Automated Generation of Interaction Graphs for Value-Factored Dec-POMDPs

William Yeoh
New Mexico State University
Las Cruces, NM 88011, USA
wyeoh@cs.nmsu.edu

Akshat Kumar
IBM Research
New Delhi, India, 110070
akshat.kumar@gmail.com

Shlomo Zilberstein
University of Massachusetts
Amherst, MA 01003, USA
shlomo@cs.umass.edu

Abstract

The Decentralized Partially Observable Markov Decision Process (Dec-POMDP) is a powerful model for multi-agent planning under uncertainty, but its applicability is hindered by its high complexity – solving Dec-POMDPs optimally is NEXP-hard. Recently, Kumar *et al.* introduced the Value Factorization (VF) framework, which exploits decomposable value functions that can be factored into subfunctions. This framework has been shown to be a generalization of several models that leverage sparse agent interactions such as TI-Dec-MDPs, ND-POMDPs and TD-POMDPs. Existing algorithms for these models assume that the interaction graph of the problem is given. In this paper, we introduce three algorithms to automatically generate interaction graphs for models within the VF framework and establish lower and upper bounds on the expected reward of an optimal joint policy. We illustrate experimentally the benefits of these techniques for sensor placement in a decentralized tracking application.

1 Introduction

Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs) have been shown to be effective models for planning under uncertainty involving a single decision maker. Consequently, Decentralized POMDPs (Dec-POMDPs) [Bernstein *et al.*, 2002] have emerged as a natural extension for modeling problems where a team of agents needs to plan under uncertainty. Scalability of Dec-POMDP algorithms, however, is a challenging issue because not only is finding optimal solutions NEXP-hard [Bernstein *et al.*, 2002], but finding constant factor approximations is also NEXP-hard [Rabinovich *et al.*, 2003].

In general, researchers have taken two approaches to address scalability. The first approach is motivated by the observation that many multi-agent planning problems, such as the sensor network problem we present, have *sparse agent interactions*, that is, each agent only interacts with a *limited* number of other agents. Thus, researchers have introduced specialized models such as ND-POMDPs [Nair *et al.*, 2005], TD-POMDPs [Witwicki and Durfee, 2010] and DPCL [Velagapudi *et al.*, 2011], which exploit the sparsity in these interactions to increase scalability. In the second approach, researchers assume that the problem can be factored into smaller factors. For example, Oliehoek *et al.* introduced models where the state space and reward function are factored into subspaces and subfunctions [Oliehoek *et al.*, 2008; Oliehoek, 2010]. More recently, Kumar *et al.* introduced the *Value Factorization* (VF) framework, where the value functions are factored into subfunctions [Kumar *et al.*, 2011]. This framework is appealing as it has been shown to unify and capture several existing sparse-interaction models like TI-Dec-MDPs [Becker *et al.*, 2004], ND-POMDPs [Nair *et al.*, 2005] and TD-POMDPs [Witwicki and Durfee, 2010]. We thus describe our work in the context of the VF framework and use ND-POMDPs as one example model within this framework.

All the value-factored Dec-POMDP algorithms developed thus far assume a *given* interaction graph, which specifies the number of agents and all the possible interactions among them. There have been no studies on the automated generation of interaction graphs, which is an important problem because not only is the interaction graph often unspecified in multi-agent applications, but the choice of interaction graph is often a design decision that needs to be optimized as well. For example, the placement of sensors (agents) to form a sensor network (interaction graph) is often unspecified in a decentralized tracking application, and the choice of interaction graph can directly affect the expected rewards of joint policies computed for that problem. In this paper, we address this gap and introduce three algorithms to automatically generate interaction graphs. These algorithms increase the size and density of the interaction graph only when the increase is believed to be beneficial. We also show how one can calculate lower and upper bounds on the expected reward of an optimal joint policy across all possible interaction graphs.

