      else if v̂[π^{i+}, π^{i-}] + absHeuristic > threshold then
32:         Π̂_i ← EXTEND-POLICY (π_i, π_i.absNodes + 1)
33:         Π_i ←+ INSERT-SORTED-POLICIES (Π̂_i)
34:   REMOVE(π_i)
35: return π^{i+,*}
```

In SPIDER, the exploration/pruning phase can only begin after the heuristic (or upper bound) computation and sorting for the policies has ended. We provide an approach to possibly circumvent the exploration of a group of policies based on heuristic computation for one abstract policy, thus leading to an improvement in runtime performance (without loss in solution quality). The important steps in this technique are defining the abstract policy and how heuristic values are computated for the abstract policies. In this paper, we propose two types of abstraction:

1. Horizon Based Abstraction (HBA): Here, the abstract policy is defined as a shorter horizon policy. It represents a group of longer horizon policies that have the same actions as the abstract policy for times less than or equal to the horizon of the abstract policy. In Figure 3(a), a T=1 abstract policy that performs "East" action, represents a group of T=2 policies, that perform "East" in the first time step.

For HBA, there are two parts to heuristic computation:
(a) Computing the upper bound for the horizon of the abstract policy. This is same as the heuristic computation defined by the GET-HEURISTIC() algorithm for SPIDER, however with a shorter time horizon (horizon of the abstract policy).
(b) Computing the maximum possible reward that can be accumulated in one time step (using GET-ABS-HEURISTIC()) and multiplying it by the number of time steps to time horizon. This maximum possible reward (for one time step) is obtained by iterating through all the actions of all the agents in $Tree(i)$ and computing the maximum joint reward for any joint action.
Sum of (a) and (b) is the heuristic value for a HBA abstract policy.

2. Node Based Abstraction (NBA): Here an abstract policy is obtained by not associating actions to certain nodes of the policy tree. Unlike in HBA, this implies multiple levels of abstraction. This is illustrated in Figure 3(b), where there are T=2 policies that do not have an action for observation 'TP'. These incomplete T=2 policies are abstractions for T=2 complete policies. Increased levels of abstraction leads to faster computation of a complete joint policy, π^{root+} and also to shorter heuristic computation and exploration, pruning phases. For NBA, the heuristic computation is similar to that of a normal policy, except in cases where there is no action associated with policy nodes. In such cases, the immediate reward is taken as R_{max} (maximum reward for any action).

We combine both the abstraction techniques mentioned above into one technique, SPIDER-ABS. Algorithm 5 provides the algorithm for this abstraction technique. For computing optimal joint policy with SPIDER-ABS, a non-leaf agent i initially examines all abstract T=1 policies (line 2) and sorts them based on abstract policy heuristic computations (line 17). The abstraction horizon is gradually increased and these abstract policies are then explored in descending order of heuristic values and ones that have heuristic values less than the *threshold* are pruned (lines 23-24). *Exploration* in SPIDER-ABS has the same definition as in SPIDER if the policy being explored has a horizon of policy computation which is equal to the actual time horizon and if all the nodes of the policy have an action associated with them (lines 25-30). However, if those conditions are not met, then it is substituted by a group of policies that it represents (using EXTEND-POLICY () function) (lines 31-32).

EXTEND-POLICY() function is also responsible for initializing the $horizon$ and $absNodes$ of a policy. $absNodes$ represents the number of nodes at the last level in the policy tree, that do not have an action assigned to them. If $\pi_i.absNodes = |\Omega_i|^{\pi_i.horizon-1}$ (i.e. total number of policy nodes possible at $\pi_i.horizon$) , then $\pi_i.absNodes$ is set to zero and $\pi_i.horizon$ is increased by 1. Otherwise, $\pi_i.absNodes$ is increased by 1. Thus, this function combines both HBA and NBA by using the policy variables, $horizon$ and $absNodes$. Before substituting the abstract policy with a group of policies, those policies are sorted based on heuristic values (line 33). Similar type of abstraction based best response computation is adopted at leaf agents (lines 3-14).

