Towards Flexible Coordination of Large Scale Multi-Agent Teams

Yang Xu,[1] Elizabeth Liao,[2] Paul Scerri,[2] Bin Yu,[2] Mike Lewis[1] and Katia Sycara[2]

[1] School of Information Sciences, University of Pittsburgh
{yxu, ml}@sis.pitt.edu
[2] School of Computer Science, Carnegie Mellon University
{eliao, pscerri, byu, katia}@cs.cmu.edu

Summary. As a paradigm for coordinating cooperative agents in dynamic environments, *teamwork* has been shown to be capable of leading to flexible and robust behavior. However, when teamwork is applied to the problem of building teams with hundreds of members, its previously existing, fundamental limitations become apparent. In this paper, we address the limitations of existing models as they apply to very large agent teams. We develop algorithms aimed at flexible and efficient coordination, applying a decentralized social network topology for team organization and the abstract coordination behaviors of Team Oriented Plans (TOPs). From this basis, we present a model to organize a team into dynamically evolving subteams, in order to flexibly coordinate the team. Additionally, we put forward a novel approach to sharing information within large teams, which provides for targeted, efficient information delivery with a localized reasoning process model built on previously incoming information. We have developed domain- independent software proxies, with which we demonstrate teams of an order of magnitude larger than those previously discussed in known published work. We implement the results of our approach, demonstrating its ability to handle the challenges of coordinating large agent teams.

1 Introduction

When a group of agents coordinates via *teamwork*, they can flexibly and robustly achieve joint goals in a distributed, dynamic and potentially hostile environment[7, 12]. Using basic teamwork ideas, many systems have been successfully implemented, including teams supporting human collaboration[4, 26], teams for disaster response[19], for manufacturing[12], for training[28] and for games[14]. While such teams have been very successful, their sizes have been severely limited. To address larger and more complex problems, we need teams that are substantially larger, yet retain the desirable properties of teamwork.

The key to the success of previous teamwork approaches is the explicit, detailed model each agent has of the other agents and the joint activity of the team. Team members use these models to reason about actions that will aid the achievement of joint goals[11, 28]. However, when the size of a team is scaled up, it becomes unfeasible to maintain up-to-date, detailed models of all other teammates, or even of all team activities. Specifically, the communication required to keep the models up to date does not scale well with the number of agents. Without these models, key elements of both the theory and operationalization of teamwork break down. For example, without accurate models of team activities, STEAM's communication reasoning[28] cannot be applied, nor can Joint Intention's reasoning about committments[11].

In this paper, we present a model of teamwork that does not rely on the accurate models of the team that previous approaches to teamwork use. By not requiring accurate models, we limit the required communication and thus make the approach applicable to very large teams. However, giving up the accurate models means that the cohesion guarantees provided by approaches such as Joint Intentions can no longer be provided. Instead, our algorithms are designed to lead to cohesive, flexible and robust teamwork *with high probability.*

The basic idea is to organize the team into dynamically evolving, overlapping subteams that work on sub-goals of the overall team goal. Members of a subteam maintain accurate models of each other and the specific subgoal on which they are working. To ensure cohesion and minimize inefficiency across the whole team, we connect all agents of the whole team into a network. By requiring agents to keep their neighbors in the network informed of the subgoals of subteams they are members of, there is high probability that inefficiencies can be detected and subsequently addressed. Using this model we have been able to develop teams that were effective, responsive and cohesive despite having 200 members. We identify three ideas in the model as being the keys to its success.

The first idea is to break the team into subteams, working on subgoals of the overall team goal. The members of a subteam will change dynamically as the overall team rearranges its resources to best meet the current challenges, respond to failures or sieze opportunities. Within these subteams, the agents will have accurate models of each other and the joint activity, in the same way a team based on the STEAM model would. Thus, using techniques developed for small teams, the subteam can be flexible and robust. Moreover, we identify two distinct groups within the subteams: the team members actually performing roles within the plan; and team members who are not, e.g., agents involved via role allocation. The fidelity of the model maintained by the role performing agents is higher than that of the non-role performing agents, which is in turn higher than other agents in the wider team. Because models are limited to subteams, communication overhead is limited.

To avoid potential inefficiencies due to subteams working at cross purposes, our second idea is to introduce an *acquaintance network*. This network

connects *all agents in the team and is independent of any relationships due to subteams*. Specifically, the network is a *small world network* [30](see figure 1), so that any two team members are separated by a small number of neighbors. Agents share information about their current activities with their direct neighbors in the network. Although the communication required to keep neighbors in the acquaintance network informed is low, due to the small world properties of the network, there is high probability for every possible pair of plans. Some agents will know both, and thus, can identify inefficiencies due to conflicts among the plans. For example, it may be detected that two subteams are attempting to achieve the same goal or one subteam is using plans that interfere with the plans of another subteam. Once detected by any agent the subteams involved can be notified and the inefficiency rectified. Moreover, in this paper we investigate the influences of other social network properties to the efficiency of coordinating large scale teams.

When limiting models of joint activities to the members of a subteam, the overall team loses the ability to leverage the sensing abilities of all its members. Specifically, an agent may locally detect a piece of information unknown to the rest of the team but does not know which members would find the information relevant[8, 33]. For example, in a disaster response team, a fire fighter may detect that a road is impassable but not know which other fire fighters or paramedics intend to use that road. While communication in teams is an extensively studied problem, [5, 13, 21, 32], current algorithms for sharing information in teams either require infeasibly accurate models of team activities, e.g., STEAM's decision theoretic communication[28], or require that centralized information *brokers* are kept up to date[27, 3], leading to potential communication bottlenecks. Our solution for information sharing among large teams is to perform distributed information sharing without the cost of maintaining accurate models of all the teammates. An agent can easily know what information it needs, but it will not know who has the information, while another agent has the information but does not know who needs it. By allowing the agents to simply forward the information to an acquaintance in a better position to make the decision, we spread the reasoning across the team, leveraging the knowledge of many agents. We also leverage the idea that information is always interrelated and a received piece of information can be useful in deciding where to send another piece of information, if there is a relationship between two pieces of information. For example, when coordinating an agent group in urban search and rescue, if agent a tells agent b about a fire at 50 Smith St, when agent b has the information about the traffic condition of Smith St, sending that information to agent a is a reasonable thing to do, since a likely either needs the information or knows who does. By utilizing the interrelationship between pieces of information, agents can more quickly route new information through the acquaintance network. Moreover, agents do not model information, rather they model the acquaintances to which they send information. It may take several hops for a message to get to an agent that needs the information. Since each piece of information informs the de-

livery of other pieces and models are updated as the message moves, as the volume of information to be shared among the team increases, the amount of effort required per piece of information actually decreases. Moreover, since agents need to only know about their acquaintances, the approach scales as the number of agents in the team increases.

