Not All Agents Are Equal: Scaling up Distributed POMDPs for Agent Networks

Janusz Marecki*, Tapana Gupta*, Pradeep Varakantham+, Milind Tambe*, Makoto Yokoo++
*University of Southern California, Los Angeles, CA 90089, {marecki, tapanagu, tambe }@usc.edu
+Carnegie Mellon University, Pittsburgh, PA 15213, pradeepv@cs.cmu.edu
++Kyushu University, Fukuoka, 819-0395 Japan, yokoo@is.kyushu-u.ac.jp

ABSTRACT

Many applications of networks of agents, including mobile sensor networks, unmanned air vehicles, autonomous underwater vehicles, involve 100s of agents acting collaboratively under uncertainty. Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are well-suited to address such applications, but so far, only limited scale-ups of up to five agents have been demonstrated. This paper escalates the scale-up, presenting an algorithm called FANS, increasing the number of agents in distributed POMDPs for the first time into double digits. FANS is founded on finite state machines (FSMs) for policy representation and expoits these FSMs to provide three key contributions: (i) Not all agents within an agent network need the same expressivity of policy representation; FANS introduces novel heuristics to automatically vary the FSM size in different agents for scale-up; (ii) FANS illustrates efficient integration of its FSM-based policy search within algorithms that exploit agent network structure; (iii) FANS provides significant speedups in policy evaluation and heuristic computations within the network algorithms by exploiting the FSMs for dynamic programming. Experimental results show not only orders of magnitude improvements over previous best known algorithms for smaller-scale domains (with similar solution quality), but also a scale-up into double digits in terms of numbers of agents.

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, Finite State Machines

Cite as: Not All Agents Are Equal: Scaling up Distributed POMDPs for Agent Networks, Janusz Marecki, Tapana Gupta, Pradeep Varakantham, Milind Tambe, Makoto Yokoo, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons(eds.), May,12-16.,2008,Estoril,Portugal,pp. 485-492.

1. INTRODUCTION

Many current and proposed applications of networks of agents, including mobile sensor networks, autonomous underwater vehicles, involve 100s of agents acting collaboratively under uncertainty [8, 10]. Distributed Partially Observable Markov Decision Problems (Distributed POMDPs) are ideally suited to plan for such agent networks, given their ability to plan in the presence of transitional and observational uncertainty in teams [9, 5, 1, 2, 13]. Unfortunately, the problem of finding the optimal joint policy for general distributed POMDPs is NEXP-Complete [3].

While this negative complexity result has not deterred some researchers from continuing to pursue global optimality in the general case [13, 7], a more popular approach has been to focus on tradeoffs for the sake of efficiency. Two types of tradeoffs have been examined. First, researchers have examined approximate techniques that sacrifice global optimality for efficiency [2, 9, 11, 5]. Second, researchers have focused on finding relevant subclasses of the general distributed POMDPs which would fit key real-world situations [1, 10]. Such real-world inspired restricted versions of the general distributed POMDPs have led to more efficient algorithms.

This paper follows the latter approach by focusing on network distributed POMDPs (ND-POMDPs) that are inspired by real-world domains, such as sensor agent networks [8, 10] or mobile WiFi router networks [4]. In particular, the paper provides a new algorithm called FANS (FSM-based Agent Network Search for policies) for ND-POMDPs. FANS exploits finite state machines (FSMs) for compact policy representation [2, 12], and uses three key novel ideas. First, a key insight in FANS is that not all agents in a network require the same expressivity in policy representation; hence, FANS introduces several heuristics to automatically vary the FSM sizes in different agents for scale-up. These heuristics also provide FANS with anytime properties. Second, FANS integrates its FSM-based policy search within algorithms that exploit agent network structure. Third, FANS shows that FSMs enable dynamic programming in policy evaluation and heuristic computation within the agent network algorithms.

We experimented with the sensor network domain, a domain representative of an important class of problems with networks of agents working in uncertain environments. FANS provides orders of magnitude improvement over SPIDER [14] as well as LID-JESP [10], two competing algorithms for ND-POMDPs, without significant loss in solution quality. Furthermore, our experimental comparison of different heuristics illustrates that smarter heuristics perform better in smaller scale domains, but their overheads overwhelm computation as we scale up.

2. DOMAIN: AGENT NETWORKS

Our work is motivated by agent networks that need to coordinate locally to achieve a global goal. Distributed sensor networks belong to this category of domains. Sensor networks have traditionally been used to track weather phenomenon, moving targets etc [8]. Recently, LANdroids [4] has been established as a significant domain, where within a sensor network, agents (each of which can only communicate with its neighbors) must position themselves to build a communication network with a strong signal strength. In this paper, we focus on an agent (sensor) network that is used for tracking targets [8]. We use the specifications of this domain provided in [10].

Figure 1 shows a specific problem instance consisting of four 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 location1, sensor1 needs to scan 'East' and sensor2 needs to scan 'West' simultaneously. Thus, sensors have to act in a coordinated fashion. There are typically multiple independent targets, and each target's movement is uncertain and unaffected by the sensor agents [8]. 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.

Figure 1: A 4-chain sensor configuration

3. NETWORK DISTRIBUTED POMDPS

Distributed POMDPs are well-suited to model the sensor network problems introduced in the previous section, given the sensors' observational uncertainty, the targets' uncertain transitions and the distributed nature of the nodes. However, instead of a general distributed POMDP model [3], a model tailored to such networks allows for more efficiency. In particular, akin to other sensor networks deployed for tracking phenomenon or targets, the sensor nodes are primarily independent. The only dependence arises from the fact that to track a target, two agents must coordinate by scanning the same region. Thus, when modelling the sensors as distributed POMDPs, the independence of sensor nodes translates into the sensors' observations and transitions being independent of each other's actions, while the dependence (arising from the need to track targets) translates into a joint reward function.