4.4 Value ApproXimation (VAX)

In this section, we present an approximate enhancement to SPIDER called VAX. The input to this technique is an approximation parameter ϵ, which determines the difference from the optimal solution quality. This approximation parameter is used at each agent for pruning out joint policies. The pruning mechanism in SPIDER and SPIDER-Abs dictates that a joint policy be pruned only if the threshold is exactly greater than the heuristic value. However, the

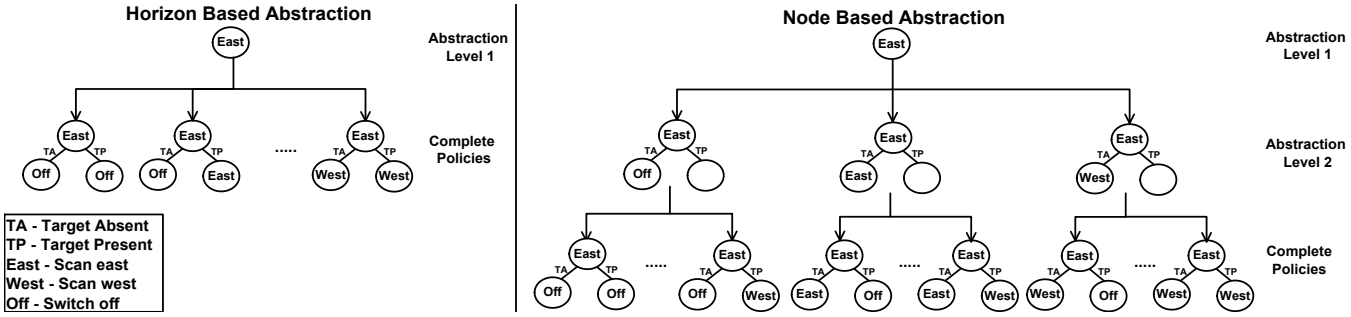

Figure 3: Example of abstraction for (a) HBA (Horizon Based Abstraction) and (b) NBA (Node Based Abstraction)

idea in this technique is to prune out joint a policy if the following condition is satisfied: $threshold + \epsilon > \hat{v}[\pi^i, \pi^{i-}]$. Apart from the pruning condition, VAX is the same as SPIDER/SPIDER-ABS.

In the example of Figure 2, if the heuristic value for the second joint policy (or second search tree node) in level 2 were 238 instead of 232, then that policy could not be be pruned using SPIDER or SPIDER-Abs. However, in VAX with an approximation parameter of 5, the joint policy in consideration would also be pruned. This is because the *threshold* (234) at that juncture plus the approximation parameter (5), i.e. 239 would have been greater than the heuristic value for that joint policy (238). It can be noted from the example (just discussed) that this kind of *pruning* can lead to fewer *explorations* and hence lead to an improvement in the overall run-time performance. However, this can entail a sacrifice in the quality of the solution because this technique can prune out a candidate optimal solution. A bound on the error introduced by this approximate algorithm as a function of ϵ, is provided by Proposition 3.

4.5 Percentage ApproXimation (PAX)

In this section, we present the second approximation enhancement over SPIDER called PAX. Input to this technique is a parameter, δ that represents the minimum percentage of the optimal solution quality that is desired. Output of this technique is a policy with an expected value that is at least δ% of the optimal solution quality. A policy is pruned if the following condition is satisfied: $threshold > \frac{\delta}{100}\hat{v}[\pi^i, \pi^{i-}]$. Like in VAX, the only difference between PAX and SPIDER/SPIDER-ABS is this pruning condition.

Again in Figure 2, if the heuristic value for the second search tree node in level 2 were 238 instead of 232, then PAX with an input parameter of 98% would be able to prune that search tree node (since $\frac{98}{100} * 238 < 234$). This type of pruning leads to fewer explorations and hence an improvement in run-time performance, while potentially leading to a loss in quality of the solution. Proposition 4 provides the bound on quality loss.

4.6 Theoretical Results

PROPOSITION 1. *Heuristic provided using the centralized MDP heuristic is admissible.*

Proof. For the value provided by the heuristic to be admissible, it should be an over estimate of the expected value for a joint policy. Thus, we need to show that: For $l \in E^{i+} \cup E^{i-}$: $\hat{v}_l^t \geq v_l^t$ (refer to notation in Section 4.2)

We use mathematical induction on t to prove this.