To evaluate our method for building large teams, we have implemented the above approach in software proxies[22] called Machinetta. A proxy encapsulating coordination algorithm works closely with a "domain level" agent and coordinates with other proxies. Although Machinetta proxies build on the successful TEAMCORE proxies[28] and have been used to build small teams[24], they were not able to scale to large teams without the fundamentally new algorithms and concepts described above. In this paper, we report results of coordinating teams of 200 proxies that exhibit effective, cohesive team behavior. Such teams are of an order of magnitude larger than previously discussed in known published work proxy-based teams[24], hence they represent a significant step forward in building large teams. To ensure that the approach is not leveraging peculiarities of a specific domain for its improved performance, we tested the approach in two distinct domains using identical proxies.[3]

2 Toward Flexible Team Coordination

In this section, we provide a detailed model of the organization and coordination of the team. At a high level, the team behavior can be understood as follows: A team is organized as a social network and team members detect events in the environment that result in plans to achieve the team's top-level goal. The team finds subteams to work on those plans and within the subteams the agents communicate to maintain accurate models to ensure cohesive behavior. Across subteams, agents communicate the goals of the subteams so that interactions between subteams can be detected and conflicts resolved. Finally, agents share locally sensed information on the associates' network to allow the whole team to leverage the local sensing abilities of each team member.

2.1 Building Large Scale Teams

A typical large scale team meets the following basic characteristics: there are large number of widely distributed team members with limited communication bandwidth. As a part of a large team, agents coordinate closely only with a subset of the total agents of the team. Based on these characteristics, we can define a logical model of the team organized as an *acquaintance network*. The *acquaintance network* is a directed graph $G = (A, N)$, where A is the team of agents and N is the set of links between any two agents. Specifically,

[3] A small amount of code was changed to interface to different domain agents.

for $< \alpha_i, \alpha_j > \in N$ for any two agents $\alpha_i, \alpha_j \in A$ denotes that α_i and α_j are acquaintances able to exchange tokens. Specifically, $n(\alpha)$ is defined as all the acquaintances of agent α. Note that the number of each agent's acquaintances is much less than the size of the agent team $|A|$. We additionally require that the acquaintance network be a *small world network*. Such networks exist among people and are popularized by the notion of "six degrees of separation" [18]. When agents are arranged in a network, having a small number of neighbours relative to the number of members in the team, the number of agents through which a message must pass to get from any agent to any other, going only from neighbour to neighbour, is typically very small. A subset of a typical acquaintance network for a large team is shown as figure 1. In the figure, each node represents an agent member in the team. When pairs of agents are connected, they can directly communicate with each other as acquaintances.

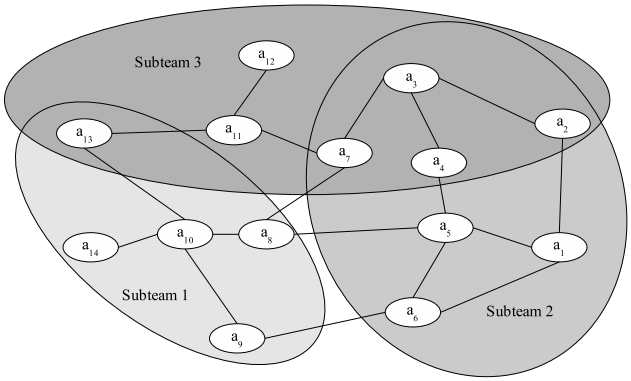


Fig. 1. Relationship between subteams and the acquaintance network

2.2 Team Oriented Plans

Team Oriented Plans (TOPs) are the abstraction that define team behavior. The TOPs provide the mapping from team level goals to individual *roles* that are performed by individual team members. Suppose the team A has a top level goal, G. The team *commits*, with the semantics of STEAM to G [28]. Achieving G requires achieving sub-goals, g_i, that are not known in advance but are functions of the environment. For example, sub-goals of a high-level goal to respond to a disaster could be to extinguish a fire and provide medical attention to particular injured civilians. To achieve sub-goals, the team follows plan templates represented in a library. These templates are parameterized while instantiated plans contain the specific details [23]. For example, when a particular fire in a building is detected by a team member, the plan will be instantiated because it matches a template for disaster response.

Each sub-goal is addressed with a plan, $plan_i =< g_i, recipe_i, roles_i, d_i, m_i >$, that matches a plan template in the library. The overall team thus has plans $Plans(t) = \{plan_1, \ldots, plan_n\}$. Individual team members will not necessarily know all plans. To maximize the responsiveness of the team to changes in the environment, we allow any team member to commit the team to the execution of a plan, when it detects that subgoal g_i is relevant. Team members can determine which sub-goals are relevant by the plan templates specified in the library. $Recipe_i$ is a description of the way the sub-goal will be achieved[11] including the execution order of the components in the plan. $Roles_i = \{r_1, r_2, r_3, ...r_r\}$ are the individual activities that must be performed to execute $recipe_i$. d_i is the domain specific information pertinent to the plan. For convenience, we write $perform(r, a)$ to signify that agent, a, is working on role, r. $Subteam_i$ includes any agents working on $plan_i$ *and* their neighbors in the acquaintance network. The identities of those agents involved in role allocation is captured with $allocate(plan_i)$. In the cases where either a conflict or synergy is detected, all but one of the plans must be terminated. The domain specific knowledge of a termination of a plan can be defined as ${}^{term}recipe_i$.