2 Motivating Example: Sensor Placement

We motivate the work in this paper with sensor network problems, where multiple sensors need to coordinate with each other to track non-adversarial targets that are moving in an area. Examples of such problems include the tracking of vehicle movements [Lesser and Corkill, 1983] and the tracking of weather phenomena such as tornadoes [Krainin *et al.*, 2007]. We assume that the sensors are immobile and at least two sensors are required to scan a potential target location. We also assume that the interaction graph, that is, the number of sensors, their locations and their possible interactions, is not given and, thus, the generation of the interaction graph is

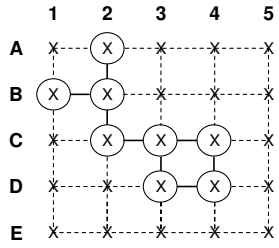

Figure 1: Example Sensor Network

part of the problem. However, the possible sensor placement locations are given and each sensor can only interact with its neighboring sensor to scan the location between them. Lastly, we assume that we have an insufficient number of sensors to place at every location. (The interaction graph is otherwise trivial.) The objective is to place the sensors and find their joint policy such that the expected reward of that joint policy is maximized. For example, Figure 1 illustrates a sensor network, where the crosses denote the 25 possible sensor placement locations, the circles denote the 8 placed sensor locations and the lines denote the possible target locations. The circles and solid lines form nodes and edges of the interaction graph, respectively.

3 Sparse-Interaction Models

We now describe the VF framework and ND-POMDP model, which we use to represent the problems in this paper.

3.1 Value Factorization Framework

The *Value Factorization* (VF) framework assumes that each joint state s can be factored such that $s = (s^1, \ldots, s^m)$, which is true in several multi-agent planning models such as TI-Dec-MDPs [Becker *et al.*, 2004], ND-POMDPs [Nair *et al.*, 2005] and TD-POMDPs [Witwicki and Durfee, 2010]. Without making further (conditional independence) assumptions on the problem structure, a general Dec-POMDP requires exact inference in the full corresponding (finite-time) DBNs, which would be exponential in the number of state variables and agents. The value factorization approach relies on a general, simplifying property of agent interaction, which can be shown to be consistent with many of the existing multi-agent planning models [Kumar *et al.*, 2011; Witwicki and Durfee, 2011]. We next summarize key ideas behind this framework.

Given a Dec-(PO)MDP defined by the set of agents N, joint states S and joint actions A, a *value factor* f defines a subset of agents $N_f \subseteq N$, joint states $S_f \subseteq S$, and joint actions $A_f \subseteq A$. A multi-agent planning problem satisfies *value factorization* if the joint-policy value function can be decomposed into a sum over value factors:

$$V(s, \theta) = \sum_{f \in F} V_f(s^f, \theta^f) \,, \tag{1}$$

where F is a set of value factors, $\theta^f \equiv \theta^{N_f}$ is the collection of parameters of the agents of factor f, and $s^f \equiv s^{S_f}$ is the collection of state variables of this factor.

Even when the value factorization property holds, planning in such models is still highly coupled because factors may overlap. That is, an agent can appear in multiple factors as state variables. Therefore, a value factor cannot be optimized *independently*. But, it has been shown that such structured agent interactions lead to tractable planning algorithms [Kumar *et al.*, 2011]. Such additive value functions have also been used to solve large factored MDPs [Koller and Parr, 1999].

3.2 Networked Distributed POMDPs

For concrete illustrations and evaluation of the results in this paper, we use *Network Distributed POMDPs* (ND-POMDPs) – one of the most commonly used model that satisfies the value factorization property. Formally, an ND-POMDP is defined as a tuple $\langle S, A, \Omega, P, O, R, b, H \rangle$, where