Base case: $t = T - 1$. Irrespective of whether $l \in E^{i-}$ or $l \in E^{i+}$, $\hat{r}_l^t$ is computed by maximizing over all actions of the agents in $Tree(i)$, while r_l^t is computed for fixed policies of the same agents. Hence, $\hat{r}_l^t \geq r_l^t$ and also $\hat{v}_l^t \geq v_l^t$.

Assumption: Proposition holds for $t = \eta$, where $1 \leq \eta < T - 1$.

We now have to prove that the proposition holds for $t = \eta - 1$.

We show the proof for $l \in E^{i-}$ and similar reasoning can be adopted to prove for $l \in E^{i+}$. The heuristic value function for $l \in E^{i-}$ is provided by the following equation:

$$\hat{v}_l^{\eta-1} = \hat{r}_l^{\eta-1} + \max_{a_{l_2}} \sum_{\omega_{l_1}^\eta, s_l^\eta} \hat{p}_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} p_u^{\eta-1} \hat{v}_l^\eta$$

Rewriting the RHS and using Eqn 2 (in Section 4.2)

$$= \hat{r}_l^{\eta-1} + \max_{a_{l_2}} \sum_{\omega_{l_1}^\eta, s_l^\eta} p_u^{\eta-1} p_{l_1}^{\eta-1} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^\eta$$

$$= \hat{r}_l^{\eta-1} + \sum_{\omega_{l_1}^\eta, s_l^\eta} p_u^{\eta-1} p_{l_1}^{\eta-1} \max_{a_{l_2}} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^\eta$$

Since $\max_{a_{l_2}} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^\eta \geq \sum_{\omega_{l_2}} o_{l_2}^{\eta-1} \hat{p}_{l_2}^{\eta-1} \hat{v}_l^\eta$ and $p_{l_2}^{\eta-1} = o_{l_2}^{\eta-1} \hat{p}_{l_2}^{\eta-1}$

$$\geq \hat{r}_l^{\eta-1} + \sum_{\omega_{l_1}^\eta, s_l^\eta} p_u^{\eta-1} p_{l_1}^{\eta-1} \sum_{\omega_{l_2}} p_{l_2}^{\eta-1} \hat{v}_l^\eta$$

Since $\hat{v}_l^\eta \geq v_l^\eta$ (from the assumption)

$$\geq \hat{r}_l^{\eta-1} + \sum_{\omega_{l_1}^\eta, s_l^\eta} p_u^{\eta-1} p_{l_1}^{\eta-1} \sum_{\omega_{l_2}} p_{l_2}^{\eta-1} v_l^\eta$$

Since $\hat{r}_l^{\eta-1} \geq r_l^{\eta-1}$ (by definition)

$$\geq r_l^{\eta-1} + \sum_{\omega_{l_1}^\eta, s_l^\eta} p_u^{\eta-1} p_{l_1}^{\eta-1} \sum_{\omega_{l_2}} p_{l_2}^{\eta-1} v_l^\eta$$

$$= r_l^{\eta-1} + \sum_{(\omega_l^\eta, s_l^\eta)} p_u^{\eta-1} p_{l_1}^{\eta-1} p_{l_2}^{\eta-1} v_l^\eta = v_l^{\eta-1}$$

Thus proved. ■

PROPOSITION 2. *SPIDER provides an optimal solution.*

Proof. SPIDER examines all possible joint policies given the interaction structure of the agents. The only exception being when a joint policy is *pruned* based on the heuristic value. Thus, as long as a candidate optimal policy is not pruned, SPIDER will return an optimal policy. As proved in Proposition 1, the expected value for a joint policy is always an upper bound. Hence when a joint policy is pruned, it cannot be an optimal solution.

PROPOSITION 3. *Error bound on the solution quality for VAX (implemented over SPIDER-ABS) with an approximation parameter of ϵ is $\rho\epsilon$, where ρ is the number of leaf nodes in the DFS tree.*

Proof. We prove this proposition using mathematical induction on the $depth$ of the DFS tree.

