Auton Agent Multi-Agent Syst (2007) 15:29–45
DOI 10.1007/s10458-006-0007-x

Local strategy learning in networked multi-agent team formation

Blazej Bulka · **Matthew Gaston** · **Marie desJardins**

Published online: 13 May 2006
Springer Science+Business Media, LLC 2006

Abstract Networked multi-agent systems are comprised of many autonomous yet interdependent agents situated in a virtual social network. Two examples of such systems are supply chain networks and sensor networks. A common challenge in many networked multi-agent systems is decentralized team formation among the spatially and logically extended agents. Even in cooperative multi-agent systems, efficient team formation is made difficult by the limited local information available to the individual agents. We present a model of distributed multi-agent team formation in networked multi-agent systems, describe a policy learning framework for joining teams based on local information, and give empirical results on improving team formation performance. In particular, we show that local policy learning from limited information leads to a significant increase in organizational team formation performance compared to a random policy.

Keywords Multi-agent learning · Networked multi-agent systems · Agent learning and adaptivity · Probabilistic reasoning

1. Introduction

Distributed team formation is a common challenge in many multi-agent systems. In sensor networks, collections of sensors, sector managers, and other agents must dynamically decide to work together to track targets or adapt to environmental changes [4, 12]. In supply networks, manufacturers, suppliers, distributors, wholesalers,

B. Bulka (✉)· M. Gaston · M. desJardins
Department of Computer Science,
University of Maryland, Baltimore County
1000 Hilltop Circle,
Baltimore, MD 21250, USA
e-mail: bulka1@umbc.edu

M. Gaston
e-mail: mgasto1@cs.umbc.edu

M. desJardins
e-mail: mariedj@cs.umbc.edu

and consumers form informal teams in order to efficiently distribute goods and to adapt to supply and demand [26, 23]. The process of distributed team formation involves agents autonomously deciding to cooperate with other agents in order to accomplish joint tasks.

Several factors compound the challenge of designing agents for efficient distributed team formation. In large agent organizations, the agent-to-agent interactions may be limited by a social network. The social network may result from a variety of factors, including physical proximity, communication limitations, limited knowledge of other agents and their capabilities, trust relationships, and organizational structures. Dynamic, real-time task environments, where the agents have limited information about the structure of future tasks, make the design of effective team joining strategies difficult. Finally, limited local information as a result of the virtual social network and the lack of a centralized controller makes it difficult for agents to determine the behaviors of other agents and to decide which teams to join.

Most of the previous work on agents learning to form teams in multi-agent systems focused on environments with access to global information and no restrictions arising from the social network. Kraus et al. described and evaluated several heuristics for coalition formation [15]. These heuristics involved an agent evaluating all possible coalitions, assuming that the agent could observe all possible tasks and all other agents. Nair et al. formulated the problem of allocating roles among existing teams as a communicating decentralized POMDP [16]. Although it is possible to solve the problem when formulated in this way, it is shown to be NEXP-complete in the case where the agents pay a penalty for communicating with each other [3]. Therefore, optimal role allocation in communication-restricted agents with more than a few agents is intractable. This observation supports our emphasis on the distributed, on-line learning of heuristics for dynamic team formation. Other previous work has focused on learning to select appropriate coalition partners assuming no restrictions imposed by the social network [6] and on learning to select an optimal, predefined coordination method [8].

This paper focuses on the problem of dynamic, distributed team formation in a networked multi-agent system with a real-time task environment. In particular, we develop and empirically test a learning framework within which agents attempt to learn effective team initiating and team joining policies online. The goal is to learn local policies that improve the aggregate performance of the agent organization, measured as the proportion of successfully completed tasks. We find that it is possible for agents to learn effective policies from experience and that locally learned policies can significantly improve organizational performance in a variety of network structures.

Note that we are interested in agents that are self-interested, but not to the point of deliberately interfering with other agents' activities. Specifically, the agents in our environment join teams based solely on the expected utility of doing so. We assume a fixed payoff for successful tasks (and zero payoff for failed tasks), so these agents can join teams based on the estimated (learned) probability of that team's success. We refer to this behavior as *locally self-interested*.[1]

[1] In contrast with our "locally self-interested" agents, a genuinely cooperative agent might exhibit sacrificial behavior: for example, it might forgo joining a high-probability team if another agent can fill that role equally well, and instead join a lower-probability team for which the agent is the only hope of success. On the other hand, a genuinely malicious agent might exhibit hostile behavior: for example, it might deliberately join a team knowing that doing so will eliminate any possibility of that team succeeding. Our framework does not account for such behaviors, which could be interpreted as rational, but would require a more complex utility model.

2. A model of multi-agent team formation

To explore simultaneous learning of team joining policies in a dynamic environment, we have selected a simple multi-agent system model motivated by previous work on agent team formation [1, 10, 15, 16]. The model provides a dynamic team formation environment in which agent teams form spontaneously in a completely decentralized manner and the agents' decision making is based solely on local information. In addition, the model allows for potentially very large agent organizations where the agents are embedded in a social network. The model is only concerned with the dynamic formation of teams and not with post-formation teamwork mechanisms or protocols, for which there is a large body of previous work [13, 17, 20, 25].

In our model, tasks are generated periodically and are globally advertised to the organization. A *task* is modeled as a set of required skills that a group of agents must provide over a specified duration of time, before a specified deadline. These tasks are simple abstractions of real-world tasks such as a coordinated rescue operation in an emergency response domain, a group of sensors tracking a vehicle in a sensor network domain, or a set of distributor agents in a supply chain domain.

Agents attempt to form teams to accomplish the advertised tasks. The agents in the organization are embedded in a virtual social network that restricts the set of possible agent teams: specifically, for an agent to be on a team, the agent must have a social connection (i.e., a shared edge in the social network) with at least one other agent on the team. Since we are only concerned with the formation process, tasks are generic in that they only require that a team of agents with the necessary skills form to accomplish the specific task.