- S is the set of joint states. $S = \times_{1 \le i \le n} S_i \times S_u$, where S_i is the set of local states of agent i and S_u is the set of uncontrollable states that are independent of the actions of the agents. Each joint state $s \in S$ is defined by $\langle s_u, s_1, \ldots, s_n \rangle$, where $s_i \in S_i$ and $s_u \in S_u$.
- A is the set of joint actions. $A = \times_{1 \le i \le n} A_i$, where A_i is the set of actions of agent i. Each joint action $a \in A$ is defined by $\langle a_1, \ldots, a_n \rangle$, where $a_i \in A_i$.
- Ω is the set of joint observations. $\Omega = \times_{1 \le i \le n} \Omega_i$, where Ω_i is the set of observations of agent i. Each joint observation $\omega \in \Omega$ is defined by $\langle \omega_1, \ldots, \omega_n \rangle$, where $\omega_i \in \Omega_i$.
- P is the set of joint transition probabilities that assume conditional transition independence. $P = \times_{s' \in S, s \in S, a \in A} P(s'|s,a)$, where $P(s'|s,a) = P_u(s'_u|s_u) \cdot \Pi_{1 \le i \le n} P_i(s'_i|s_i, s_u, a_i)$ is the probability of transitioning to joint state s' after taking joint action a in joint state s.
- O is the set of joint observation probabilities that assume conditional observation independence. $O = \times_{\omega \in \Omega, s \in S, a \in A} O(\omega|s,a)$, where $O(\omega|s,a) = \Pi_{1 \le i \le n} O_i(\omega_i|s_i, s_u, a_i)$ is the probability of jointly observing ω after taking joint action a in joint state s.
- R is the set of joint reward functions that are decomposable among the agent subgroups $e = \{e_1, \ldots, e_k\}$. $R = \times_{s \in S, a \in A} R(s,a)$, where $R(s,a) = \Sigma_e R_e(s_e, s_u, a_e)$ is the reward of taking joint action a in joint state s. $R_e(s_e, s_u, a_e)$ is the reward of taking joint action a_e, which is defined by $\langle a_{e_1}, \ldots, a_{e_k} \rangle$, in joint states s_e, which is defined by $\langle s_{e_1}, \ldots, s_{e_k} \rangle$, and s_u. a_{e_i} and s_{e_i} is the action and state of agent $e_i \in e$, respectively.
- b is the belief over the initial joint state. $b = \times_{s \in S} b(s)$, where $b(s) = b(s_u) \cdot \Pi_{1 \le i \le n} b(s_i)$ is the belief for joint state s.
- H is the horizon of the problem. In this paper, we address finite-horizon problems.

ND-POMDPs can be represented within the VF framework by associating one value factor with each agent subgroup e. In our sensor network problem, each agent subgroup corresponds to an edge in the graph.

4 Automated Interaction Graph Generation

All the existing value-factored Dec-POMDP algorithms assume that the interaction graph of the problem is given. How-

ever, in many multi-agent applications the interaction graph is initially unknown. Moreover, the choice of interaction graph is an important design decision that affects the quality of the best plan the agents can execute. For example, when there is an insufficient number of sensors to place in every possible location, the placement of sensors to form a sensor network can directly affect the expected rewards of joint policies computed for that problem. Additionally, the choice of interaction graph can also affect the time and space complexities of ND-POMDP algorithms. For example, the time and space complexities of CBDP, a current state-of-the-art ND-POMDP algorithm, are exponential in the induced width of the interaction graph [Kumar and Zilberstein, 2009]. Therefore, in this paper, we formalize the problem of finding an optimal interaction graph and introduce three algorithms to automatically generate interaction graphs based on contribution estimates of edges to the expected reward: two greedy algorithms and a mixed integer linear programming-based algorithm.

4.1 Problem Statement

Given a fixed number of available homogeneous agents and a set of feasible value factors (which is the set of all feasible edges in our sensor network) and their transition, observation and reward functions, a solution is a subset of value factors (which together define an interaction graph and accompanying Dec-POMDP) that satisfy some feasibility constraints (e.g., the number of agents involved in the value factors is no more than the number of available agents). The quality of a solution is the expected reward of an optimal Dec-POMDP policy for that solution. An optimal solution is a solution with the best quality. While our approaches only apply to problems with homogeneous agents, they can be extended to work with heterogeneous agents with additional constraints in the MILPs.

4.2 Greedy Algorithm

Since not all the candidate edges that can be added to the interaction graph are equally important, a natural starting point would be to consider a greedy algorithm that generates an interaction graph by incrementally adding edges based on their contribution, or estimates of their contributions, to the expected reward. This idea is similar to how the algorithm presented in [Krause *et al.*, 2008] incrementally places sensors based on the additional information that they provide about the unsensed locations. We now introduce two variants of this greedy algorithm.

Naive Greedy Algorithm: This algorithm greedily adds value factors based on their actual contribution to the expected reward. In each iteration, it repeatedly loops over all candidate value factors, that is, unchosen value factors that does not require more agents than available; computes a joint policy with each candidate value factor when it is added to the interaction graph; and chooses the value factor with the largest positive gain in expected reward to be added to the interaction graph. This process continues until no candidate value factor results in a positive gain or there are no remaining candidate value factors to consider.

Heuristic Greedy Algorithm: The computation of the joint policy for each candidate value factor can be inefficient. For example, the time and space complexities of a current state-of-the-art ND-POMDP algorithm is exponential in the induced width of the interaction graph [Kumar and Zilberstein, 2009]. Thus, we also introduce a variant of the greedy algorithm that uses contribution estimates of each value factor to the expected reward to greedily select value factors.