We can think about TOPs as active objects in a distributed database. Each TOP "object" captures the state of a particular team plan. Team members involved in the execution of that plan need to have up-to-date versions of the TOP "object", e.g., knowing which team members are performing which roles and when TOPs are complete. Information needs to be shared to ensure there is synchronization across the same object held by different team members. Viewed in this manner, coordination can be thought of as a set of algorithms to fill in fields on the TOP objects and ensure synchronized objects across the team. For example, some coordination algorithms are triggered when there are open roles in the TOP objects and other algorithms are triggered when the post-conditions on the plan are satisfied.

2.3 Subteams

Although individual agents commit the team to a sub-goal, it is a subteam that will realize the sub-goal. The subteams formation process commences when an individual agent detects all the appropriate preconditions that matches a plan template in the library and subsequently instantiates a plan, $plan_i$. For each of the $roles_i$ in $plan_i$, a role token is created to be allocated to the team. We are using LA-DCOP for role allocation[6], which results in a dynamically changing subset of the overall team involved in role allocation. This works as follows: the token is passed from one team member to the next until an agent finally accepts the role. Once accepted, the agent becomes a member of the subteam and makes a temporary commitment to perform the role represented by the token. Note that agents can accept multiple tokens and therefor can perform more than one role and thus, belong to multiple subteams. Since allocation of team members to roles may change due to failures or changing

circumstances, the members of a subteam also change. One example of this is when a member decides to drop a role for a more suitable task. This will lead to the best use of team resources because team members will execute roles that they are most capable of doing.

All subteam members, agents performing roles and their informed acquaintances, must be kept informed of the state of the plan, e.g., they must be informed if the plan becomes irrelevant. This maximizes cohesion and minimizes wasted effort. Typically $|subteam_i| < 20$, although it may vary with plan complexity and notice that typically, $subteam_i \cap subteam_j \neq \emptyset$ where $i \neq j$. In the experiments that follow, a simple plan contains 1-2 roles and 1-2 preconditions compared to a complex plans that have 4-5 roles and 9-10 preconditions. This occurs because agents can accept more than one role and usually belong to more than one subteam due the acquaintance network. These subteams are the basis for our coordination framework and leads to scalability in teams.

2.4 Plan Deconfliction

In this section, we describe how to resolve plan conflicts. When using distributed plan creation, two problems may occur. Upon detecting the appropriate preconditions, different team members may create identical plans or plans with the same p_g but different p_{recipe}. To reduce the need for plan deconfliction, we need to choose a rule for plan instantiation to reduce the number of plans created with the same p_g. These instantiation rules include *always instantiate, probabilistic* and *local information*. The choice of the plan instantiation rule will vary with the domain setting.

If two plans, $plan_i$ and $plan_j$ have some conflict or potential synergy, then we require $subteam_i \cap subteam_j \neq \emptyset$ to detect it. There must be a common team member on both subteams to maintain mutuals beliefs of the plans and hence detect the conflict. A simple probability calculation reveals that the probability of a non-empty intersection between subteams, i.e., the probability of an overlap between the teams, is:

$$Pr(overlap) = 1 - \frac{{}_{(n-k)}C_m}{{}_nC_m}$$

where ${}_aC_b$ denotes a combination, n = number of agents, k = size of $subteam_i$ and m = size of $subteam_j$.

Hence, the size of the subteams is critical to the probability of overlap. For example, if $|subteam_i| = |subteam_j| = 20$ and $|A| = 200$, then $P(overlap) = 0.88$, despite each subteam involving only 10% of the overall team. Since the constituents of a subteam change over time, this is actually a lower bound on the probability that a conflict is detected.

After a conflict is detected, the plan needs to be terminated; the same follows with completion of goals or recipes and irrelevant or unachievable

plans. We capture the domain specific knowledge that defines these conditions with ${}^{term}p_{recipe}$. In exactly the same way as STEAM, when any $a \in subteam_i$ detects any conditions in ${}^{term}p_{recipe}$, it is obliged to ensure that all other members of $subteam_i$ also know that the plan should be terminated. In this way, the team can ensure that $plan_i \subseteq plans(t)$, i.e., no agent believes the team is performing any plan that it is not performing.

2.5 Plan Instantiation Rules

In distributed plan instantiation, an agent can create a plan when all preconditions have been fulfilled and the plan matches a template in a library. However, since this may increase the total number of plans created, agents can only create a plan using one of three rules for instantiating plans. These rules differ in terms of the information needed to compute whether the instantiation conditions apply. The first rule, the *always instantiate rule*, is used as a baseline for the other instantiation rules. An agent is allowed to create a plan when it knows of all the preconditions necessary for the plan.

The second rule, the *probabilistic instantiation rule*, requires no knowledge of other team members. This method requires that team members wait a random amount of time before creating the plan. If during that time, it has not been informed by an informed acquaintance that another teammate is creating the same plan, it will proceed and create the plan. Thus plans will only be created during the time it takes for all team members to hear of the plan. The advantage of this rule is that no information is required of other team members. There are two disadvantages. First, there may be conflicting plans which must be later resolved. Second, there may be a significant delay between detection of the preconditions and the instantiation of the plan. These disadvantages can be traded off in the following manner. By increasing the length of time a team member can wait, the number of conflicts will be reduced, but the delay will be increased.

We can use information about who locally senses information to define another rule. This rule, which we refer to as the *local information rule*, requires that a team member detect some of the plan's preconditions locally in order to instantiate the plan. Although this will lead to conflicting plans when multiple agents locally sense preconditions, it is easier to determine where the conflicts might occur and resolve them quickly. The major disadvantage of this rule is that when a plan has many preconditions, the team members that may detect specific preconditions may never get to know all the preconditions and thus the plan will never be created.