In this model of team formation, the organization consists of N agents, $A = \{a_1, a_2, \ldots, a_N\}$, where each agent can be considered as a unique node in a social network. The social network is modeled by an adjacency matrix E, where an element of the adjacency matrix $e_{ij} = 1$ if there is an edge between agent a_i and a_j and $e_{ij} = 0$ otherwise. The social relationships among the agents are undirected, or symmetric, so $e_{ij} = e_{ji}$. In the agent organization, each agent is also assigned a single fixed skill, $\sigma_i \in [1, \sigma]$, where σ is the number of different types of skills that are present in the organization. In the experiments presented in this paper, skills are assigned to each agent uniformly at random from the set of potential skills.

During the team formation process, each agent can be in one of three states: UNCOMMITTED, COMMITTED, or ACTIVE. An agent in the UNCOMMITTED state is available and not assigned to any task. An agent in the COMMITTED state has selected a task, but the full team to work on the task has not yet formed. Finally, an agent in the ACTIVE state is a member of a team that has fulfilled all of the skill requirements for a task and is actively working on that task. Only uncommitted agents can commit to a new or partially filled task. In the case when multiple uncommitted agents compete to satisfy a skill in the task, only one of them, selected at random, will become COMMITTED and the rest will remain UNCOMMITTED. Committed agents cannot decommit from a given task. Upon task completion, agents in the ACTIVE state return to the UNCOMMITTED state. An agent's state is denoted by s_i.

Tasks are introduced at fixed task introduction intervals, where the length of the interval between tasks is given by the model parameter μ. Tasks are globally advertised (i.e., announced to all agents). Each task T_k has an associated size requirement, $|T_k|$, and a $|T_k|$-dimensional vector of required skills, R_{T_k}. The skills required for a given task T_k are chosen uniformly with replacement from $[1, \sigma]$. Each task is advertised for a finite number of time steps γ, ensuring that the resources (i.e., agents) committed to the tasks are freed if the full requirements of the task cannot be met. That is, if a task is advertised for more than γ time steps without a full team forming, the agents committed to the task return to the UNCOMMITTED state. Similarly, teams that successfully form to fill the requirements of a given task are only active for a finite number of time steps α.

Agent social networks. Since we are primarily interested in large agent organizations where the control of the agents does not necessarily fall under a single authority, we explicitly model the agent social network. Real-world constraints such as cognitive limitations, geographical considerations, and limited communication capabilities make it impossible for an individual agent to know about or communicate with *all* other agents. In our team formation model, the agent social network explicitly restricts the sets of agents that can form teams.

Definition 1 A valid team is a set of agents $M = \{a_i\}$ that induce a connected subgraph of the agent social network and whose skill set $\{\sigma_i\}$ fulfills the skill requirements for task T_k.

The requirement of a team to induce a connected subgraph of the agent social network means that for some agent in the team, $a_i \in M$, there must exist at least one other agent, $a_j \in M, i \neq j$, such that $e_{ij} = 1$. An agent is therefore considered to be *eligible* to commit to a task in two situations: (1) when no other agents are committed to the task or (2) when at least one neighbor of the agent is already committed to the task. In essence, this requires that an agent on a team must "know" at least one other agent on the team.

Previous work on networked multi-agent team formation has demonstrated that the ability of an organization of agents to effectively form teams depends heavily on the structure of the agent social network [9]. In particular, networks with short average path length and hub-like structures support a diverse set of possible teams and, therefore, a large number of task configurations. Given the dependence of team formation performance on network structure, we explored policies for adapting the network structure as a means of improving performance in our earlier work [11]. In this paper, we embed the agents in fixed network structures and focus on the ability of individual agents to learn effective team joining policies.

Team formation policies. During each iteration of the model, the agents receive advertisements about the current tasks in the system. Each agent with s_i= UNCOMMITTED considers all currently advertised tasks for which it is eligible and selects at most one of them. If the selected task has no other agents committed to it, an agent initiates a new team; otherwise, it joins the existing team. We refer to an agent's strategy for selecting teams to initiate as the agent's *team initiation policy*. The strategy for joining existing teams is termed the agent's *team joining policy*.

The policies for initiating and joining teams are an important factor in the performance of multi-agent team formation. There are many possibilities for these policies. One simple policy involves an agent always trying to commit to a task by randomly selecting from all tasks for which the agent is eligible. Such a policy, which combines team initiation and team joining, may create unnecessarily high demand on, or inefficient use of, the resources in the system (e.g., agents may commit to tasks for which it is impossible for a team to form). Another simple policy, which may place less burden on the agents, is a policy in which agents commit to tasks with a given probability that depends in a fixed way on local information, such as the relative percentage of positions filled. We are interested in designing agents that autonomously learn effective team formation policies to improve organizational performance. Our framework for learning team joining and team initiating policies is discussed in Section 3.

In our experiments, we use as a baseline the *random policy*, which selects uniformly from all of the tasks for which the agent is eligible, including a virtual *wait task*. Let E denote the number of open tasks for which an agent is eligible. An agent using the random policy selects from the eligible tasks (including the wait task) with probability

$$P(J_i) = \frac{1}{E+1} \tag{1}$$

where J_i is the act of committing to the ith eligible task. The purpose of the wait task is to prevent agents from always committing to a team at the first opportunity, since doing so may not necessarily be the best option.

We also developed several simple hand-coded, rule-based strategies for over-constrained networked team formation environments as a baseline (e.g., prefer to join a task whose team is almost formed, or always join a team when eligible to do so). Although the random policy is quite simple, we have found it to outperform the hand-coded strategies that we tried.