We estimate the contribution of each value factor as the sum of expected joint rewards gained by agents defined for that value factor from coordinating with each other across all time steps. More precisely, we calculate the estimated contribution w_f of each value factor f:

$$w_f = \sum_{t=1}^{H} \max_{\vec{a} \in A_f} w_f^{\vec{a},t} \tag{2}$$

$$w_f^{\vec{a},t} = \sum_{s \in S} b'_f(s,t) \cdot R_f(s,\vec{a}) \tag{3}$$

where, for each value factor f, A_f is the set of joint actions, $b'_f(s,t)$ is the belief at state s and time step t calculated using MDP-based sampling starting from the initial belief b [Szer and Charpillet, 2006], and $R_f(s,\vec{a})$ is the reward given state s and joint action $\vec{a}$. For ND-POMDPs, value factors correspond to edges. Thus, the equations are:

$$w_e = \sum_{t=1}^{H} \max_{\vec{a} \in A_e} w_e^{\vec{a},t} \tag{4}$$

$$w_e^{\vec{a},t} = \sum_{s \in S} b'_e(s,t) \cdot R_e(s,\vec{a}) \tag{5}$$

We expect this version of the greedy algorithm to run significantly faster since it only needs to compute the joint policy *once* at the end of the algorithm instead of *each time* it evaluates a candidate value factor.

4.3 MILP-based Algorithm

While the Heuristic Greedy algorithm can efficiently generate interaction graphs, it uses heuristic values that assume that an agent involved in multiple VFs can take different actions for each VF, which is an incorrect assumption. Thus, we introduce a mixed integer linear program (MILP) that assumes that each agent must choose the same action for all VFs that it is involved in. This MILP finds an *open-loop* joint policy[1] that is optimal across all possible interaction graphs and returns the interaction graph that joint policy is operating on. Figure 2 shows the MILP, where

- B is the maximum number of available agents. It is an input parameter.
- $w_f^{\vec{a},t}$ is the normalized estimate of the expected joint rewards of agents in value factor f taking joint action $\vec{a}$ at time step t. It is the same input parameter described earlier in Eq. 3, except that it is now normalized.
- n_i is a Boolean variable indicating if agent i is chosen for the interaction graph (Line 9). The constraint on Line 2

[1] An open-loop joint policy is a policy that is independent of agent observations. In other words, the joint policy is a sequence of H actions for each agent, where H is the horizon of the problem.

Maximize $\sum_{t,f,\vec{a}\in A_f} obj_f^{\vec{a},t}$	**subject to**	(Line 1)
$\sum_i n_i \leq B$		(Line 2)
$fac_f \leq n_i$	$\forall\, f, i \in f$	(Line 3)
$\sum_{a\in A_i} act_i^{a,t} \leq 1$	$\forall\, t, i$	(Line 4)
$act_f^{\vec{a},t} \leq fac_f$	$\forall\, t, f, \vec{a} \in A_f$	(Line 5)
$act_f^{\vec{a},t} \leq act_i^{a,t}$	$\forall\, t, f, i \in f,$ $\vec{a} \in A_f, a \in A_i \cap \vec{a}$	(Line 6)
$obj_f^{\vec{a},t} \leq act_f^{\vec{a},t}$	$\forall\, t, f, \vec{a} \in A_f$	(Line 7)
$obj_f^{\vec{a},t} \leq w_f^{\vec{a},t}$	$\forall\, t, f, \vec{a} \in A_f$	(Line 8)
$n_i, act_i^{a,t} \in \{0,1\}$	$\forall\, t, i, a \in A_i$	(Line 9)
$fac_f, act_f^{\vec{a},t} \in \{0,1\}$	$\forall\, t, f, \vec{a} \in A_f$	(Line 10)
$obj_f^{\vec{a},t} \in [0,1]$	$\forall\, t, f, \vec{a} \in A_f$	(Line 11)

Figure 2: MILP for the VF Framework

ensures that the number of agents chosen does not exceed the maximum number of available agents.

fac_f is a Boolean variable indicating if value factor f is chosen for the interaction graph (Line 10). The constraint on Line 3 ensures a value factor is chosen only if all agents involved in that factor are chosen.

$act_i^{a,t}$ is a Boolean variable indicating if agent i is taking action a at time step t (Line 9). The constraint on Line 4 ensures that an agent can take at most one action in each time step.