Thus to model such agent networks, the ND-POMDP model was introduced in [10]. Formally, for a group of n agents an ND-POMDP is defined as the tuple $\langle I, S, A, P, \Omega, O, R, b\rangle$, where $I = \{1, ..., n\}$ is a set of agent indices and $S = \times_{i\in I} S_i \times S_u$ 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 [1]). $A = \times_{i\in I} A_i$ is the set of joint actions, where A_i is the set of actions for agent i. $\Omega = \times_{i\in I}\Omega_i$ is the set of joint observations where Ω_i is the set of observations for agent i. To model the independence of transitions, the transition function in ND-POMDPs is defined as $P(s, a, s') = P_u(s_u, s'_u) \cdot \prod_{i\in I} P_i(s_i, s_u, a_i, s'_i)$, 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. Similarly, to model the independence of observations, the observation function is defined as $O(s', a, \omega) = \prod_{i\in I} O_i(s'_i, s'_u, a_i, \omega_i)$, where $s' = \langle s'_1, \ldots, s'_n, s'_u\rangle$ is the world state that results from the agents executing $a = \langle a_1, \ldots, a_n\rangle$ in the previous state, and observing $\omega = \langle\omega_1, \ldots, \omega_n\rangle \in \Omega$ 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.

Coordination among agents is achieved by the joint reward function, R, decomposable into reward functions R_l for all possible groups $l \subset I$ of agents whose joint action yields a reward specified for the domain (e.g. a reward for tracking a target). Precisely, $R(s, a) = \sum_{l\in E} R_l((s_i)_{i\in l}, s_u, \langle a_i\rangle_{i\in l})$ where E is a set of hyper-links[1] and I is the set of vertices in a domain hypergraph $G = (I, E)$ that we simply refer to as the *interaction graph*. For example, in Figure 1, I = {sensor1, sensor2, sensor3, sensor4 } and E = {(sensor1), (sensor2), (sensor3), (sensor4), (sensor1,sensor2), (sensor2,sensor3), (sensor3,sensor4)}.

The initial belief state $b = (b_u, b_1, ..., n_n)$ is the initial distribution over $s = (s_1, ..., s_n, s_u) \in S$ defined as $b(s) = b_u(s_u) \cdot \prod_{i\in I} b_i(s_i)$, where b_u and b_i are the initial distributions over S_u and S_i respectively. The goal in ND-POMDP is to compute the joint policy $\pi = \langle\pi_1, ..., \pi_n\rangle$ that maximizes agent group's expected reward over a finite action horizon T starting from the belief state b.

4. FANS

Previous work on distributed POMDPs, including ND-POMDPs [14, 10], has provided policies of equal expressive power to all agents. However, in agent networks, not all agents may be burdened with tasks of the same complexity. For example, in the sensor network domain to track targets, agents in some areas of the network may face far fewer targets to track than other areas of the network. If observing scientific phenomena, not all agents in the network face observation tasks of equal complexity.

Thus, the key insight in our work is that not all agents be provided policies of the same expressive power, and such "inequality" may be accomplished without any significant loss in solution quality. A centralized planner can thus focus its available planning time on agents that require careful planning with more detailed plans, and less on agents which do not, whereas a distributed policy generation algorithm [6] can benefit from the reduced communication burden across the network. In order to realize our insights we choose to represent agent policies with finite-state machines and vary their expressivity by varying the number of FSM nodes. In this section we explain this process in detail.

4.1 FANS algorithm

At a basic level, the FANS algorithm repeatedly searches for a best joint policy in the space of deterministic FSMs of given sizes. To this end, at each iteration it (i) uses heuristics to identify the agents whose policies will gain expressivity, (ii) expands the FSMs of these agents with additional FSM nodes and finally (iii) calls a joint policy search function to find the best joint policy in the space of deterministic FSMs of new sizes — this iteration is repeated until the gain in reward from expanding FSMs in one iteration is below a certain threshold. Before we describe the FANS algorithm in detail, we first formalize the class of FSMs that FANS operates on.

[1]In general, E can contain hyper-links l such that $|l| \neq 2$. For example, when sensors triangulate target's position, $|l| = 3$ whereas if a single sensor can track a target, $|l| = 1$.

While both stochastic and deterministic controllers have been used for single-agent POMDPs [2, 12], FANS uses deterministic FSMs. A deterministic FSM of agent i is defined as the tuple: $\langle Q_i, \psi_i, \eta_i, q_i^0 \rangle$ where Q_i is the finite set of FSM nodes, $\psi_i : Q_i \rightarrow A_i$ is an action selection function, $\eta_i : Q_i \times O_i \rightarrow Q_i$ is an FSM transition function, and $q_i^0 \in Q_i$ is the starting node of the FSM which allows FANS to uniquely determine an action of agent i at every time step t given its observations $(\omega_1, ..., \omega_{t-1}) \in O_i$ received at times $0, ..., t-1$. $|Q_i|$ denotes the set of FSM nodes in the policy for agent i.

Algorithm 1 FANS(Q)

```
1: π ← JOINT-POLICY-SEARCH(root, null, −∞, Q)
2: V_max ← V(π)
3: improving ← True
4: while improving do
5:    Q ← EXPAND-FSMS(Q)
6:    π ← JOINT-POLICY-SEARCH(root, null, −∞, Q)
7:    if V(π) − V_max > δ then
8:       V_max ← V(π)
9:    else
10:      improving ← False
11: return π
```

The FANS algorithm (Algorithm 1) is called with the initial vector $|Q| = (|Q_1|, ..., |Q_n|)$ of agent FSM sizes which may or may not be uniform — $|Q_i|$ are initially small. First, JOINT-POLICY-SEARCH (further discussed in Section 4.3), which takes Q as input, provides the optimal joint policies in space of FSMs of sizes $|Q|$. FANS then iterates: in each iteration it uses heuristics to choose which FSMs will be expanded and then expands these FSMs (line 5), and given the new FSM sizes, again searches for a new optimal joint policy (line 6). This iteration continues until FANS's convergence criterion is met, which is when the expected value $V(\pi)$ of the joint policy π obtained in the current iteration does not exceed the maximum expected value V_{max} obtained thus far by δ. Notice, that because FANS always has an optimal joint policy π given current Q, it has the *anytime* property.

4.2 Not All Agents are Equal: Heuristics

This section focuses on the heuristics used for the EXPAND-FSMS function of Algorithm 1 that decide which agents' FSMs should be provided with more nodes. We start with simpler, low-overhead heuristics and gradually increase the complexity. A key question we wish to ask is how much computational investment should FANS make in intelligently selecting the FSMs to expand. Through these heuristics, we address the trade-off between the runtime necessary to execute each heuristic and runtime necessary to execute the JOINT-POLICY-SEARCH function.

Equality: This heuristic is a baseline for comparison: it has no selectivity and simply adds one node to the FSMs of all agents, i.e., it sets $|Q_i| \leftarrow |Q_i| + 1$ for all agents $i \in I$.