Organizational performance. We measure the team formation performance of the agent organization as the ratio of number of teams successfully formed to the total number of tasks introduced to the system:

$$\text{org. performance} = \frac{\text{\# of teams successfully formed}}{\text{\# of tasks introduced}} \tag{2}$$

This measure of performance provides a global measure of the effectiveness of the agent organization in forming teams to execute the advertised tasks. In the following section, we describe a framework for individual agents to learn effective team formation policies and discuss the challenges associated with distributed simultaneous learning.

3. Learning to initiate and join teams

Rather than designing specific team formation policies for agents, mechanisms for the individual learning of effective team joining and team initiating policies allow the agents and the organization to deal with variability in the team formation environment, including task structure and network structure (e.g., node failures). Moreover, in many network structures, the agents have diverse patterns of connectivity with the rest of the organization. It is difficult to specify team joining policies for all possible positions in a social network: for example, a hub agent with many connections may need a different policy than a "fringe" agent with only one or two connections. Therefore, learning to effectively join teams from any position in the network is extremely useful.

In order to capture a realistic team formation environment, agents are forced to make decisions based solely on local information. Any access to global data would require either extensive communication among agents or a centralized point in the system. Moreover, relying only on local information reduces the computational burden on the agents.

Some of the key challenges involved in multi-agent learning in a distributed team formation environment include:

- *Decisions based solely on local information*: Each agent must make decisions about joining teams, initiating teams, or waiting given only partial information about the state of the organization. The information is limited by the position of the agent in the social network and by the dynamic nature of the environment. This means that an agent's knowledge of the topology of the network, skills and states of other agents is limited only to its immediate neighbors. Additionally, an agent cannot know what tasks will be introduced to the system in the near future.
- *Local policy interference*: The policies of two nearby agents (e.g., neighbors or neighbors of neighbors) can interfere with each other when the agents attempt to select teams to initiate and join. For example, one agent may learn to join teams only when a specific neighbor is UNCOMMITTED (hoping that the UNCOMMITTED neighbor will subsequently join the team), while the neighbor may learn to avoid teams of which the first agent is a member. Moreover, interference can propagate through the agent organization. We observed this problem to be common in simultaneous interdependent adaptation.
- *Learning to wait*: In the dynamic team formation environment, it may be in an agent's best interest to delay, rather than committing to any task for which the agent is eligible. For example, an agent having very few neighbors available (UNCOMMITTED neighbors) could wait until the other neighbors become UNCOMMITTED, rather than participating in a team that is unlikely to be formed because of an insufficient number of available agents. However, the reward for waiting is necessarily delayed, making it difficult to explicitly learn to wait. Additionally, the value of deciding to wait is dependent upon the actions of an agent's neighbors (and, indirectly, on the rest of the agent network). After all, waiting for the busy neighbors may be futile if the neighbors decline to cooperate.
- *Blocking*: If agents are strictly greedy and attempt to join as many successful teams as possible, blocking may increase. Blocking occurs when an agent is in the COMMITTED or ACTIVE state, preventing teams from forming in that part of the network structure. This phenomenon is similar to a bottleneck in a flow network. Effectively, it may cause the network to fragment into separate pieces. It is desirable for agents to learn to prevent or reduce blocking.
- *Local performance $\neq$ global performance*: If the agents in the organization attempt to naively optimize local performance (i.e., join as many successful teams as possible), global performance is not necessarily optimized. One "greedy" agent can reduce the performance of all of its neighbors, by making poor task choices or "stealing" skill slots on a specific task, resulting in lower collective performance. Alternatively, an agent trying to locally optimize performance may decide only to join teams that are almost a certain team formation success. If such risk-averse behavior becomes common, almost no agent would like to undertake the high-risk operation of establishing a new team. Ultimately, in this "tragedy of the commons" scenario, very few teams may form at all, reducing both global and local performance.

– *Inhibiting learning*: Poor performance in one region of the organization can propagate through the organization. If an agent in a certain part of the organization learns a poor or selfish policy, it can inhibit a neighbor's ability to learn which tasks are good to initiate or join. This can be a result of teams being co-opted or corrupted by the agent with the poor or selfish policy, resulting in failure of teams that would otherwise have a high likelihood of success.

Many of these challenges are interdependent, manifesting themselves simultaneously in the dynamic team formation environment.

Overview of the decision making and learning process. In our learning framework, the agents try to learn effective team joining and team initiating policies in real time, based on their previous experiences. The learned policy is then used to make specific team joining and team initiating decisions based on the current, locally perceived state of the agent's environment.

During each iteration, the agents receive advertisements of tasks for which they are eligible at this time. For each of the tasks, an agent estimates the probability of a task's success p_i, which is defined to be the probability that a full, valid team will be formed before the task's expiration, provided that the agent joins this team. The agents learn to estimate probabilities p_i by using classifiers, which determine how favorable the current conditions are. The details of classifying tasks are described in more detail below.

Once the agent calculates all the probabilities, it considers the option of not committing to any task because the current conditions are deemed unfavorable. Based on the estimated probabilities of success of the available tasks p_i, the agent computes the expected discounted utility of waiting for one more iteration and the expected utility of committing to any of the available tasks in this iteration. Next, the agent chooses one of these two options randomly with probabilities proportional to the utilities of the two options. The details of calculating the utilities are given below.

If an agent decides to commit to any task, rather than waiting, it selects a task randomly from the available tasks. The probability of a task being selected is proportional to its probability of success p_i, as determined by an appropriate classifier. Having selected a task and successfully committing to it, the agent becomes COMMITTED, and will eventually experience the result of its team formation decision: whether a team formed before the task's expiration date or the team formation failed. After receiving the feedback, the agent updates its classifiers and utilities based on the state of its local environment when the commitment decision was made.