3 Toward Efficient Communication in Large Scale Teams

Information is important in coordinating large scale multi-agent teams because each team member has to adjust its activity according to the changes

in its team, teammates, and the environments. Communication is difficult because the members only have a partial views of the environment and a team member may have a piece of valuable information but not know who needs the information [31]. In this section, we explain our objective of efficient communication in terms of providing high quality information with targeted information delivery.

3.1 Information fusion

Each of the agents, when working in physical working plate, can be deemed as mobile sensors and the team can be deemed as a sensor network. We first look at the problem of information fusion in large scale teams, which not only observe physical phenomena, but also conduct high-level information processing tasks, e.g., attacking a target in a battlefield. In large teams, the sensor data generated by a single agent usually has low confidence. The low confidence sensor data cannot be used directly for coordinating plans and actions and needs to be fused with other relevant data in the team [25]. Many power-aware protocols and algorithms have been developed for static sensor networks, but very limited research has been done for the design of routing algorithms for information fusion [1, 35]. For example, in directed diffusion and geographic routing [9, 15], each source agent does not send the data back to the sink until it receives a query from the sink. For this reason, these routing protocols are called *reactive protocols*.

Reactive protocols are mainly designed for static sensor networks and are not appropriate for large scale teams, which are mobile sensor networks. Specifically, there are two key reasons. 1. The location of the data is not correlated with existing positions of mobile sensors., i.e., agent b previously knew agent a has the data in one location, but when his query comes, agent a has moved to another location. 2. Sinks agents usually do not know when source agents will have the data, so they have always sent out volume of query.

In this section we present a *proactive protocol* for information fusion in large scale team based on our acquaintance network model. In proactive protocols, there is no querying process and each source agent, when sensing a piece of data, can proactively deliver the data to other nodes in the network. Without the querying process, the source agent has to reason about who might have other relevant data and can fuse its sensor data. In order to minimize the traffic and redundant data in the network, each node forwards the sensor data to only one of its neighbors. Without centralized control, the agent has to intelligently deliver data for fusion solely based on itself and its neighbors. The challenge, with various decisions being made by the individual agents, is how to maximize the probability that relevant data will be fused in the network, e.g., fused by at least one node in the network.

Random walks are a simple algorithm for information fusion. In random walks, when an agent receivessensor data it randomly choses a neighbor to send to. Once the neighbor receives the data, it repeats the same process

until the events are successfully fused or the data reaches the stop condition. However, random walks are not efficient for information delivery when more than two agents detect the same event on the ground and there is a need to fuse them together. We propose an efficient and failure-resistant localized algorithm — *path reinforcement* algorithm [34], in which each node learns routing decisions from past information delivery processes. The logic behind the algorithm is that relevant information is likely to be fused earlier if agents follow a path they have followed earlier. In the algorithm, a agent a may pass the event to neighbor b if a has passed b relevant events before.

The experiments show that controlled information flows significantly increase the probability of relevant information being fused in the network, such that the probability could be improved by 2 - 5 times for the same hops of information propagation in comparison with random walks [34]. Our experiments indicate that the probability of fusion is surprisingly high even with limited local knowledge of each node and relatively small hops.

3.2 Information Sharing

In the previous section, we showed how requiring mutual beliefs only within subteams acting on specific goals can dramatically reduce the communication required in a large team. However, individual team members will sometimes get domain level information, via local sensors, that is relevant to members of another subteam. Due to the fact that team members do not know what each other subteam is doing, they will sometimes have locally sensed information, while not knowing who requires it. In this section, we present an approach to sharing such information, leveraging the small world properties of the acquaintance network. The basic idea is to forward information to the acquaintance in the acquaintance network who is most likely to either need the information or have a neighbor who does.

The key to the algorithm is the model that the agent maintains of its acquaintances. P_a is a matrix where $P_a[i,b] \rightarrow [0,1], b \in N(a), i \in I$ represents the probability that acquaintance b is the best to send information i to. To obey the rules of probability, we require $\forall i \in I, \sum_{b \in N(a)} P_a^t[i,b] = 1$. For example, if $P_a[i,b] = 0.7$, then a will usually forward i to agent b as b is very likely the best of its neighbors to send to. This situation is illustrated in Figure 4. The more accurate the model of P_a, the more efficient the information sharing because the agent will send information to agents that need it more often and more quickly. P_a is inferred from incoming messages and thus the key to our algorithm is for the agents to build the best possible model of P_a.

Information is encapsulated in *messages*, with some supporting information which is helpful for information sharing. Specifically, a message consists of two parts, $M =< i, path >$. $i \in I$ is the information being communicated. $path$ records the track over which the message has been taken in the network. $last(path)$ denotes the last agent to which the message was sent previous to current agent recipient, via acquaintance network. To ensure that messages

do not travel indefinitely around the network, we stop the message when $|path| \geq$MAX_STEPS.

When a message arrives, the agent state, S_a, is updated by the transition function, δ, which has three parts, δ_H, δ_K, δ_P . First, the message is appended to the history, $\delta_H(m, H_a) = H_a \cup m$. Secondly, the information contained in the message is added to the agent's local information knowledge K_a, $\delta_H(m, K_a) = K_a \cup m.i$.[4] Finally, and most critically for the purpose of the algorithm, δ_P is used to update agent's probability matrix, to help route future message. (We described δ_P in the next section.)

Each agent in the team runs the following algorithm when receiving message m:

Algorithm 1: Information Share (S_a)

(1) $While(true)$
(2) $m \leftarrow getMsg$
(3) $S_a \leftarrow \delta(m,\ S_a)$
(4) $if\ m.|path|\ < MAX\_STEPS$
(5) $\quad APPEND(self,\ m.path)$
(6) $\quad next \leftarrow CHOOSE(P[i, m.j])$
(7) $\quad SEND(next, m)$

In Algorithm 1, when an agent gets a message, it updates its state according to function δ. If an agent finds that the message does not meet the stop condition (line 4), then the function CHOOSE (line 6) selects an acquaintance, according to the probabilities in matrix to pass the message to. Notice, CHOOSE can select any acquaintance, with the likelihood of choosing a particular acquaintance being proportional to their probability of being the best to send to.