Greedy: In this heuristic we greedily add nodes to FSMs of agents that have more connectivity with other agents, since more connectivity usually requires additional coordination that can be achieved by more complex policies. Formally, for the interaction graph $G = (I, E)$ defined earlier, let $N_i = \{j \in I : \langle i, j \rangle \in l, l \in E\}$ denote the set of neighbors of agent i. Also, let $\overrightarrow{I} = (i_1, ..., i_n)$ denote a list of agents $i \in I$ arranged in decreasing number of their neighbors, i.e., $|N_{i_k}| \geq |N_{i_{k+1}}|$ and $p \in \{1, ..., n\}$ be a pointer to a position on the list $\overrightarrow{I}$ ($p \leftarrow 1$). Now, each time EXPAND-FSMS is called, it greedily expands FSMs of agents with higher number of neighbors, yet does not discriminate agents with lower number of neighbors. That is, for a pointer p from the previous call of EXPAND-FSMS, the function increases by one node FSMs of agents $i_p, ..., i_{p+k}$ that have the same number of neighbors, i.e., $|N_{i_p}| = ... = |N_{i_{p+k}}| > |N_{i_{p+k+1}}|$; FSMs of other agents remain intact. The function then sets p to point to the next agent on the list $\overrightarrow{I}$ whose FSM did not receive an additional node, i.e., it updates $p \leftarrow (p + k + 1) \bmod n$.

Node: The Node heuristic does not use connectivity information at all. In particular, the Node heuristic asks the question: what is the gain in the joint expected utility if we enable one particular agent to compute its best policy, assuming full observability and unbounded FSM size, given the fixed policies of all other agents (from a previous iteration of JOINT-POLICY-SEARCH). In other words, we use an MDP to obtain an *upper-bound* on the expected utility gained by increasing the FSM size of a particular agent while keeping the policies of other agents fixed (from previous FANS iteration). We use this upper-bound as an indicator of how deserving that particular agent is of an additional FSM node. Thus, for each agent $i \in I$, we first compute the upper bound on the utility gain g_i (using a method described in Section 4.4) and then construct a list $\overrightarrow{I} = (i_1, ..., i_n)$ of agents $i \in I$ sorted in decreasing value of their upper bounds, i.e., $g_{i_m} \geq g_{i_{m+1}}$. We finally add an additional node to FSMs of first $\lfloor kn \rfloor$ agents in list $\overrightarrow{I}$, where $k \in (0, 1]$.

Link: This heuristic is similar to the Node heuristic above, except that instead of considering individual agents when choosing which FSM should get additional nodes, it considers agent links. Formally, for each link $l \in E$ in the interaction graph $G = (I, E)$ the Link heuristic computes the upper bound on link l, denoted as g_l, understood as the maximum utility gained by increasing the FSM size of each agent $i \in l$ by one additional node while keeping the policies of other agents fixed (from previous FANS iteration). The Link heuristic then selects the link l with the highest g_l and sets $|Q_i| \leftarrow |Q_i| + 1$ for FSMs of all agents $i \in l$.

Searcher: The most complex of our heuristics, Searcher attempts a more systematic search of the FSM space using ideas from the Node heuristic, but goes one step further by attempting to very precisely estimate the gain by adding FSM nodes before assigning an additional node to an agent. In other words, by running JOINT-POLICY-SEARCH for itself, Searcher attempts to verify that adding one FSM node to an agent while keeping the FSM size of all other agents the same will actually provide benefits in terms of expected value before increasing the FSM size of that agent. Thus, we iterate over all agents $i \in I$. During each iteration, we set $|Q_i| \leftarrow |Q_i| + 1$ and obtain $V_i \leftarrow$ JOINT-POLICY-SEARCH $(root, null, -\infty, Q)$, which provides the highest V_i possible given this Q (as described in Section 4.3). Now, if $V_i \leq V_{max}$(computed in the last iteration of FANS), we reset $|Q_i| \leftarrow |Q_i| - 1$, i.e., if adding one additional node to agent i did not increase the expected value, the size of its FSM stays the same as before. However, if the additional node did result in an increase in expected value, we keep it. Searcher returns the updated vector Q after iterating over all $i \in I$.

Fairness: This heuristic is a compromise between Greedy and Searcher. In particular, rather than terminating where Greedy would terminate, it continues down the sorted agent list $\overrightarrow{I}$, offering each agent one additional chance to increase the size of their FSMs, until the end of the sorted list is reached.

4.3 Joint Policy Search

In solving an ND-POMDP, the goal is to compute a joint policy that maximizes the overall expected joint reward. The Joint Policy Search within FANS uses the idea from SPIDER [14] of branch and bound search with upper bounds. To this end, as in SPIDER,

a DFS tree is constructed [14] corresponding to the reward interaction structure, to exploit the locality in interactions among agents. In short, if agent i needs to coordinate with agent j, i.e., there exist $l \in E$ in the interaction graph $G = (I, E)$ such that $\langle i, j \rangle \in l$ then either i is an ancestor of j or j is an ancestor of i in the corresponding DFS tree.

However, a fundamental difference from SPIDER is that the policy representation in FANS is based on FSMs of varying sizes instead of policy trees. In this paper, we employ the following notation to denote agents in the DFS tree, their policies and expected values:
$A^i \Rightarrow$ ancestor agents, i.e. agents from i to the $root$ (excluding i) .
$S^i \Rightarrow$ agents in the sub-tree (excluding i) for which i is the root.
$\pi \Rightarrow$ joint policy of all agents.
$\pi^{i-} \Rightarrow$ partial joint policy of agents that are in A^i.
$\pi^{i+} \Rightarrow$ partial joint policy of all agents in $S^i \cup i$.
$\pi_i \Rightarrow$ 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-}.
$\hat{v}[\pi_i, \pi^{i-}, j] \Rightarrow$ upper bound on the expected value for π^{i+} from the jth child of i.
$v[\pi_i, \pi^{i-}] \Rightarrow$ 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-}.
$v[\pi^{i+}, \pi^{i-}, j] \Rightarrow$ expected value for π^{i+} from the jth child of i.
$\Pi_i \Rightarrow$ set of all possible FSMs of agent i.