Learning probabilities of tasks' success. Whenever an agent receives a task advertisement, it records the current state of its environment, including the state of the team for this task (how many and which slots of the tasks are still unfilled) and the states of its neighbors. Later, when it is known whether the team formed successfully or failed to form within the time limit, this stored state is used to create a training instance. The instance is labeled as JOIN or IGNORE, based on the success or failure of the team, respectively. Subsequently, the learning model is updated using these training instances to influence future team formation decisions. In our approach, the agent only learns probabilities of tasks' success based on the tasks it decided to join. The tasks the agent decided to ignore are not remembered because it is not known whether they would have been successful or not.

A training instance includes the following attributes of an agent's local environment: the states of each of the neighbors; the skills of each of the neighbors; the number of unfilled slots in the task; the skills required for the task; and the amount of time that the task has been advertised. The information about the skills required for a task includes whether each of the R_{T_k} skills is *not required for this task*, *required but already filled*, or *required but not filled*.

In our learning framework, the agents use two classifiers: one for team initiation and another for joining existing teams. The decision to use two classifiers was based on the fact that initiating a team and joining a partially filled team are very different situations. For example, joining a team that already has most of its skill slots filled should have a greater probability of success than joining a team with mostly empty skill slots. However, if all agents only join nearly full teams, then overall organizational performance would suffer, since agents would never initiate new teams.

The knowledge summarized in a classifier may become outdated over time, because the neighboring agents also learn and change their behavior. This creates a need to gradually discard old experience that does not reflect the current situation. We solved this problem by assigning a weight to each training instance. Initially, each instance has a weight of 1.0, but at every iteration afterwards, its weight is decreased by multiplying it by a *decay discount factor* β. This ensures that the most recent experiences have a dominant role in the decision-making process.

We used the WEKA library [28] to provide the actual implementation of the classifiers. Initially, we experimented with a range of classifiers, including support vector machines with kernels (SMO) [19], decision trees (J48) [21], and Lazy Bayesian Rules (LBR) [29]. We selected Naïve Bayes [5] as our classifier of choice because of its ability to be updated with new learning instances in real time, its simplicity, and its performance.

Learning to wait. Having obtained the probabilities of each task's success from the classifiers, an agent has to decide whether the probabilities are high enough at a moment to merit joining a task, or if it is better to wait for a more advantageous situation. In our simulation, agents try to learn a good behavior by using reinforcement learning [22]. They estimate the utility of *join* or *wait* in this iteration by learning a value function $Q(s_{RL}, a_{RL})$, where s_{RL} is one of the states describing the agent's situation, and a_{RL} is one of the two possible actions: joining an available task and waiting.

We define a set of abstract states that correspond to the expected probability that joining a task will be successful. If the agent chooses to join some task, then the tasks are selected at random, proportionally to their estimated probability of success, p_i. Therefore, it is possible to compute the *expected probability of success of a randomly selected task*:

$$Ep = \sum_i \frac{p_i}{\sum_j p_j} p_i = \sum_i \frac{p_i^2}{\sum_j p_j} \tag{3}$$

The expected probability Ep determines the agent's state, which we refer to as s_{RL}. The potential range of values of $Ep[0,1]$ is divided into n_{RL} intervals, where n_{RL} is a parameter set by the designer:

$$\left[0, \frac{1}{n_{RL}}\right], \left(\frac{1}{n_{RL}}, \frac{2}{n_{RL}}\right], \ldots, \left(\frac{n_{RL}-1}{n_{RL}}, 1\right] \tag{4}$$

The value function $Q(s_{RL}, a_{RL})$ is updated when the result of each team joining attempt is known. For a decision to join that was made at time t in state s^t_{RL} (dependent on the value of E_p), an appropriate reward r_{RL} is assigned (1.0 for a team that successfully formed, and 0.0 for a team that failed) to the state s^t_{RL}:

$$Q(s^t_{RL}, join) \leftarrow Q(s^t_{RL}, join) + \alpha_{RL}(r_{RL} - Q(s^t_{RL}, join)), \tag{5}$$

where α_{RL} is a learning rate set by the designer.

Additionally, if the agent was waiting for j iterations prior to the joining decision, the states at iteration $t-1, t-2, \ldots, t-j$ are modified with reward r_{RL}, discounted respectively with the discount factors $\gamma_{RL}, \gamma^2_{RL}, \ldots, \gamma^j_{RL}$:

$$Q(s^{t-i}_{RL}, wait) \leftarrow Q(s^{t-i}_{RL}, wait) + \alpha_{RL}(\gamma^i_{RL} r_{RL} - Q(s^{t-i}_{RL}, wait)). \tag{6}$$

Idleness limit. During a simulation, the agents may perceive the utility of joining as very low, especially during the initial iterations of the simulation, when the initial utility values promote waiting rather than joining. This can also happen after a long series of team joining failures. Because of this, affected agents would not join any teams, would gather no further experience, and a vicious cycle of inexperience would ensue. To avoid this form of "learned helplessness," agents always switch to using the random policy after periods of inactivity. We denote the length of this period of inactivity as the *idleness limit*. If an agent exceeds its idleness limit (i.e., it does not attempt to join a team for a specified amount of time), it is forced to select a task to join at random for that time step.

Filtering negative instances. Since we focus on heavily loaded task environments, most of the agents see far more team failures than successes. If this is not taken into account, the perceived task success probability and joining utilities may drop. As a result, the agents will quickly stop joining any teams (unless forced by the idleness limit) and the performance will degrade below that of the random policy. In order to avoid this difficulty, we use filtering, or sampling, of negative training instances. In our learning framework, negative training instances are filtered if the ratio of positive training instances to negative training instances is too low. Specifically, a negative training instance will be kept with probability p_{acc}, where

$$p_{acc} = \min\left(1.0, 2 \times r \times \frac{\text{\# of positive instances}}{\text{\# of all instances}}\right). \tag{7}$$

The ratio r ($0 \leq r \leq 1$) is a parameter that determines the acceptance rate of negative instances.