The key to our algorithm is for the agent to often pass information to an acquaintance who either needs it or knows who does. These models are created based on previously received information. This requires us making use of the relationship between pieces of information and then mapping it into a mathematic description, i.e. via Bayes Rule. We define the relationships between pieces of information as $rel(i, j) \rightarrow [0, 1], i, j \in I$, where $rel(i, j) > 0.5$ indicates that an agent interested in i will also be interested in j, while $rel(i, j) < 0.5$ indicates that an agent interested in i is unlikely to be interested in j. If $rel(i, j) = 0.5$ then nothing can be inferred. Since rel relates two pieces of domain level information, we assume that it is given (or can be easily inferred from the domain).

Our information sharing algorithm defined an action of δ_P for each piece of relative information i when a received message containing j can be described as follows: assuming information j arrives to agent a from b, then agent a will first decrease the probability of sending this information back to b because clearly b already knows that information. Then H_a should be searched for to

[4] In this paper, we ignore difficult issues related to contradictory information.

find any relevant former information. For each piece of relevant information i, j should be additional evidence for a to make a decision about sending i, and the probability of sending i to b should be strengthened.

The update of agent a's $P_a$based on an incoming message m containing j which is received from c can be achieved by leveraging Bayes Rule as follows:

$$\forall i, j \in I, b \in N(a) \;\; \delta_P^I(P_a[i,b], m =< j, path >, d = first(N(a), m.path))$$
$$= \begin{cases} P_a[i,b] \times rel(i,j) \times \frac{2}{|N|} & \text{if } i \neq j, b = d \\ P_a[i,b] \times \frac{1}{|N|} & \text{if } i \neq j, b \neq d \\ \varepsilon & \text{if } i = j, b \in m.path \cap N(a) \end{cases}$$

Then P must be normalized to ensure $\forall i \in I, \sum_{b \in N(a)} P_a^t[i,b] = 1$. The first case in our equation is the most interesting. It updates the probability that the agent that just sent some information is the best to send other information to, based on the relationships of other pieces of information to the one just sent. Please note, to avoid potential path detours, the message path is determined not according to who directly sent the message, but rather according to the fact that it was a's acquaintance who first got the message. The latter condition changes the probability of sending that information to agents other than the sender in a way that ensures the normalization works. Finally, the third case encodes the idea that you typically would not want to send a piece of information to an agent that sent it to you.

To see how δ_P works, consider the following example at some point doing execution:

$$P_a = \begin{array}{c} \\ i \\ j \\ k \end{array} \begin{array}{c} \begin{array}{cccc} b & c & d & e \end{array} \\ \begin{bmatrix} 0.6 & 0.1 & 0.2 & 0.1 \\ 0.4 & 0.2 & 0.3 & 0.1 \\ 0.4 & 0.4 & 0.1 & 0.1 \end{bmatrix} \end{array}$$

The first row of the matrix shows that if a gets information i it will likely send it to agent b, since $P[i,b] = 0.6$. We assume that agents wanting information i also probably want information j but those wanting k definitely do not want j. That is, $rel(i,j) = 0.6$ and $rel(k,j) = 0.2$. Then a message $m =< j, \{, , d, , b\} >$ with information j arrives from agent b. Applying δ_P^I to P_a we get the following result:

$$P_a = \begin{array}{c} \\ i \\ j \\ k \end{array} \begin{array}{c} \begin{array}{cccc} b & c & d & e \end{array} \\ \begin{bmatrix} 0.5769 & 0.096 & 0.2308 & 0.096 \\ \varepsilon & 0.67 & \varepsilon & 033 \\ 0.4255 & 0.4255 & 0.0426 & 0.1064 \end{bmatrix} \end{array}$$

The effects on P can be inferred as follows: (i) j will likely not be sent back to d and b who previously have gotten j, i.e., $P_a[i,b] = \varepsilon$; (ii) the probability of sending i to d is increased because agents wanting j probably also want i; (iii) the probability of sending k to d is decreased, since agents wanting

j probably do not want k. Notice a knows nothing of the network topology *beyond* its acquaintances $n(a)$.

3.3 Effects of Network Topology on Sharing Efficiency

As noted by social scientists, information sharing efficiency will be impacted by network topology. We have found that in order to share information among large-scale teams, agents adopt the same manners as exhibited by humans operating in social groups.

The properties of social network structures have been comprehensively studied [2, 17]. According to such research, there are several parameters that are important for helping us to understand or predict the behavior of information sharing in large-scale teams. Key factors include the small-world effect, degree distributions, clustering, network correlations, random graph models, models of network growth and preferential attachment, and dynamical processes taking place on networks [11]. Most of them are interrelated. For the purpose of this paper, we specifically focus on only three properties: average distance, degree distribution and average acquaintance.

- Average distance: (commonly studied as "small world effect" [30]. The average distance $l = \frac{1}{\frac{1}{2}n(n+1)} \sum_{a_i,a_j \in A, i>j} distance(a_i, a_j)$, where $n = |A|$ and $distance(a_i, a_j)$ represents the minimum number of agents a_i, a_j that a message must pass through one agent to another via acquaintance network. For example, if agent $a_1$and a_2 are not acquaintances but share an acquaintance, $distance(a_1, a_2) = 1$.
- Degree distribution: (Commonly studied as "scale free effect") The frequency of agents having different number of acquaintances. The distribution can be represented as a histogram where the bins represent a given number of acquaintances and the size of a bin is how many agents have such number of acquaintances [2].
- Average acquaintances: is the average number of acquaintances that agents have in the teams. Its value can be used to infer how many choices agents may have when delivering a message.