Algorithm 2 JOINT-POLICY-SEARCH$(i, \pi^{i-}, threshold, Q)$

```
1: π^{i+,*} ← null
2: Π_i ← GET-ALL-FSMS (T, A_i, Ω_i, |Q_i|)
3: if IS-LEAF(i) then
4:    for all π_i ∈ Π_i do
5:       v[π_i, π^{i-}] ← JOINT-REWARD-DP (π_i, π^{i-})
6:       if v[π_i, π^{i-}] > threshold then
7:          π^{i+,*} ← π_i
8:          threshold ← v[π_i, π^{i-}]
9: else
10:   Π̃_i ← UPPER-BOUND-SORT-DP(i, Π_i, π^{i-})
11:   for all π_i ∈ Π̃_i do
12:      π̃^{i+} ← π_i
13:      if v̂[π_i, π^{i-}] < threshold then
14:         Go to line 11
15:      for all j ∈CHILDREN (i) do
16:         threshold(j) ← threshold − v[π_i, π^{i-}]
17:                        −Σ_{k∈CHILDREN(i)\j} v̂_k[π_i, π^{i-}]
18:         π^{j+,*} ←JOINT-POLICY-SEARCH(j, π^{i-} ∪ π_i,
19:                                       threshold(j), Q)
20:         π̃^{i+} ← π̃^{i+} ∪ π^{j+,*}
21:         v̂_j[π_i, π^{i-}] ← v[π^{j+,*}, π_i ∪ π^{i-}]
22:      if v[π̃^{i+}, π^{i-}] > threshold then
23:         threshold ← v[π̃^{i+}, π^{i-}]
24:         π^{i+,*} ← π̃^{i+}
25: return π^{i+,*}
```

Algorithm 2 shows the algorithm for Joint Policy Search within FANS. We illustrate the key ideas in its execution using Figure 2 on the four agent example from Figure 1. Agents are arranged in a DFS tree with agent 3 as the root. For simplicity of exposition we consider the case where we have assigned two node FSMs for agents 2 and 3, and 1 node for agents 1 and 4. The algorithm begins with no policy assigned to any of the agents (Level 1 of the search tree in Figure 2). In the next step (Level 2 of the search tree), there is a search node corresponding to each policy of the root agent (i.e. agent 3). Except for the root agent, no other agent is assigned any policies. These new nodes in the search tree (corresponding to each of root agents policies) are sorted based on upper bounds (on expected value) of the partial joint policies (line 10 of algorithm 2). This upper bound computation will be discussed in the next section. The search node with the highest upper bound is further expanded. In Figure 2, this corresponds to the search node on Level 2 with upper bound 250 (number on top right hand corner of the search node). Expansion here implies creation of new search nodes corresponding to policies of children agents (in the DFS tree) to the root agent (line 18). Upper bounds are computed for these new nodes and the nodes are sorted based on these upper bounds. This process continues until a complete joint policy (a policy assigned to each of the agents) is obtained. We must run a policy evaluation algorithm to obtain the value of this joint policy (line 5), which is described in the next section. Level 4 of the search tree provides a search node with a complete joint policy and expected value of 244. This value of the best known complete joint policy can be used to prune out search nodes with upper bounds less than it. For instance, all the nodes at Level 3 and Level 2 can be pruned out since their upper bounds are less than or equal to 244.

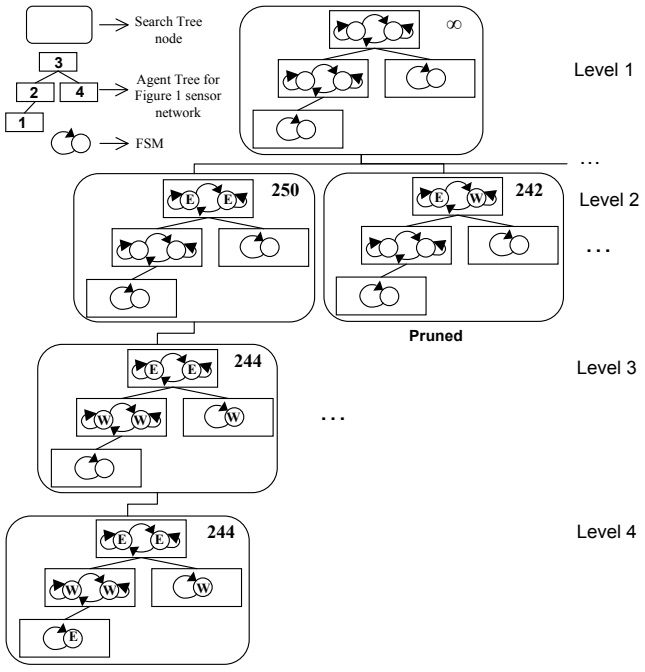

Figure 2: Execution of Joint Policy Search, an example

4.4 FANS: Dynamic programming

In this section, we describe evaluation of the joint policy (JOINT-REWARD-DP) and computation of the upper bound for partial joint policies using an MDP heuristic (UPPER-BOUND-SORT-DP). These are two major steps in joint policy search that can be significant bottlenecks. These bottlenecks result from the fact that joint policy trees have to be constructed and evaluated for all agents for policy evaluation and all agents in A^i for the MDP heuristic. We first describe these computations and then explain how FSMs can help speed them up using dynamic programming.

Let i be an agent on the DFS tree, E^{i-} denote the set of links connecting agents in $A^i \cup i$ with agents in S^i, and E^{i+} denote the set of links connecting agents in S^i. Furthermore, for each link l in either E^{i-} or E^{i+}, let $l(1), l(2) \in l$ denote the agents at the ends of

l; assume WLOG that for $l \in E^{i-}$, agent $l(1)$ is in $A^i \cup i$ whereas agent $l(2)$ is in S^i. Also, let $z = \langle s_u, (s_i)_{i=1}^n, (q_i)_{i=1}^n \rangle$ denote the current *multiagent state* where s_u is the current unaffected state, $(s_i)_{i=1}^n : s_i \in S_i$ is the vector of current agent states and $(q_i)_{i=1}^n : q_i \in Q_i$ is the vector of current nodes of agent FSMs.