4. Results

The goal of our experiments was to determine if it is possible for agents to learn effective team joining and team initiating policies. We also wished to evaluate the ability of the agents to learn *regardless of the topology of the agent social network*. That is, the goal was to develop and test a learning framework for an agent embedded in an arbitrary network structure. Therefore, we tested our approach on a variety of commonly used and realistic network structures: regular graphs (lattices), small-world networks [27], random graphs [7], and scale-free graphs [2].

Experimental design. In order to evaluate our learning framework, we ran simulations of the team formation model with the agents embedded in various social network structures. The primary metric is global performance: that is, the percentage of advertised tasks that are completed successfully. All of our experiments were in agent societies that consisted of 100 agents connected in a social network with approximately 400 edges. A similar number of edges (network density) ensured that on average, nodes in each of the networks had a comparable number of neighbors with whom they could form teams.

We generated 30 such network structures from each of the four network classes: lattice, small-world, random, and scale-free. All inconsistent network structures and network structures whose number of edges did not fall into the range $[390, 410]$ were rejected as deviating too much from the average network density, and their generation was repeated until we obtained a structure meeting these criteria.

The regular graphs we generated were wrapped two-dimensional lattices of size 10×10 (effectively forming a torus) with a coordination number of 2 (i.e., with each node connected to its two nearest neighbors in each direction). These graphs have 100 nodes and 400 edges, and were used to determine the standard size and network density of the other generated graphs.

Small-world networks followed the model presented by Watts and Strogatz [27], and were built by starting with a lattice, then randomly rewiring each edge with a probability of 0.1.

Random graphs were generated by adding an edge $e_{i,j}$ between agents i and j randomly with probability of 0.0808. (The expected number of edges for a random graph with this edge probability and 100 nodes is approximately 400.)

Scale-free graphs were generated by first adding two connected nodes, and then the rest of the nodes were added iteratively following the preferential attachment model described by Albert and Barabási [2]. (The nodes tend to connect to nodes that are already highly connected, i.e., that have high degree.) In this model, the probability that a new node i will be connected to an already existing node j is $p_{i,j}$, where d_j denotes the current degree of node j:

$$p_{i,j} = m_0 \left(\frac{d_j}{\sum_{k=1}^{i-1} d_k} \right)^c . \tag{8}$$

In our networks, we used $m_0 = 2.7$ and $c = 0.88$; with 100 vertices, these parameter settings yield graphs with approximately 400 edges.

We ran a series of simulations on each of the network structures with the agents using the random policy to initiate and join teams, and a second series with the agents learning team formation policies in real time. The team formation model was parameterized to provide a heavily loaded task environment. Each simulation was run for 20,000 time steps. In all simulations, σ=10 and $|T_k|$=10. A new task was introduced to the system every time step (μ=1) and each task was advertised for up to 10 time steps (γ=10). Once a task began execution, it executed for 10 time steps ($\alpha = 10$). Given this environment, an agent could receive at most ten simultaneous task advertisements. An environment with shorter task lengths would decrease the workload for the agents, and consequently make the problem easier. (Under a sufficiently light task load, agents simply learn to join as often as possible, given the eligibility constraints.) Conversely, longer task lengths would overload the network significantly.

The initial utility values returned by the value functions (see Section 3) were set to 1.0 for waiting and to 0.1 for joining any available task ($Q(s_{RL}, wait) = 1.0$ and $Q(s_{RL}, join) = 0.1$). Other reinforcement learning parameters were $n_{RL} = 10$, $\alpha_{RL} = 0.01$, and $\gamma_{RL} = 0.975$. The idleness limit was set to 100; i.e., learning agents were forced to join some team whenever they went 100 time steps without attempting to join any teams. The *decay discount factor* was $\beta = 0.9995$ and the negative instances were filtered with ratio $r = 0.1$ (see Equation 7).

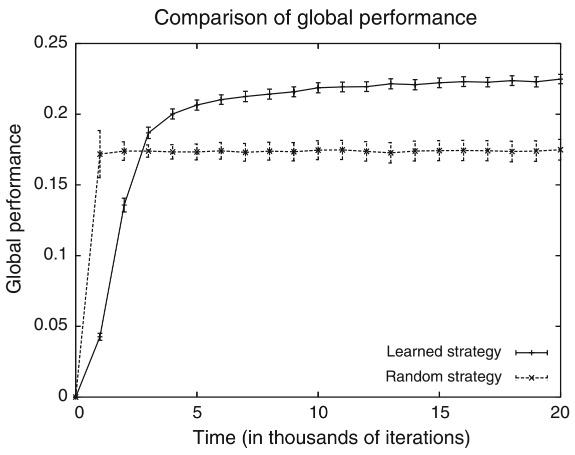
These parameters were determined experimentally. A higher idleness limit led to slow learning, because initially some agents refused to join any teams, although they eventually managed to improve their behavior. A lower idleness limit only slightly influenced learning negatively (agents usually were eager to join teams after the initial period), unless the limit was set extremely low. In such cases, agents were forced to join almost all the time, and the system's performance resembled the performance of the random team joining policy. Interestingly, agents were usually forced to join because of the idleness limit only during the very beginning of the simulation, when they had no experience. After the initial period, forced joins were almost non-existent, and did not significantly influence the policy learned by the agents.

Similarly, setting the decay discount factor to a lower value caused agents to ignore their past experiences too quickly, and agents never managed to learn a good policy. On the other hand, further increasing the decay discount factor caused a decrease in performance after some time, when the old experiences—which were no longer relevant to the current situation—were still taken into significant consideration by the classifier.