Well-known types of social networks can be described using these properties. For example, a *random network* has the "flat" degree distribution. While *grid network* is distinct in that all nodes have the same degree (e.g, four is the only degree in a two dimension grid network). *Small World Network* and *Scale Free Network* [2] are two important types of social network topologies and research has shown that each of them possesses some interesting properties. Small world networks have much shorter average distances as compared with regular grid networks. We hypothesize that the low average distance will improve information sharing efficiency because information can potentially take less "hops" to reach a defined destination. A scale-free network is a specific kind of network in which the degree distribution forms a power-law, i.e, some nodes are very

connected hubs and connect to other nodes much more than ordinary nodes. The hubs in scale-free networks give the advantages of centralized networks, in which the distribution provides the advantages of centralized approaches.

4 Machinetta

A number of algorithms work together to achieve the teamwork, given the framework described above. There are algorithms for allocation roles[6], instantiating plans[16], sharing information[31], human interaction[20] and resource allocation. To avoid requiring a reimplementation of the algorithms for each new domain, the coordination algorithms are encapsulated in a *proxy*[10, 29, 21, 24]. Proxies are becoming a standard mechanism for building heterogeneous teams. Each team member works closely with a single proxy that coordinates with the other proxies to implement the teamwork. The basic architecture is shown in Figure 2. The proxy communicates via a high-level, domain-specific protocol with the robot, agent or person it is representing in the team. Most of the proxy code is domain-independent and can be readily used in a variety of domains requiring distributed control. Our current proxy code, known as Machinetta, is a substantially extended and updated version of the TEAMCORE proxy code[29]. Machinetta proxies are in the public domain and can be downloaded from http://teamcore.usc.edu/doc/Machinetta.

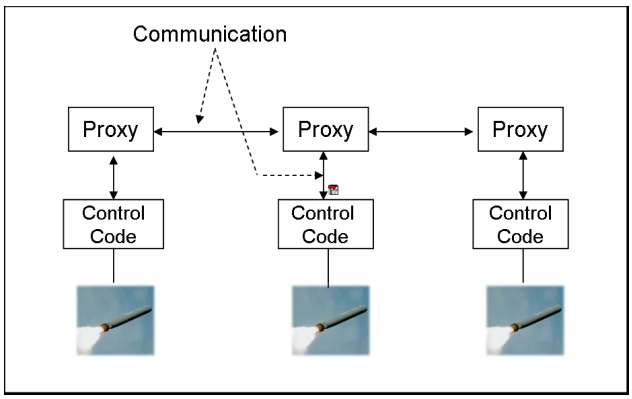


Fig. 2. The basic system architecture showing proxies, control code and Unmanned Aerial Vehicles (UAVs) being controlled.

In a dynamic, distributed system, protocols for performing coordination need to be extremely robust. When we scale the size of a team to hundreds of agents, this becomes more of an issue than simply writing bug-free code. Instead we need abstractions and designs that promote robustness. Towards

this end, we are encapsulating "chunks" of coordination in *coordination agents*. Each coordination agent manages one specific piece of the overall coordination. When control over that piece of coordination moves from one proxy to another proxy, the coordination agent moves from proxy to proxy, taking with it any relevant state information. We have coordination agents for each plan or subplan (PlanAgents), each role (RoleAgents) and each piece of information that needs to be shared (InformationAgents). For example, a RoleAgent looks after everything to do with a specific role. This encapsulation makes it far easier to build robust coordination.

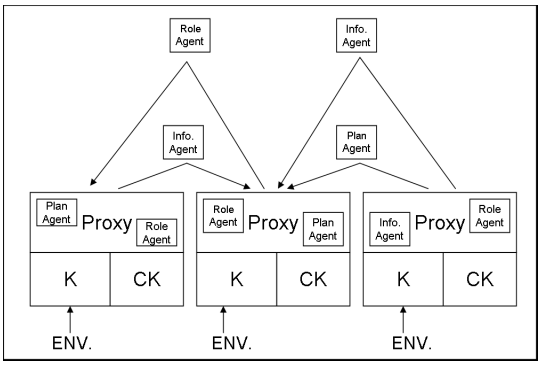


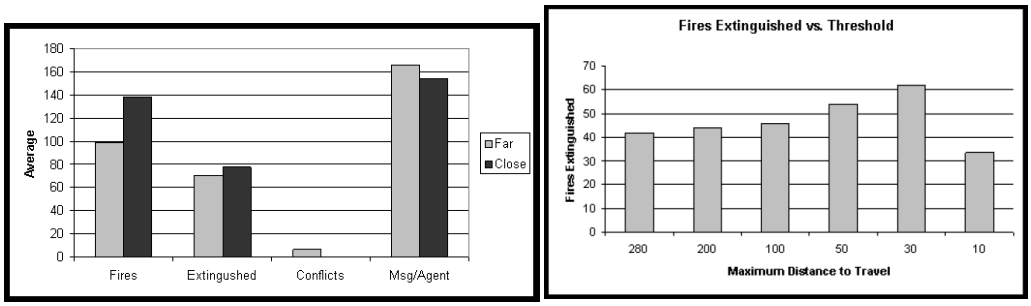
Fig. 3. High level view of the implementation, with coordination agents moving around a network of proxies.

Coordination agents manage the coordination in the network of proxies. Thus, the proxy can be viewed simply as a mobile agent platform that facilitates the functioning of the coordination agents. However, the proxies play the additional important role of providing and storing local information. We divide the information stored by the proxies into two categories, domain specific knowledge, K, and the coordination knowledge of the proxy, CK. K is the information this proxy knows about the state of the environment. For example, the proxy for a UAV knows its own location and fuel level as well as the the location of some targets. This information comes both from local sensors, reported via the domain agent, and from coordination agents (specifically InformationAgents, see below) that arrive at the proxy. CK is what the proxy knows about the state of the team and the coordination the team is involved in. For example, CK includes the known team plans, some knowledge about which team member is performing which role, and the TOP templates. At the most abstract level, the activities of the coordination agents involve moving around the proxy network, adding and changing information in C and CK for each agent. The content of K as it pertains to the local proxy, e.g., roles for the local proxy, govern the behavior of that team member. The details of

how a role is executed by the control agent, i.e., the UAV, are domain- (and even team member-) dependent.