Base case: depth $= 1$ (i.e. one node). Best response is computed by iterating through all policies, Π_k. A policy,π_k is pruned if $\hat{v}[\pi_k, \pi^{k-}] < threshold + \epsilon$. Thus the best response policy computed by VAX would be at most ϵ away from the optimal best response. Hence the proposition holds for the base case.

Assumption: Proposition holds for d, where $1 \leq$ depth $\leq d$.

We now have to prove that the proposition holds for $d + 1$.

Without loss of generality, lets assume that the root node of this tree has k children. Each of this children is of depth $\leq d$, and hence from the assumption, the error introduced in kth child is $\rho_k \epsilon$, where ρ_k is the number of leaf nodes in kth child of the root. Therefore, $\rho = \sum_k \rho_k$, where ρ is the number of leaf nodes in the tree.

In SPIDER-ABS, $threshold$ at the root agent, $thres_{spider} = \sum_k v[\pi^{k+}, \pi^{k-}]$. However, with VAX the $threshold$ at the root agent will be (in the worst case), $thresh_{vax} = \sum_k v[\pi^{k+}, \pi^{k-}] - \sum_k \rho_k \epsilon$. Hence, with VAX a joint policy is pruned at the root agent if $\hat{v}[\pi_{root}, \pi^{root-}] < thresh_{vax} + \epsilon \Rightarrow \hat{v}[\pi_{root}, \pi^{root-}] < thresh_{spider} - ((\sum_k \rho_k) - 1)\epsilon \leq thresh_{spider} - (\sum_k \rho_k)\epsilon \leq thresh_{spider} - \rho\epsilon$. Hence proved. ■

PROPOSITION 4. *For PAX (implemented over SPIDER-ABS) with an input parameter of δ, the solution quality is at least $\frac{\delta}{100} v[\pi^{root+,*}]$, where $v[\pi^{root+,*}]$ denotes the optimal solution quality.*

Proof. We prove this proposition using mathematical induction on the $depth$ of the DFS tree.

Base case: depth $= 1$ (i.e. one node). Best response is computed by iterating through all policies, Π_k. A policy,π_k is pruned if $\frac{\delta}{100}\hat{v}[\pi_k, \pi^{k-}] < threshold$. Thus the best response policy computed by PAX would be at least $\frac{\delta}{100}$ times the optimal best response. Hence the proposition holds for the base case.

Assumption: Proposition holds for d, where $1 \leq$ depth $\leq d$.

We now have to prove that the proposition holds for $d + 1$.

Without loss of generality, lets assume that the root node of this tree has k children. Each of this children is of depth $\leq d$, and hence from the assumption, the solution quality in the kth child is at least $\frac{\delta}{100} v[\pi^{k+,*}, \pi^{k-}]$ for PAX. With SPIDER-ABS, a joint policy is pruned at the root agent if $\hat{v}[\pi_{root}, \pi^{root-}] < \sum_k v[\pi^{k+,*}, \pi^{k-}]$. However with PAX, a joint policy is pruned if $\frac{\delta}{100}\hat{v}[\pi_{root}, \pi^{root-}] < \sum_k \frac{\delta}{100} v[\pi^{k+,*}, \pi^{k-}] \Rightarrow \hat{v}[\pi_{root}, \pi^{root-}] < \sum_k v[\pi^{k+,*}, \pi^{k-}]$. Since the pruning condition at the root agent in PAX is the same as the one in SPIDER-ABS, there is no error introduced at the root agent and all the error is introduced in the children. Thus, overall solution quality is at least $\frac{\delta}{100}$ of the optimal solution. Hence proved.■

5. EXPERIMENTAL RESULTS

All our experiments were conducted on the sensor network domain from Section 2. The five network configurations employed are shown in Figure 4. Algorithms that we experimented with are GOA, SPIDER, SPIDER-ABS, PAX and VAX. We compare against GOA because it is the only global optimal algorithm that considers more than two agents. We performed two sets of experiments: (i) firstly, we compared the run-time performance of the above algorithms and (ii) secondly, we experimented with PAX and VAX to study the tradeoff between run-time and solution quality. Experiments were terminated after 10000 seconds[1].