$act_f^{\vec{a},t}$ is a Boolean variable indicating if all the agents involved in value factor f are taking joint action $\vec{a}$ at time step t (Line 10). Note that we use the vector notation to represent joint actions and the regular notation to represent individual actions. The constraint on Line 5 ensures that the joint action $\vec{a} \in A_f$ is taken only if value factor f is chosen and the constraint on Line 6 ensures that the joint action $\vec{a}$ is taken only if all the individual actions $a \in \vec{a}$ are taken.

$obj_f^{\vec{a},t}$ is the objective variable whose sum is maximized by the MILP (Lines 1 and 11). The constraints on Lines 7 and 8 ensure that $obj_f^{\vec{a},t}$ equals $w_f^{\vec{a},t}$ if $act_f^{\vec{a},t}$ is 1 and equals 0 otherwise. Thus, maximizing the objective variables over all time steps, value factors and joint actions maximizes the expected reward of the open-loop joint policy.

Once we solve the MILP, the interaction graph is formed by exactly those agents i and value factors f whose Boolean variables n_i and fac_f, respectively, equals 1. One can then use it to compute a *closed-loop* joint policy.[2]

Optimizations for ND-POMDPs: For ND-POMDPs, a positive reward is typically obtained for only *one* joint action in each edge (value factor). For example, in our sensor network problem, a reward is obtained only if two agents sharing an edge coordinates with each other to scan the area between them (denoted by that edge). If either agent does not scan that area, then neither agent gets any reward. As such, one can optimize the general MILP for the VF framework to a specialized MILP for ND-POMDPs by removing the superscripts $\vec{a}$ (since each edge maps to exactly one useful joint action) resulting in a significant reduction in the number of variables and constraints.

[2] A closed-loop joint policy is a regular Dec-POMDP joint policy that is dependent on agent observations.

Maximize $\sum_{t,i,j} obj_{ij}^t$	**subject to**	(Line 12)
$\sum_i n_i \leq B$		(Line 13)
$fac_{ij} \leq n_i$	$fac_{ij} \leq n_j$	(Line 14)
$fac_{ij} = fac_{ji}$	$act_{ij}^t = act_{ji}^t$	(Line 15)
$\sum_i act_{ij}^t \leq 1$	$act_{ij}^t \leq fac_{ji}$	(Line 16)
$act_{ij}^t = 0$ for all i and j that are not neighbors		(Line 17)
$obj_{ij}^t \leq act_{ij}^t$	$obj_{ij}^t \leq w_{ij}^t$	(Line 18)
$n_i \in \{0,1\}$	$fac_{ij} \in \{0,1\}$	(Line 19)
$act_{ij}^t \in \{0,1\}$	$obj_{ij}^t \in [0,1]$	(Line 20)

Figure 3: MILP for ND-POMDPs

Figure 3 shows this MILP, where subscripts i and j denote agent IDs and ij denotes the edge between agents i and j. The constraints on Lines 13 and 14 correspond to those on Lines 2 and 3, respectively. The constraints on Line 15 are to ensure symmetry. The constraints on Lines 16 and 17 correspond to those on Lines 4 and 5, and the constraints on Line 18 correspond to those on Lines 7 and 8.

Complexity: The MILP requires $O(H \cdot n \cdot |\hat{A}_i|)$ variables and $O(H \cdot |F| \cdot |\hat{f}| \cdot |\hat{A}_f| \cdot |\hat{A}_i|)$ constraints, where H is the horizon, n is the number of agents, $|\hat{A}_i|$ is the maximum number of actions per agent, $|F|$ is the number of value factors, $|\hat{f}|$ is the maximum number of agents in a value factor and $|\hat{A}_f|$ is the maximum number of joint actions in a value factor. The dominating terms for the number of variables and constraints are the number of $act_f^{a,t}$ variables and the constraints on Line 6, respectively.

While the number of constraints might appear intractably large, bear in mind that one of the main motivations of sparse-interaction models is that the number of agents and joint actions in a value factor is *small* such that they can be exploited for scalability.