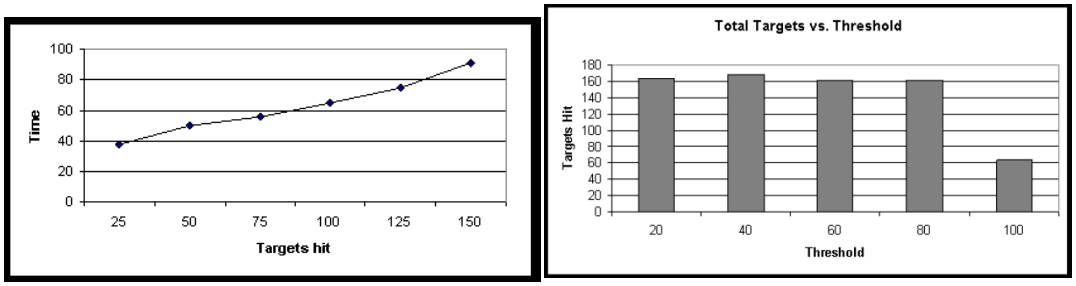
5 Experimental Results

In this section, we present empirical evidence of the above approach with a combination of high and low fidelity experiments.



(a) **(b)**

Fig. 4. Coordinating 200 agents in (a) disaster response simulation (average on y-axis, fires, extinguished, conflicts and messages per agent on x-axis); and (b) the number of fires extinguished by 200 fire trucks versus threshold.



(a) **(b)**

Fig. 5. Simulated coordinating 200 UAVs in a battlespace (a) time vs the number of targets hit and (b) the number of targets hit versus threshold.

5.1 Machinetta

In Figures 4 and 5, we show the results of an experiment using 200 Machinetta proxies running the coordination algorithms described in Section 3.

These experiments represent high fidelity tests of the coordination algorithms and illustrate the overall effectiveness of the approach. In the first experiment, the proxies control fire trucks responding to an urban disaster. The trucks must travel around an environment, locate fires (which spread if they are not extinguished) and extinguish them. The top level goal of the team, G, was to put out all the fires. A single plan requires that an individual fire be put out. In this experiment, the plan included only one function, which was to put out the fire. We varied the sensing range of the fire trucks ("'Far"' and "'Close"') and measured some key parameters. The most critical thing to note is that the approach was successful in coordinating a very large team. The first column compares the number of fires started. The "'Close"' sensing team required more searching to find fires, and as a result, unsurprisingly, the fires spread more. However, they were able extinguish them slightly faster than the "'Far"' sensing team, partly because the "'Far"' sensing team wasted resources in situations where there were two plans for the same fire (see Column 3, "'Conflicts"'). Although these conflicts were resolved it took a nontrivial amount of time and slightly lowered the team's ability to fight fires. Resolving conflicts also increased the number of messages required (see Column 4), although most of the differences in the number of messages can be attributed to more fire fighters sensing fires and spreading that information. The experiment showed that the overall number of messages required to effectively coordinate the team was extremely low, partially due to the fact that no low- level coordination between agents was required (given the one fire truck per plan). Moreover, we varied the thresholds corresponds to the maximum distances the truck will travel to a fire and 4(b) shows increasing thresholds initially improves the number of fires extinguished, but too high a threshold results in a lack of trucks accepting tasks and a decrease in performance.

In the second domain, figure 5(a) shows high level results from a second domain using exactly the same proxy code. The graph shows the rate at which 200 simulated UAVs, coordinated with Machinetta proxies, searched a battle space and destroyed targets. Moreover, figure 5(b) shows while we have effectively allocated tasks across a large team, thresholds (correspond to the maximum distances UAVs can hit a target) are of no benefit. Taken together, the experiments in the two domains show not only that our approach is effective at coordinating very large teams, but it also suggests that it is reasonably general.

5.2 Information Sharing

We test our information sharing algorithm by using a team with 400 agents and each of them has, on average, four acquaintances. One agent is randomly chosen as the source of some information and another is randomly picked as the sink for that information. The sink agent first sends out 20 messages containing relative information j, each with MAX_STEPS=50. Then the source agent sends out a message with information i with $rel(i, j)$ varied. We measure

how many steps or messages that it takes i to be encapsulated into message and sent to get to the sink agent. In our experiments, four different types of acquaintance network topologies are involved: two dimension grid networks, random networks, small world networks, and scale free networks. The small world network is based on the grid network with 8% links randomly changed. The key difference between the random network and the scale free network is that the random has a "flat" degree distribution but the scale free network has a power law distribution. Each point on each graph is based on the average of 1000 runs in a simple simulation environment.

Information sharing with different information relevance

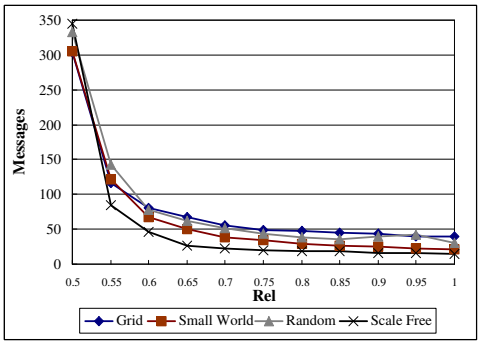


Fig. 6. The number of messages dramatically reduces as the association between information received and information to be sent increases.

We first verify our basic algorithm in different types of acquaintance network topologies. In Figure 6, we show the average number of steps taken to deliver i as we varied the strength of the relationship between the information originally sent out by the sink agent and the information i sent by the source agent from 0.5 to 1. As expected, our algorithm works on the four different acquaintance networks; further, the stronger the relationship between originally sent information and the new information the more efficient is the information delivery.