We now recall how [14] derives $v_z^t :=$ the total expected reward for all agents if they execute a joint policy π (specified by a joint action selection function $(\psi_1, ..., \psi_n)$ of agent FSMs) from time $t \in [0, ..., T]$ until time T from the multiagent state z. To this end, we first show how to derive $v_z^t[\pi^{i-}, \pi^{i+}] :=$ the expected reward for agents in S^i if they execute a *partial joint policy* π^{i+} from time $t \in [0, ..., T]$ until time T from the multiagent state z. We can derive $v_z^t[\pi^{i-}, \pi^{i+}]$ using the following recursive equations:

$$v_z^t[\pi^{i-}, \pi^{i+}] = \sum_{l \in E^{i-}} v_{z,l}^t + \sum_{l \in E^{i+}} v_{z,l}^t$$

where

$$v_{z,l}^t = R_{z,l} + \sum_{z' = \langle s'_u, (s'_i)_{i=1}^n, (q'_i)_{i=1}^n \rangle} p_{l(1)} \cdot p_{l(2)} \cdot p_u \cdot o_{l(1)} \cdot o_{l(2)} \cdot v_{z',l}^{t+1} \quad (1)$$

where $R_{z,l} = R_l((s_{l(1)}, s_{l(2)}), s_u, \langle \psi_{l(1)}(q_{l(1)}), \psi_{l(2)}(q_{l(2)}) \rangle)$ and:

$$o_i = O_i(s'_i, s'_u, \psi_i(q_i), \omega'_i)$$
$$p_i = P_i(s_i, s_u, \psi_i(q_i), s'_i)$$
$$p_u = P(s_u, s'_u)$$

And the recursion continues until $t = T$. Observe that to calculate v_z^t we then simply need to calculate $v_z^t[\emptyset, \pi^{j+}]$ where $j \in I$ is the root agent in the DFS tree.

On the other hand, to derive the upper bounds on expected values $v_z^t[\pi^{i-}, \pi^{i+}]$ of partial joint policies π^{i+}, denoted as $\hat{v}_z^t[\pi^{i-}, \pi^{i+}]$, [14] uses an MDP based heuristic that we now briefly discuss. Given fixed policies of agents in $A^i \cup i$, this heuristic computes an upper bound on the expected value obtainable from the agents in S^i (Step 10 of Algorithm 2). 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. In Equation 1, the p_i and o_i terms refer to the transition and observation probabilities respectively for agent i, when it transitions from state s_i to s'_i . Upper bound on the expected value for a link $l \in E^{i-} \cup E^{i+}$ can be computed by modifying Equation 1 to reflect the full observability assumption which involves removing the observational probability term for agents in S^i and maximizing the future value $\hat{v}_{z,l}^t$ over the actions of those agents (in S^i). Thus, we compute the upper bound on a link l as follows: If $l \in E^{i-}$ then

$$\hat{v}_{z,l}^t = \max_{a \in A_{l(2)}} \{R_{z,l} + \sum_{z'} p_{l(1)} \cdot p_{l(2),a} \cdot p_u \cdot o_{l(1)} \cdot \hat{v}_{z',l}^{t+1}$$

Whereas if $l \in E^{i+}$ then

$$\hat{v}_{z,l}^t = \max_{a' \in A_{l(1)}, a'' \in A_{l(2)}} \{R_{z,l} + \sum_{z'} p_{l(1),a'} \cdot p_{l(2),a''} \cdot p_u \cdot \hat{v}_{z',l}^{t+1}\}$$

Where $p_{i,a} = P_i(s_i, s_u, a, s'_i)$.

We now show our next contribution: the application of dynamic programming for policy evaluation and upper bound computation when agent policies are represented using FSMs. Such dynamic programming was not feasible using the traditional representation [10, 14], since the size of the joint policy tree grows exponentially in $|\Omega| = | \times_{1 \le i \le n} \Omega_i|$. Hence, both the policy evaluation and the upper bound computation were slow. On the other hand, with FSMs, the policy of agent i at time t is uniquely determined by $\psi_i(q_i)$ where $q_i \in Q_i$ is the current node of the FSM of agent i (at time t) as opposed to agent i's observation history up to time t. In general, joint agent policy is determined by the current nodes of agents' FSMs, i.e., $(q_i)_{i=1}^n$ as opposed to joint observation histories of all agents up to time t. The following theorem shows how to perform both the policy evaluation and the MDP heuristic computation without considering joint observation histories of all agents:

THEOREM 1. *Let $V(t)$ be the set of values v_z^t for all possible multiagent states at time t. It then holds that:*

***(i)** Size of $V(t)$ only depends on $|S|$ and $\prod_{i \in I} |Q_i|$ and thus, does not increase with the planning horizon T.*

***(ii)** $V(t)$ can be derived from $V(t+1)$ for all $t \in [0, ..., T]$*

PROOF. *Statement (i) holds because $|V(t)| \le |S| \prod_{i \in I} |Q_i| = K$. We prove statement (ii) by induction on t from T to 0:*

***Induction base**: Assume $t = T$. In this case, for all $v_z^T \in V(T)$, $v_z^T = \sum_{l \in E} R_l$ because T is the planning horizon after which no reward can be earned.*

***Induction step**: Suppose that the theorem holds for t and we want to prove it for $t-1$, i.e. derive the values $v_z^{t-1} \in V(t-1)$ in terms of values $v_z^t \in V(t)$ that have already been found. Let $v_z^{t-1} \in V(t-1)$ be a value to be derived for some multiagent state $z = \langle s_u, (s_i)_{i=1}^n, (q_i)_{i=1}^n \rangle$ at time t. We know that the joint action $a = (a_i)_{i=1}^n = (\psi_i(q_i))_{i=1}^n$ will be executed from state $s = \langle s_1, ..., s_n, s_u \rangle$ and with probability $p(s, a, s') = P_u(s_u, s'_u) \cdot \prod_{1 \le i \le n} P_i(s_i, s_u, a_i, s'_i)$ the system will transition to a new state $s' = \langle s'_1, ..., s'_n, s'_u \rangle$. Then, agents will observe $\omega = (\omega_1, ..., \omega_n)$ with probability $o(s', a, \omega) = \prod_{1 \le i \le n} O_i(s'_i, s'_u, a_i, \omega_i)$ and consequently agent FSMs will transition to nodes $(q'_i)_{i=1}^n$, where FSM i transitions under observation ω_i from node q_i to node q'_i. At this point reward R_l is earned and the system finds itself the multiagent state $z' = \langle s'_u, (s'_i)_{i=1}^n, (q'_i)_{i=1}^n, \rangle$ for which (from the induction assumption) we already know the total expected reward $v_{z'}^t \in V(t)$. Consequently, we have shown how to derive the values $v_z^{t-1} \in V(t-1)$ in terms of values $v_{z'}^t \in V(t)$.* □

The immediate consequence of Theorem 1 is that the expected value of a policy π started from a belief state b is given by:

$$\sum_{(s_u, s_1, ..., s_n) \in S} b_u(s_u) \prod_{1 \le i \le n} b_i(s_i) \cdot v_{z^0(s)}^0$$

Where $z^0(s) = \langle s_u, (s_i)_{i=1}^n, (q_i^0)_{i=1}^n \rangle$ is the starting multiagent state for $s \in S$.