4.4 Bounds

Lower Bounds: The open-loop joint policy found by the MILP can be extracted from its result by taking the union of all actions a of agent i at time step t whose Boolean variable $act_i^{a,t}$ equals 1. One can then evaluate that policy to get an expected reward, which will form a lower bound on the optimal expected reward. For ND-POMDPs, evaluating an open-loop joint policy π to get the expected reward $V^\pi(b)$ for the initial belief b can be done:

$$V^{\pi}(b) = \sum_{s \in S} b(s) \cdot \sum_{e \in E} V_e^{\pi}(s, 0) \quad (6)$$

$$V_e^{\pi}(s,t) = R_e(s, \pi(t)) + \sum_{s' \in S} P(s'|s, \pi(t)) \cdot \sum_{\omega \in \Omega} O(\omega|s', \pi(t)) \cdot V_e^{\pi}(s', t+1) \quad (7)$$

$$= R_e(s, \pi(t)) + \sum_{s' \in S} P(s'|s, \pi(t)) \cdot V_e^{\pi}(s', t+1) \quad (8)$$

where E is the set of edges in the interaction graph and $\pi(t)$ is the joint action of the agents at time step t according to joint policy π. Eq. 7 simplifies to Eq. 8 because π is independent of the observations received.

Upper Bounds: To obtain an upper bound on the optimal expected reward, one can solve the underlying MDP for each possible interaction graph and take the largest expected reward. For ND-POMDPs with a given interaction graph, one can calculate the upper bound $V(b)$ for initial belief b using the expected rewards $V(s)$ for starting states s:

$$V(b) = \sum_{s \in S} b(s) \cdot V(s) \quad (9)$$

$$V(s) = \max_{a \in A} \Big\{ R(s,a) + \sum_{s' \in S} P(s'|s,a) \cdot V(s') \Big\} \quad (10)$$

$$= \max_{a \in A} \Big\{ \sum_{e \in E} R_e(s,a) \Big\} + \sum_{s' \in S} P(s'|s) \cdot V(s') \quad (11)$$

where E is the set of edges in the given interaction graph. Eq. 10 simplifies to Eq. 11 because all the states in our problem are uncontrollable states that are independent of the action of agents. If not all states are uncontrollable states, then one can still calculate an upper bound on the expected reward of an optimal joint policy for the MDP [Kumar and Zilberstein, 2009] or use existing techniques to solve factored MDPs [Guestrin *et al.*, 2001].

5 Experimental Evaluation

We compare the greedy and MILP-based algorithms on a problem with 25 possible sensor placement locations arranged in a 5×5 grid as in Figure 1. We model the problem within the VF framework as an ND-POMDP, varying the maximum number of available sensors from 5 to 15 to investigate the scalability of the algorithms. The number of targets to track is 2. To simulate problems with both known and unknown target trajectories, we experiment with two types of trajectories: fixed trajectories, where each target can stay at its current location with probability 0.1 and move to the next location in its trajectory with probability 0.9, and random trajectories, where each target can stay at its current location or move to any neighboring location with equal probability.

We use CBDP [Kumar and Zilberstein, 2009], a state-of-the-art ND-POMDP algorithm, as a subroutine in the greedy and MILP-based algorithms to compute joint policies. We set the horizon to 10 and the number of samples to 5. We ran the experiments on a quad core machine with 16GB of RAM and 3.1GHz CPU, and set a cut-off time limit of 40 hours. Figures 4 and 5 show the results for problems with random and fixed target trajectories, respectively, and Table 1 shows the size of interaction graphs built.

(a) Random Trajectories

Max Sensors	Naive Greedy	Heuristic Greedy	MILP
5	4.0	4.0	4.0
7	6.0	7.0	6.0
9	8.5	10.0	8.5
11	10.0	13.5	11.0
13	10.0	17.0	11.5
15	11.0	18.5	13.0

(b) Fixed Trajectories

Max Sensors	Naive Greedy	Heuristic Greedy	MILP
5	4.0	4.0	5.0
7	7.0	7.0	8.0
9	9.0	10.0	10.0
11	12.5	12.5	12.0
13	15.5	14.5	14.5
15	16.5	16.0	16.0

Table 1: Number of Edges in the Interaction Graphs

Runtime: The runtimes are similar for both problem types. The only exception is that the upper bound computations are one order of magnitude slower on random trajectory problems than on fixed trajectory problems. This result is to be expected since there is a larger number of state transitions with non-zero probabilities in random trajectory problems and, as such, more computation is necessary.

For both problem types, Naive Greedy runs up to one order of magnitude longer than the Heuristic Greedy and MILP-based algorithms. The reason is that Naive Greedy runs CBDP multiple times each time it adds an edge to the interaction graph. (It runs CBDP each time it evaluates a candidate edge.) On the other hand, the Heuristic Greedy and MILP-based algorithms run CBDP only once to compute the joint policy for its interaction graph.