Comparison of strategies. We found that agents were able to learn effective team formation policies (compared to the random policy), regardless of the underlying social network structure. In all of the networks, the learned policies significantly outperformed the random policy. The average results over all networks are presented in Fig. 1. The error bars show the 95% confidence interval for each measurement.

The agents consistently improved their performance over the first 3000 time steps. This gain in performance can be explained by the agents' use of accumulated experience, and the fact that initially not many agents were deciding to join any teams,

Fig. 1 Average performance



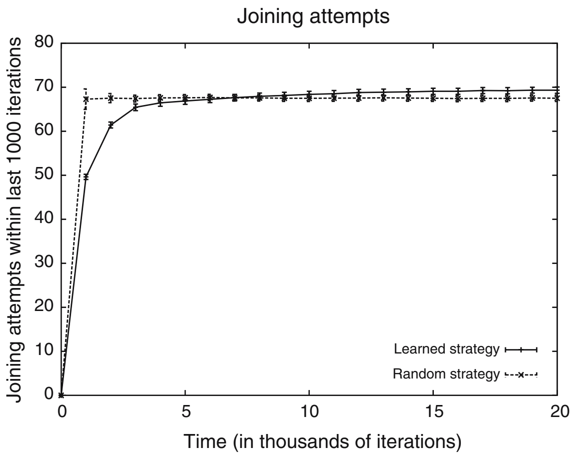
or were choosing to start different teams than their neighbors. The first observation (initially no team formation) was partially caused by the fact that the initial utility of waiting was set very high. (Setting it to a lower value prevented agents from properly learning to wait and not steal positions on tasks away from other agents with higher probabilities of success on those tasks.)

We also found that the improved performance of the learned strategy did not significantly increase the average number of team joining attempts over that of the random policy. This result suggests that the learned strategies were also able to effectively decide when to wait, and were able to choose tasks that were more likely to succeed when they did join teams. This phenomenon can be seen in Figs. 2 and 3.

Discussion. The results of our simulation show that the agent community ultimately learned how to coordinate their team formation decisions. We observed that the agents sometimes effectively "partition" the network into areas, depending on the current conditions, so that agents from one area do not interfere with forming teams from another area. The agents also developed different joining policies that were sensitive to their varying positions in the network. The last observation is consistent with intuition because an agent who has multiple neighbors (a hub node in the network) should have a different team joining policy than a satellite node. A hub node has a much wider selection of tasks and far greater possibility of blocking the network (a possible bottleneck). A satellite node, on the other hand, depends very much on the strategies of its neighbors (e.g., it makes little sense for such a node to start a new team, if its neighbors have not demonstrated in the past that they are willing to join such a team). Moreover, the fact that the number of joining attempts did not significantly increase despite the improved performance indicates that agents learned to wait more efficiently. That is, agents can perceive when the conditions are unfavorable, allowing other agents with a higher probability of success to join a task.

Although the learned strategy always achieved higher performance than the random strategy, the performance for both strategies also varied depending on the underlying network structure. The values for organizational performance qualitatively matched previously reported results [9, 10]. Figure 4 shows the average performance of the learned strategies compared to the random policy for each of the network

Fig. 2 Average number of joining attempts for each agent



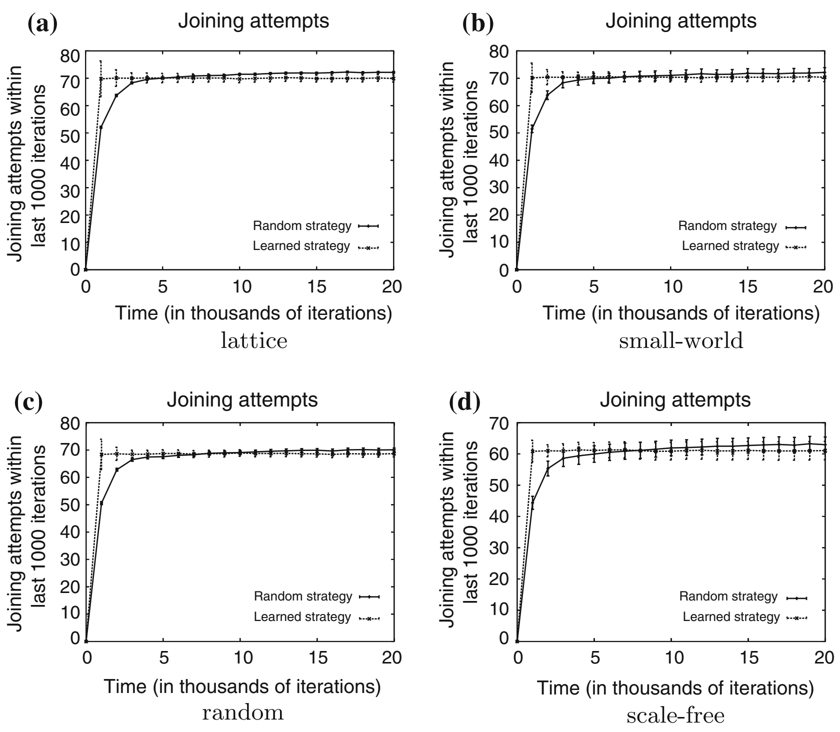


Fig. 3 Average number of joining attempts for each agent for each of the four network structures. (**a**) lattice, (**b**) small-world, (**c**) random, (**d**) scale-free

structures. For all network structures, the agents that learned were able to increase organizational performance, demonstrating that it is possible to learn, independent of the social network topology.

Our approach only compares the learned strategy to the random strategy, which is a useful lower baseline. The task load in the environment was intentionally very heavy, and agents were not capable of executing all the tasks given the network restrictions, lack of global knowledge, and skill diversity. However, we selected the new task introduction rate, task size, advertisement and execution times so that the upper theoretical performance limit would be 1.0 in a society represented by a fully connected graph (effectively no network restriction) and with only one skill existing in the society (all agents are identical). Additionally, we tried to estimate experimentally the upper bound in an environment where there is no network restriction, agents have access to global information, and they have diverse skills (10 skills; $\sigma = 10$). The best result we obtained in such an environment using a greedy approach was 0.55. Unfortunately, there is no easy way to estimate the exact theoretical upper limit of performance for the parameters we used, since the size and complexity of the problem makes an optimal solution intractable.