We can now more clearly see the reason why policy evaluation given FSM representation of agent policies is much faster than policy evaluation given observation histories representation of agent policies: In the former case policy evaluation is done by calculating $v_z^t \in V(t)$ for all $t \in [0, ..., T]$ which can be done in polynomial time $O(KT)$ whereas in the latter case policy evaluation runs in exponential time $O(| \times_{1 \le i \le n} \Omega_i|^T)$. Finally, our dynamic programming technique differs significantly from [7] as we are not backing up policy trees.

5. EXPERIMENTAL RESULTS

This section provides six sets of experimental results on FANS. The first three experiments focus on runtime and scale-up, comparing FANS with SPIDER (a globally optimal algorithm) [14] and LID-JESP (a locally optimal algorithm) [10], which are competing algorithms for ND-POMDPs; while the remaining results focus on solution quality and analysis. All our experiments were conducted

on the sensor network domain provided in Section 2. Our first experimental result focuses on providing a head-to-head comparison of FANS with different heuristics, against SPIDER, since SPIDER is the latest algorithm for ND-PODMPs, and also because it uses branch-and-bound search using an MDP heuristic. The comparison uses some of the sensor network configurations presented in [14], and an additional "7-H" configuration as shown in Figure 3. In this experiment, we limit the number of agents to less than 10, in order to allow SPIDER to run.

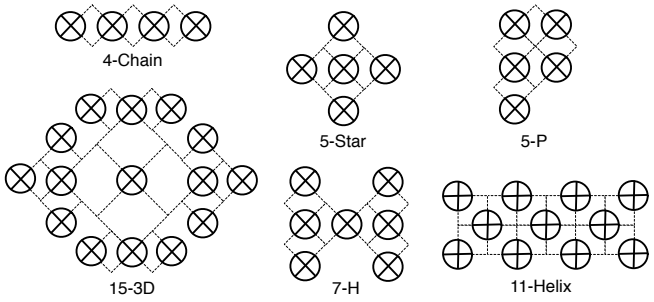

Figure 3: Sensor network configurations

Figure 4 shows a comparison of runtime of SPIDER vs. FANS with different heuristics. The x-axis shows the different sensor network domain configurations, and the y-axis plots the runtime in log-scale (in all our experiments, we imposed a 10000 second cutoff on runtime). Also, we chose a time horizon of 3 for policy computation, to be consistent with the results shown in [14]. For the node heuristic, we chose $k = 1/2$; we discuss this choice of k later on. We can observe that (i) SPIDER is the slowest of all of the different algorithms considered and it hits the 10000 sec cutoff in the 5-P and 7-H cases; (ii) The Node and Link heuristics are the two fastest in all cases; (iii) Unfortunately, Searcher's extra search over FSMs appears have a significant runtime penalty; (iv) The Equality and Fairness heuristics are the two worst performing heuristics; (v) Surprisingly, the Greedy heuristic matches the performance of the Link heuristic for the 5-P and 7-H cases. The key conclusion from this graph is that FANS with either the Node, Link or Greedy heuristic is able to provide more than two orders of magnitude speedup over SPIDER.

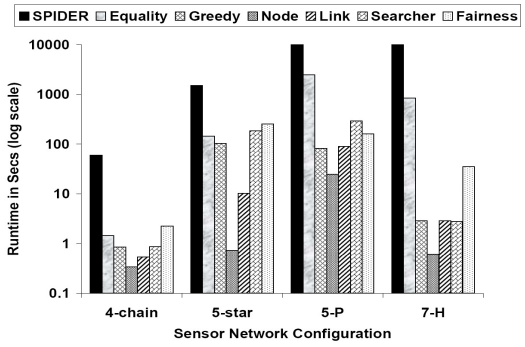

Figure 4: Runtime comparison of SPIDER vs. FANS

Next, we show scale up in terms of time horizon for policy computation, and also provide a comparison of FANS against the LID-JESP algorithm, a locally optimal algorithm for ND-POMDPs [10]. Runtime results for LID-JESP are available up to T=6, as shown below. Figure 5 shows a plot of the runtime as we increase the time horizon. The x-axis shows the time horizon, and the y-axis plots the runtime in log-scale. Runtime values for different time horizons are shown for LID-JESP and FANS with Node heuristic ($k = 1/2$), for the 4-chain and 5P configurations. The key observations here are that FANS is able to provide orders of magnitude speedup over LID-JESP for higher time horizons. For example, for the 4-chain configuration with T=6, LID-JESP runs for 309 s, while FANS terminates in 0.73 s, thus providing a 423-fold speedup over LID-JESP. Moreover, for the 5-P configuration, a speedup of more than 1 order of magnitude is visible for T=5. We discuss solution quality in detail later, but for both 4-chain and 5-P, the Node heuristic with $k = 1/2$ appears to match the quality of the global optimal solution.

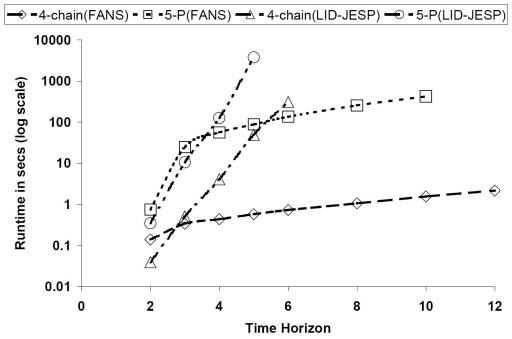

Figure 5: Runtime comparison of FANS with LID-JESP

Figure 6(a) shows a comparison of runtime of FANS with different heuristics as we scale-up the number of agents. The x-axis shows the different sensor network domain configurations[2] (see Figure 3), and the y-axis plots the runtime in log-scale (again, in all our experiments, we imposed a 10000 second cutoff on runtime). Also, the time horizon for policy computation is 3 and we again choose $k = 1/2$ for the Node heuristic. We can make the following observations: (i) The Equality and Searcher heuristics are not able to terminate within the cutoff time of 10000 s; (ii) The Node heuristic still remains the best for all configurations, but Greedy beats Link in the 15-3D domain. Thus FANS can scale up to as many as 15 agents, triple the number of agents SPIDER could handle.