Figure 5(a) provides run-time comparisons between the optimal algorithms GOA, SPIDER, SPIDER-Abs and the approximate algorithms, PAX (ϵ of 30) and VAX(δ of 80). X-axis denotes the sensor network configuration used, while Y-axis indicates the runtime (on a *log-scale*). The time horizon of policy computation was 3. For each configuration (3-chain, 4-chain, 4-star and 5-star), there are five bars indicating the time taken by GOA, SPIDER, SPIDER-Abs, PAX and VAX. GOA did not terminate within the time limit for 4-star and 5-star configurations. SPIDER-Abs dominated the SPIDER and GOA for all the configurations. For instance, in the 3-chain configuration, SPIDER-ABS provides 230-fold speedup over GOA and 2-fold speedup over SPIDER and for the 4-chain configuration it provides 58-fold speedup over GOA and 2-fold speedup over SPIDER. The two approximation approaches, VAX and PAX provided further improvement in performance over SPIDER-Abs. For instance, in the 5-star configuration VAX provides a 15-fold speedup and PAX provides a 8-fold speedup over SPIDER-Abs.

Figures 5(b) provides a comparison of the solution quality obtained using the different algorithms for the problems tested in Figure 5(a). X-axis denotes the sensor network configuration while Y-axis indicates the solution quality. Since GOA, SPIDER, and SPIDER-Abs are all global optimal algorithms, the solution quality is the same for all those algorithms. For 5-P configuration, the global optimal algorithms did not terminate within the limit of 10000 seconds, so the bar for optimal quality indicates an upper bound on the optimal solution quality. With both the approximations, we obtained a solution quality that was close to the optimal solution quality. In 3-chain and 4-star configurations, it is remarkable that both PAX and VAX obtained almost the same actual quality as the global optimal algorithms, despite the approximation parameter ϵ and δ. For other configurations as well, the loss in quality was less than 20% of the optimal solution quality.

Figure 5(c) provides the time to solution with PAX (for varying epsilons). X-axis denotes the approximation parameter, δ (percentage to optimal) used, while Y-axis denotes the time taken to compute the solution (on a *log-scale*). The time horizon for all the configurations was 4. As δ was decreased from 70 to 30, the time to solution decreased drastically. For instance, in the 3-chain case there was a total speedup of 170-fold when the δ was changed from 70 to 30. Interestingly, even with a low δ of 30%, the actual solution quality remained equal to the one obtained at 70%.

Figure 5(d) provides the time to solution for all the configurations with VAX (for varying epsilons). X-axis denotes the approximation parameter, ϵ used, while Y-axis denotes the time taken to compute the solution (on a *log-scale*). The time horizon for all the configurations was 4. As ϵ was increased, the time to solution decreased drastically. For instance, in the 4-star case there was a total speedup of 73-fold when the ϵ was changed from 60 to 140. Again, the actual solution quality did not change with varying epsilon.