The main criterion we measured was the performance of the whole organization, defined earlier as the ratio of tasks executed (equivalent to number of teams successfully formed) to the number of all tasks introduced to the system.

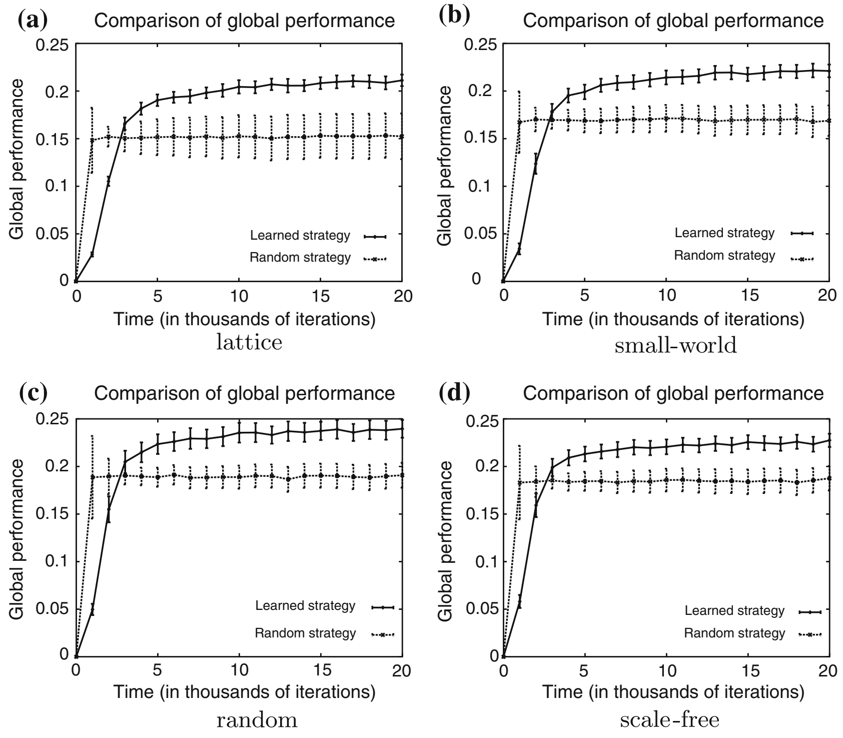


Fig. 4 Average organizational performance for each of the four network structures. (**a**) lattice, (**b**) small-world, (**c**) random, and (**d**) scale-free

An alternative criterion we considered was the local efficiency of agents, which is defined as:

$$\text{efficiency} = \frac{\text{\# of teams successfully formed}}{\text{\# of task commitments}} \qquad (9)$$

Local efficiency reflects the strain on resources put by the joining policy. Low efficiency indicates that agents may waste their efforts by trying to join teams that will not be successful. It should also be stressed that high efficiency does not necessarily correspond to high performance. It is possible to maintain high efficiency by taking few risks, and only joining tasks with a high probability of success. Such a strategy, however, will lead to a situation where agents rarely join tasks unless forced to do so, which would impact organizational performance adversely.

5. Motivational application domains

The model for team formation presented above and our results show that it is possible to learn efficient strategies in a distributed, networked multi-agent system. The ideas presented here can be of practical importance, if further advanced and expanded. Mechanisms for team formation and protocols for efficient and effective teamwork are required by many different types of multi-agent systems. As agent systems are increasingly deployed in large-scale environments, the impact of agent interactions will grow. It will become necessary to understand, engineer, and control the behavior of agents and agent organizations, preferably promoting self-organization and learning. We briefly present two domains that motivate the need for understanding patterns of agent interactions.

Disaster and emergency response. Disaster and emergency response has been identified as a challenging and important research application for multi-agent systems. Simulated emergency-response environments have been created for use in multi-agent systems research [14, 18, 24]. In multi-agent disaster response, there can be many agents with many different skills, such as fire brigades, ambulance crews, police officers, civilian volunteers, and even military personnel.

One disaster response scenario requires decentralized coordination (or coordination with a local dispatcher) among the agents in the environment. When these agents must coordinate to respond to disasters, the relations among the agents play an essential role. The agents must know about, and be able to communicate with, other agents in the environment. Relationships between agents could also be built on other factors (e.g., trust or quality of historical support). An additional dimension of complexity is added when the dynamics of emergencies are considered. It is possible to imagine scenarios in which several emergencies happen simultaneously, and agents in local regions of the environment become busy. If the network structure of the first responders facilitates it, agents from other regions of the environment could assist in areas of high demand. Clearly, in this setting, the interconnections among the different agents and the different types of agents will have an impact on the effectiveness of the agent organizations to respond to emergencies.

Sensor networks. Sensor networks are rapidly becoming an important area of research for the multi-agent systems community. Applications of sensor networks include environmental monitoring, structural modeling, disaster management, health care, and manufacturing [4]. Sensor networks can be either wired or wireless. Wireless sensor networks present several unique challenges, including network connectivity among sensors (i.e., agents) situated in some physical space, which largely determines the connectivity of a sensor network.

Some sensor networks are comprised of homogeneous agents, but more realistic environments are made up of heterogeneous agents. Many different types of agents can be included in sensor networks, including sector managers, data collectors, data routers, and end point sensors. In these situations, the role that an agent plays and the interconnectivity of the agents is important for the overall efficiency of the network. An ability to learn the effects of interconnectivity by the agents themselves will be essential to designing autonomously operating networks, and ensuring the effectiveness of multi-agent sensor networks.