For random trajectory problems, the MILP-based algorithm is faster than the Heuristic Greedy algorithm except for problems that are very small (problems with 5 sensors). The reason for this behavior is the following: In these problems, there is a large number of edges with non-zero probabilities that a target will be at those edges since the targets are performing random walks. Additionally, the Heuristic Greedy algorithm uses heuristic values (in Eq. 4) that assume that the sensors at locations connected by an edge *will* coordinate with each other to get the reward at that edge (since we are taking the maximum over all joint actions in that edge). Therefore, the algorithm will add a large number of edges with non-zero probabilities to its interaction graph as long as adding those edges do not require more sensors than available.

On the other hand, the MILP-based algorithm takes into account the fact that sensors can only coordinate with at most one neighboring sensor at each time step in choosing their edges. (Each agent must have the same action for all value factors.) Thus, the number of edges in the MILP interaction graph is smaller than that in the Greedy Heuristic interaction graph, as shown in Table 1(a). Thus, the runtime of CBDP on the MILP interaction graph is also smaller that on the Greedy Heuristic interaction graph.

For fixed trajectory problems, the Heuristic Greedy algorithm is faster than the MILP-based algorithm. The reason

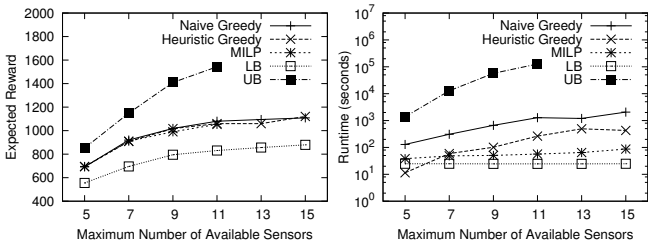

Figure 4: Results for Random Trajectories

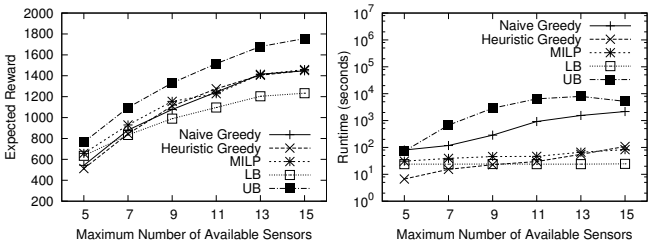

Figure 5: Results for Fixed Trajectories

for this behavior is the following: In these problems, there is a small number of edges with non-zero probabilities that a target will be at those edges since the targets are always in one of the edges in their respective trajectories. Therefore, the Heuristic Greedy algorithm will add a small number of edges to its interaction graph. The MILP-based algorithm is also able to exploit this property and only include a small number of edges to its interaction graph. Thus, the number of edges in the MILP interaction graph is similar to that in the Greedy Heuristic interaction graph, as shown in Table 1(b). Thus, the runtime of CBDP on both interaction graphs are also similar.

However, on the computational effort in generating the interaction graph, the Heuristic Greedy algorithm is more efficient than the MILP-based algorithm – Heuristic Greedy takes approximately 0.5s while the MILP-based algorithm takes approximately 25s. This difference is more noticeable when the problems are small (as the runtimes of CBDP are small) and diminishes as the problems become larger.

Solution Quality: The expected rewards of joint policies found for random trajectory problems are smaller than those found for fixed trajectory problems, which is to be expected since there is a larger entropy in random trajectory problems.

For both problem types, all three algorithms find comparable joint policies except for problems with 5 maximum available sensors. In these problems, both greedy algorithms found joint policies with expected rewards that are about 20% smaller than the expected rewards of joint policies found by the MILP-based algorithm. The reason is that the first two edges chosen by the greedy algorithm typically correspond to the starting edges (locations) of the two targets. These edges are typically disjoint, and four sensors are thus placed just to track the targets along these two edges. On the other hand, the MILP-based algorithm typically places the first four sensors more efficiently by placing them in a square, and they can thus track targets along the four edges of the square. Thus, the MILP joint policies found by the MILP-based algorithm have larger expected rewards than those found by the greedy algorithm. Table 1(b) shows this behavior, where the MILP interaction graph has 25% more edges than the greedy interaction graphs on problems with 5 sensors.