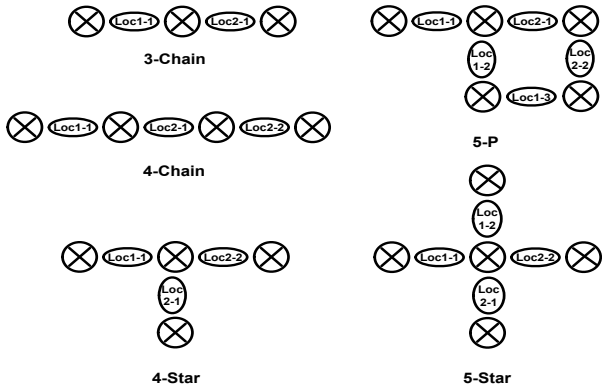

Figure 4: Sensor network configurations

[1]Machine specs for all experiments: Intel Xeon 3.6 GHZ processor, 2GB RAM

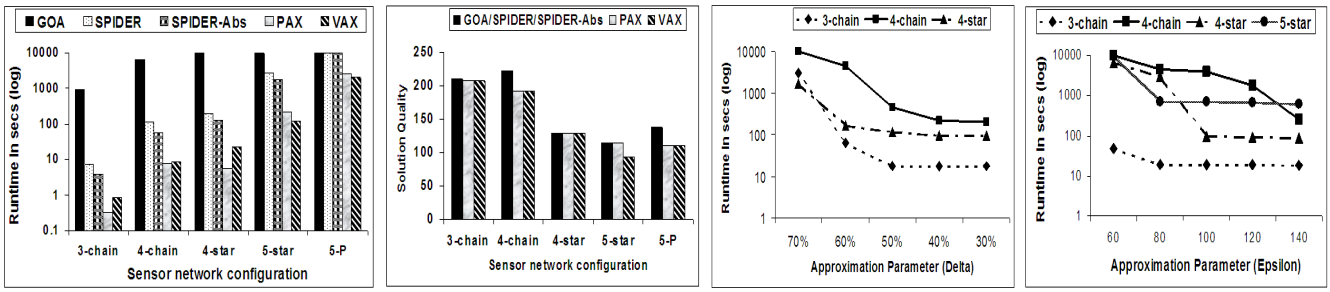

Figure 5: Comparison of GOA, SPIDER, SPIDER-Abs and VAX for T = 3 on (a) Runtime and (b) Solution quality; (c) Time to solution for PAX with varying percentage to optimal for T=4 (d) Time to solution for VAX with varying epsilon for T=4

6. SUMMARY AND RELATED WORK

This paper presents four algorithms SPIDER, SPIDER-ABS, PAX and VAX that provide a novel combination of features for policy search in distributed POMDPs: (i) exploiting agent interaction structure given a network of agents (i.e. easier scale-up to larger number of agents); (ii) using branch and bound search with an MDP based heuristic function; (iii) utilizing abstraction to improve runtime performance without sacrificing solution quality; (iv) providing *a priori* percentage bounds on quality of solutions using PAX; and (v) providing expected value bounds on the quality of solutions using VAX. These features allow for systematic tradeoff of solution quality for run-time in networks of agents operating under uncertainty. Experimental results show orders of magnitude improvement in performance over previous global optimal algorithms.

Researchers have typically employed two types of techniques for solving distributed POMDPs. The first set of techniques compute global optimal solutions. Hansen *et al.* [5] present an algorithm based on dynamic programming and iterated elimination of dominant policies, that provides optimal solutions for distributed POMDPs. Szer *et al.* [13] provide an optimal heuristic search method for solving Decentralized POMDPs. This algorithm is based on the combination of a classical heuristic search algorithm, A^* and decentralized control theory. The key differences between SPIDER and MAA* are: (a) Enhancements to SPIDER (VAX and PAX) provide for quality guaranteed approximations, while MAA* is a global optimal algorithm and hence involves significant computational complexity; (b) Due to MAA*'s inability to exploit interaction structure, it was illustrated only with two agents. However, SPIDER has been illustrated for networks of agents; and (c) SPIDER explores the joint policy one agent at a time, while MAA* expands it one time step at a time (simultaneously for all the agents).

The second set of techniques seek approximate policies. Emery-Montemerlo *et al.* [4] approximate POSGs as a series of one-step Bayesian games using heuristics to approximate future value, trading off limited lookahead for computational efficiency, resulting in locally optimal policies (with respect to the selected heuristic). Nair *et al.* [9]'s JESP algorithm uses dynamic programming to reach a local optimum solution for finite horizon decentralized POMDPs. Peshkin *et al.* [11] and Bernstein *et al.* [2] are examples of policy search techniques that search for locally optimal policies. Though all the above techniques improve the efficiency of policy computation considerably, they are unable to provide error bounds on the quality of the solution. This aspect of quality bounds differentiates SPIDER from all the above techniques.

Acknowledgements. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division under Contract No. NBCHD030010. 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.

7. REFERENCES

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

[2] D. S. Bernstein, E.A. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *IJCAI*, 2005.

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

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

[5] E. Hansen, D. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, 2004.

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

[7] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking dcop to the real world : Efficient complete solutions for distributed event scheduling. In *AAMAS*, 2004.

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

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

[10] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, 2005.

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

[12] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, 2005.

[13] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *IJCAI*, 2005.