6. Future work and conclusions

We have demonstrated that agents can learn efficient and effective policies for joining and initiating teams in a real-time, heavily loaded task environment. Additionally, our learning framework allows the agents to significantly improve their organizational performance in a wide variety of network structures.

In the future, we plan to further investigate the use of various machine learning techniques in our learning framework. In particular, we are exploring the use of ensemble techniques for online learning, as well as mechanisms for learning from imbalanced data sets. It would be useful for the designers of agents for large-scale multi-agent systems to understand the trade-offs among different learning techniques for team formation and the impact of these techniques on organizational performance and individual agent policies. We also plan to conduct a more in-depth study of the policies that agents learn, with a particular focus on comparing the learned strategies for various social network structures and for different positions within these structures.

As discussed earlier, the current model assumes that all agents are behaving in a locally self-interested manner. An environment that included genuinely cooperative or genuinely malicious agents would require different learning methods. In such an environment, it could be useful for the agents to incorporate notions of trust and reputation when deciding which agents to work with. The task model could also be extended to incorporate richer task payoff models (e.g., unequal distribution of profits to agents depending on role or performance).

Acknowledgements The authors would like to thank the other members of the MAPLE Laboratory for invaluable comments and assistance. We would also like to thank the reviewers, whose comments helped us to improve this article.

References

1. Abdallah, S., & Lesser, V. (2004). Organization-based cooperative coalition formation. In: *Proceedings of the IEEE/WIC/ACM international conference on intelligent agent technology (IAT)*.
2. Albert, R., & Barabási, A. (2002). Statistical mechanics of complex networks. *Review of Modern Physics*, 99, 7314–7316.
3. Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research, 27,* 819–840.
4. Culler, D., Estrin, D., & Srivastava, M. (2004). Overview of sensor networks. *IEEE Computer, 37,* 41–19.
5. Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning, 29*, 103–130.
6. Dutta, S., & Sen, S. (2003). Forming stable partnerships. *Cognitive Systems Research, 4*, 211–221.
7. Erdos, P., & Renyi, A. (1959). On random graphs. *Publicationes Mathematicae, 6*, 290–297.
8. Excelente-Toledo, C. B., & Jennings, N. R. (2003). Learning when and how to coordinate. *Web Intelligence and Agent System, 1,* 203–218.
9. Gaston, M., & desJardins, M. (2003). Team formation in complex networks. In: *Proceedings of the 1st NAACSOS conference*.
10. Gaston, M., Simmons, J., & desJardins, M. (2004). Adapting network structures for efficient team formation. In: *Proceedings of the AAMAS-04 workshop on learning and evolution in agent-based systems*.
11. Gaston, M. E., & desJardins, M. (2005). Agent-organized networks for dynamic team formation. In: *Proceedings of the 2005 international conference on autonomous agents and multi-agent systems*.
12. Horling, B., Benyo, B., & Lesser, V. (2001). Using self diagnosis to adapt organizational structure. In: *Proceedings of the 5th international conference on autonomous agents*.
13. Hunsberger, L., & Grosz, B. (2000). A combinatorial auction for collaborative planning. In: *Proceedings of the fourth international conference on multi-agent systems (ICMAS-2000)*.
14. Kitano, H. (2000). RoboCup rescue: a grand challenge for multi-agent systems. In: *Proceedings of the fourth international conference on multi-agent systems (ICMAS-2000)*.
15. Kraus, S., Shehory, O., & Taase, G. (2003). Coalition formation with uncertain heterogeneous information. In: *Proceedings of the second international joint conference on autonomous agents and multiagent systems (AAMAS ’03)*.

16. Nair, R., Tambe, M., & Marsella, S. (2002). Team formation for reformation. In: *Proceedings of the AAAI spring symposium on intelligent distributed and embedded systems*.
17. Nair, R., Tambe, M., & Marsella, S. (2003). Role allocation and reallocation in multiagent teams: Towards a practical analysis. In: *Proceedings of the second international joint conference on agents and multiagent systems (AAMAS)*.
18. Nair, R., Tambe, M., & Marsella, S. (2003). Team formation for reformation in multiagent domains like RoboCupRescue. In: G. A. Kaminka, P.U. Lima & R. Rojas (Eds.), *Proceedings of RoboCup-2002 international symposium*. Springer Verlag.
19. Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In: B. Schölkopf, C. Burges, & A. Smola (Eds.), *Advances in kernel methods—support vector learning.* MIT Press.
20. Pynadath, D., & Tambe, M. (2002). Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In: *Proceedings of the international conference on autonomous agents and multiagent systems (AAMAS '02)*.
21. Quinlan, R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers.
22. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. MIT Press.
23. Swaminathan, J. M., Smith, S. F., & Sadeh, N. M. (1998). Modeling supply chain dynamics: a multiagent approach. *Decision Sciences, 29*, 607–632.
24. Tadokoro, S. (2003). RoboCupRescue robot league. In: G. A. Kaminka, P. U. Lima and R. Rojas (Eds.), *Proceedings of RoboCup-2002 international symposium*. Springer-Verlag.
25. Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research, 7*, 83–124.
26. Thadakamalla, H. P., Raghaven, U. N., Kumera, S., & Albert, R. (2004). Survivability of multiagent-based supply networks: a topological perspective. *IEEE Intelligent Systems, 19*, 24–31.
27. Watts, D., & Strogatz, S. (1998). Collective dynamics of 'small-world' networks. *Nature, 393,* 440–442.
28. Witten, I. H., & Frank, E. (2000). *Data mining: practical machine learning tools with Java implementations.* Morgan Kaufmann.
29. Zhen, Z., & Webb, G. (2000). Lazy learning of Bayesian rules. *Machine Learning, 41,* 53–84.