For both problem types, the lower bounds are at most 20% smaller than the expected rewards of the joint policies found by the MILP-based algorithm, which are at most 35% smaller than the upper bounds. The lower bounds have smaller expected rewards since they are expected rewards of open-loop joint policies. The upper bounds have larger expected rewards since they are expected rewards of joint policies on fully observable problems. These bounds are tighter in fixed trajectory problems than in random trajectory problems since fixed trajectory problems are simpler.

Lastly, we also exhaustively enumerated all possible interaction graphs for small problems where it is possible to do so, and in all of those cases, both the optimal and MILP-based graphs are *very* similar.

In summary, our experimental results show that the Heuristic Greedy and MILP-based algorithms are good ways to automatically generate interaction graphs and find joint policies that are within reasonable error bounds. The Heuristic Greedy algorithm is better suited for problems with little transition uncertainty and the MILP-based algorithm is better suited for problems with more transition uncertainty. Furthermore, one can use the *open-loop* joint policies if there is an insufficient amount of time to compute better *closed-loop* joint policies.

6 Conclusions

The VF framework has been shown to be a general framework that subsumes many sparse-interaction Dec-POMDP models including ND-POMDPs. Existing algorithms for these models assume that the interaction graph of the problem is given. Thus far, there have been no studies on the automated generation of interaction graphs. In this paper, we introduced two greedy algorithms and a MILP-based algorithm to automatically generate interaction graphs and establish lower and upper bounds on the expected reward of an optimal joint policy. The greedy algorithms incrementally add value factors (or edges) to the interaction graph based on their actual or estimated contribution to the expected reward. The MILP-based algorithm generates an interaction graph by choosing value factors to form an interaction graph such that an optimal open-loop joint policy on that interaction graph is optimal across all possible interaction graphs.

Our experimental results show that these methods produce reasonable joint policies (their expected rewards are at least 65% of a loose upper bound). The Heuristic Greedy algorithm is faster than the MILP-based algorithm in problems with less transition uncertainty and vice versa in problems with more transition uncertainty. In sum, we examined the challenge of automatically generating interaction graphs and offered several general methods that performed well in a sensor network coordination testbed. These methods offer a foundation for further exploration of this area.

Acknowledgments

This work was funded in part by the National Science Foundation Grant IIS-1116917.

References

[Becker *et al.*, 2004] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.

[Bernstein *et al.*, 2002] Daniel Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[Guestrin *et al.*, 2001] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1523–1530, 2001.

[Koller and Parr, 1999] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1339, 1999.

[Krainin *et al.*, 2007] Michael Krainin, Bo An, and Victor Lesser. An application of automated negotiation to distributed task allocation. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pages 138–145, 2007.

[Krause *et al.*, 2008] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

[Kumar and Zilberstein, 2009] Akshat Kumar and Shlomo Zilberstein. Constraint-based dynamic programming for decentralized POMDPs with structured interactions. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 561–568, 2009.

[Kumar *et al.*, 2011] Akshat Kumar, Shlomo Zilberstein, and Marc Toussaint. Scalable multiagent planning using probabilistic inference. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2140–2146, 2011.

[Lesser and Corkill, 1983] Victor Lesser and Daniel Corkill. The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, 1983.

[Nair *et al.*, 2005] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 133–139, 2005.

[Oliehoek *et al.*, 2008] Frans Oliehoek, Matthijs Spaan, Shimon Whiteson, and Nikos Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 517–524, 2008.

[Oliehoek, 2010] Frans Oliehoek. *Value-Based Planning for Teams of Agents in Stochastic Partially Observable Environments*. PhD thesis, University of Amsterdam, Amsterdam (Netherlands), 2010.

[Rabinovich *et al.*, 2003] Zinovi Rabinovich, Claudia Goldman, and Jeffrey Rosenschein. The complexity of multiagent systems: The price of silence. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1102–1103, 2003.

[Szer and Charpillet, 2006] Daniel Szer and Francois Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1233–1238, 2006.

[Velagapudi *et al.*, 2011] Prasanna Velagapudi, Pradeep Varakantham, Paul Scerri, and Katia Sycara. Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 955–962, 2011.

[Witwicki and Durfee, 2010] Stefan Witwicki and Edmund Durfee. From policies to influences: A framework for nonlocal abstraction in transition-dependent Dec-POMDP agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1397–1398, 2010.

[Witwicki and Durfee, 2011] Stefan Witwicki and Edmund Durfee. Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 29–36, 2011.