The next set of experiments examine tradeoffs in solution quality and provide analysis of these speedups. Table 1 shows a comparison of solution quality for SPIDER and FANS with select heuristics. The rows show all agent configurations considered so far while the columns show the algorithm used to obtain the solution quality. We can observe the following:(i) For the 4-chain and 5-star domains, FANS (even with its faster Greedy heuristic) is able to reach the same quality as SPIDER, which is the global optimum. For the 5-P configuration, only an upper bound on the solution quality is provided in [14], and FANS is able to obtain a solution quality within 5% of the upper bound obtained by SPIDER. (ii) The Equality heuristic obtains the highest solution quality in all cases; (iii) For domains of less than 10 agents, the Link heuristic matches the quality of the Equality heuristic (it loses less than 1% quality and less than 10% of quality in the 5-star and 7H domains respectively); (iv) As we scale up the Network, none of the three fastest heuristics (Link, Node, Greedy) can keep up to the solution quality provided by Equality or Searcher; (v) Even though the Equality and Searcher

[2]Configuration 15-mod is a modification to the 15-3D configuration with different target paths.

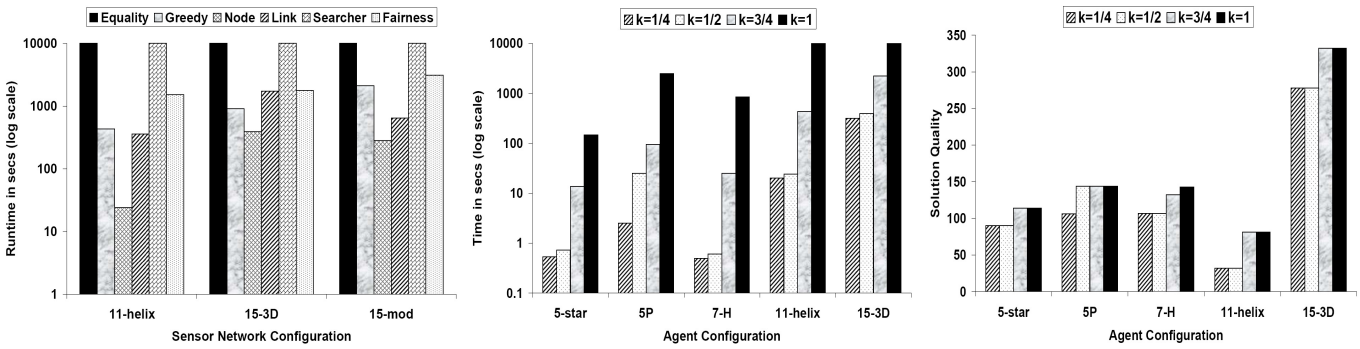

Figure 6: (a) Runtime comparison of FANS using different heuristics for larger domains using T=3, (b) runtime performance of Node Heuristic with varying 'k', (c) solution quality of node heuristic with varying 'k'

heuristics do not terminate within the cutoff time limit for the larger domains, we still are able to provide quality results (marked with *) — this is due to the anytime nature of FANS (discussed later). We simply provide the solution quality obtained from the last iteration of FANS that terminated within the cutoff time limit. Overall, taking into account both runtime and solution quality, for smaller-scale networks, the Link heuristic may provide the best tradeoff of time to solution quality; while the Greedy heuristic might be preferred for the larger networks. The Node heuristic is always the fastest, but may lose upto 35% of solution quality.

Table 1: Comparison of solution quality

	SPIDER	Equality	Link	Node	Greedy	Searcher
4-chain	226.6	226.6	226.6	226.6	226.6	226.6
5-star	114.4	114.4	113.8	90.1	114.4	114.4
5-P	-	143.9	143.9	143.9	143.9	143.9
7-H	-	142.8	132.3	106.7	106.7	106.7
11-helix	-	113.3*	80.8	32.2	80.8	113.3*
15-3D	-	332.4*	332.4	278.2	332.4	332.4*
15-mod	-	322.7*	207.2	207.2	304.9	322.7*

So while the Node heuristic is the fastest, it loses in solution quality. Our next experiment illustrates that varying k provides us solution time vs quality tradeoffs in the Node heuristic. Figure 6(b) shows how k affects the runtime of the Node Heuristic for different configurations (k=1 is the Equality heuristic). The x-axis shows the different sensor network domain configurations, and the y-axis plots the runtime in log-scale. It is easy to see that for each example, the runtime decreases as we decrease k from 1 to 1/4. For $k = 3/4$, the runtime for 11-helix and 15-3D is comparable to the Greedy heuristic. Figure 6(c) shows how k affects the solution quality of the Node Heuristic for different configurations. From this graph, we can observe that k=3/4 may provide the desired increase in solution quality. Thus, k=1/2 may be appropriate where speed is of the essence, but increasing k is valuable if solution quality is critical.

As noted in the previous experiment, for larger scale domains, the runtimes for Node and Link start approaching the Greedy heuristic. Our next experiment explains why. Figure 7(a) shows the time overhead to compute the upper bound using an MDP for the Node and Link heuristics. The x-axis shows the different sensor network domain configurations, and the y-axis plots the runtime in log-scale. Node (Overhead) denotes the overhead for upper bound computation (described in Section 4.4, while Node (Total) denotes the total time taken by the Node heuristic. Link (Overhead) and Link (Total) should be interpreted similarly. Also, k=1/2 for the node heuristic. We can observe that the overhead percentage over the total runtime increases as we scale up the number of agents, for both the Node and Link heuristic. For example, for the Link heuristic, the overhead is 0.6% for the 5-P domain as compared to 83% for the 15-3D example. Thus, as we increase the number of agents, the Link and Node heuristics suffer due to the time overhead for upper bound computation.

In the next set of results, we provide a comparitive analysis of the performance of the different heuristics for FANS. It has been shown that the Equality heuristic is the slowest among all heuristics, while the heuristics that assign FSMs of varying size to different agents are faster. Figure 7(b) shows the size of FSMs for each agent using different heuristics for the 5-star configuration. The x-axis shows the different agents, and the y-axis shows number of nodes in an FSM. It may be recalled that all three heuristics shown here reached the same solution quality (Link loses less than 1%), with the Link heuristic being the fastest followed by Searcher and Equality. We observe that the Equality heuristic assigns 3 nodes to each agent while, the Link and Searcher heuristics assign FSMs of different sizes to each agent. The Link heuristic assigns the least number of nodes per agent and performs the best. Thus, not all agents require the same expressivity as far as FSM size is concerned.

Our final result examines the anytime nature of FANS. Figure 7(c) shows a plot of quality vs. runtime for different heuristics, for the 7-H domain. The x-axis shows runtime in seconds on log scale while the y-axis shows solution quality. The lines plotted in each of these graphs show the progression of different heuristics over time in terms of solution quality, illustrating that FANS can return progressively improved policies as we let it run longer before interruption; and we need not run it to completion. For example, interrupting a run of the Equality heuristic at 10 seconds would still return a plan of quality 106.7; and interrupting it at 90 seconds will yield a plan of quality 132.3, etc.

These experiments illustrate that FANS with heuristics is able to outperform its competitors SPIDER and LID-JESP by orders of magnitude in runtime, while showing promising scale-up beyond what has been attempted before in the literature in terms of numbers of agents. Among the FANS heuristics, the Searcher heuristic with the highest search overhead never wins; the Node and Link heuristics which have moderate heuristic overheads dominate for smaller-scale networks, but the lower-overhead Greedy heuristic may provide the right trade-off in larger scale networks.

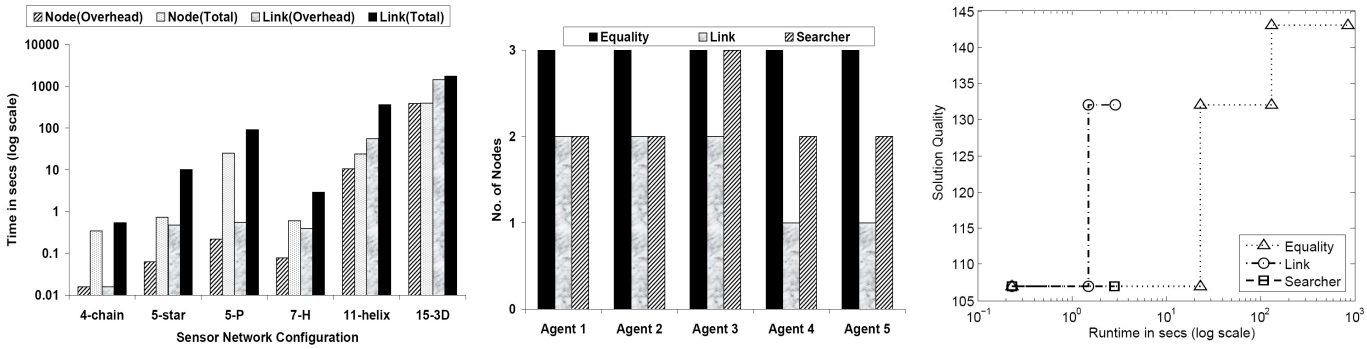

Figure 7: (a) Time overhead for Node and Link Heuristics for different configurations, (b) FSM size for each agent (5-star configuration), (c) solution quality vs. runtime for 7-H configuration

6. SUMMARY AND RELATED WORK

Recently there have been many exciting theoretical advances in Distributed POMDPs. It is now important for the field to take steps towards real practical applications. Agent networks, exemplified by sensor networks for target tracking [8], the recently developed LANDroids[4] domain, and others, are distributed applications with transitional and observational uncertainties; these agent networks appear ideally suited for distributed POMDPs. However, these applications also suggest tailoring distributed POMDPs, via models such as ND-POMDPs, and most importantly, they require algorithms that will demonstrate scale-up in number of agents. This paper is motivated by such a need for scale-up, and presents a new algorithm called FANS which shows scale-up in ND-POMDPs of up to 15 agents. FANS uses FSMs for policy representation and provides three key contributions: (i) FANS introduces novel heuristics to automatically vary the FSM size in different agents for scale-up; (ii) FANS exploits FSMs for dynamic programming in policy evaluation and heuristic computations and provides significant speedups; (iii) FANS illustrates efficient integration of its FSM-based policy search within an algorithm that exploit agent network structures.

In terms of related work, of key relevance are algorithms focused on agent networks. Of these, distributed POMDP algorithms for ND-POMDPs, SPIDER[14] and LID-JESP[10] are most relevant and we have already provided a detailed comparison of FANS with SPIDER and LID-JESP. Within the broader distributed POMDP arena, among globally optimal approaches, Hansen *et al.* [7] present an algorithm for solving partially observable stochastic games (POSGs) based on dynamic programming, while Szer *et al.* [13] provide an optimal heuristic search method for solving Decentralized POMDPs. These globally optimal algorithms have traditionally been demonstrated in domains involving two agents. Insights from FANS of differing the policy expressivity across agents, and heuristics for doing so may help in scaling up these algorithms. In related work focusing on approximate policies, Peshkin *et al.* [11] and Bernstein *et al.* [2] are examples of techniques that search for locally optimal policies using FSMs. Interactive POMDP (I-POMDP) model by [6] is presented as an alternative to the distributed POMDP model and particle filters have been proposed to solve them. The key difference between our work and theirs is our focus on scaling up in the agent network dimension, scaling up the number of agents into double digits: thus the techniques we provide to search for joint policies in an agent network, and our varying FSM nodes in the scale-up are key novelties missing in these previous works.

ACKNOWLEDGEMENTS

This material is based upon work supported by the DARPA/SBRI Contract W31P4Q-07-C-0146 and DARPA/STTR Contract No. W31P4Q-06-C-0410 via a subcontract from Perceptronics Solutions, Inc. The authors also want to thank the anonymous reviewers for their valuable comments.

7. REFERENCES

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

[2] D. S. Bernstein, E. 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] DARPA. Landroids. *http://fs1.fbo.gov/EPSData/ODA/Synopses/4965/BAA07-46/BAA07- 46LANdroidsPIP.pdf*

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

[6] P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

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

[8] V. Lesser, C. Ortiz, and M. Tambe. *Distributed sensor nets: A multiagent perspective*. Kluwer, 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] P. Poupart and C. Boutilier. Bounded finite state controllers. In *NIPS*, 2003.

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

[14] P. Varakantham, J. Marecki, Y.Yabu, M. Tambe, and M. Yokoo. Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In *AAMAS*, 2007.