Information sharing with different number of previous messages

Next, we look in detail at exactly how many messages must be sent by the source to make the delivery from the sink efficient. We use the same settings as above except the number of messages the sink sends out is varied and the relationship between these messages and i, rel (i, j) is forced at 0.9. Notice

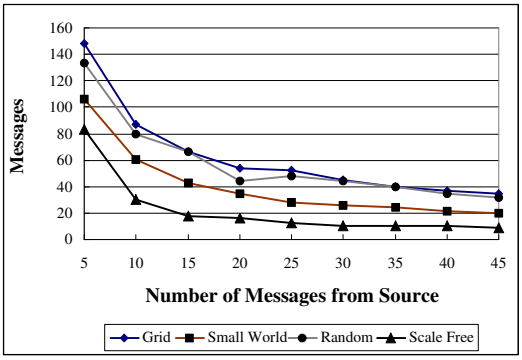


Fig. 7. The number of messages reduces as the related previous messages increased.

that only a few messages are required to dramatically impact the number of messages required. This result also shows us that a few messages is enough for agents to make a "precise guess" about where to send messages.

The influence of average acquaintances

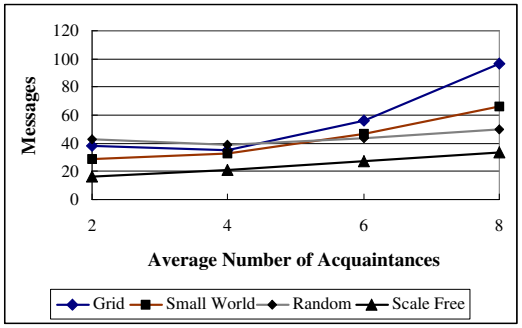


Fig. 8. The number of messages increases sligthly if each agent has more average acquaintances in acquaintance networks.

In next experiment, we looked in detail at exactly how the number of acquaintances can help to make the information sharing efficient. We run experiments with $rel(i, j) = 0.8$ and in acquaintance networks in which each agent has an average of from 2 to 8 acquaintances. The result in Figure 8 shows that the greater the number of acquaintances, the more messages that are necessary to deliver i. This means that information sharing cannot be

enhanced by connecting agents with more acquaintances. Moreover, in our experiment, we don't consider the limitation of communication breadth for agent members.

Algorithm efficiency among different size teams

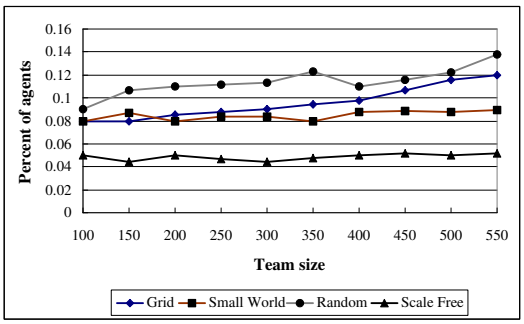


Fig. 9. Information sharing algorithm works even slightly better on large scale teams according to the measure of percentage.

To investigate the influence of team scale on information sharing performance, as shown in Figure 9, we ran experiments using different sizes of agent teams, from 100 to 550 with rel(i,j)=0.7. The information sharing efficiency is measured as the percentage of agents involved for information sharing use $percentage = \frac{agents\ involved\ in\ fodelivery}{Total\ \#\ of\ agent\ team}$. The experiment result shows that with different team sizes, the efficiency of information sharing is almost the same. This indicates that the team size is not a factor for information sharing efficiency.

5.3 Plan Deconfliction

We use TeamSim, a simple simulator, to analyze the effect our acquaintance model with dynamically changing subteams. TeamSim, which runs the coordination algorithm without simulating time intensive communication, quickly evaluates different combinations of parameter settings on the order of thousands. These parameters settings, which correspond to various domains, include free parameters based on our model and domain parameters. Free parameters are specific to our algorithm and include the acquaintance network density, and plan instantiation rule. A few of the domain parameters included team size, total preconditions, and roles per plan (see Figure 10). Our algorithm is based on the fact that the acquaintances network will detect conflicts

Parameter	Minimum	Maximum	Parameter Type
Number of Team Members	10	999	Domain Dependent
Number of Plan Templates	1	20	Domain Dependent
Roles Per Team Member	1	1	Domain Dependent
Total Preconditions	20	219	Domain Dependent
Preconditions Per Plan	1	10	Domain Dependent
Roles Per Plan	1	5	Domain Dependent
Number of Capability Types	2	21	Domain Dependent
Percent Capable	0.1	1.1	Domain Dependent
Instantiate Rate	0	1	Input (Free Parameter)
New Precondition Rate	0.0020	0.5020	Domain Dependent
Precondition Detection Rate	0.0020	0.2020	Domain Dependent
Associate Network Density	2	16	Input (Free Parameter)
Information Token	1	10	Input (Free Parameter)
Instantiation Rule*	1	3	Input (Free Parameter)
Percentage Possible	0	100	Output
Reward	0.00	85.35	Output
Messages per agent	0.10	1977.38	Output

*Instantiation Type(1-Always 2-Local 3-Probabalistic)

Fig. 10. Parameter Table

with a high probability. As team size is scaled, we can assume that the number of duplicate plan will also increase. This is shown in Figure 11 where the average number of plans increases with respect to team size using the probabilistic instantiation rule. In the graph, both the actual and expected conflicts are shown. Figure 12 shows a non-linear relationship between an input parameter, team size and an output parameter, and messages per agent.

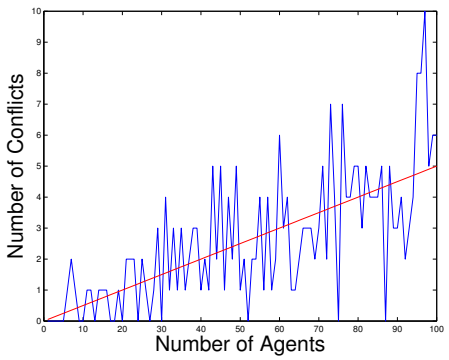


Fig. 11. The average number of plan conflicts increases with respect to team size

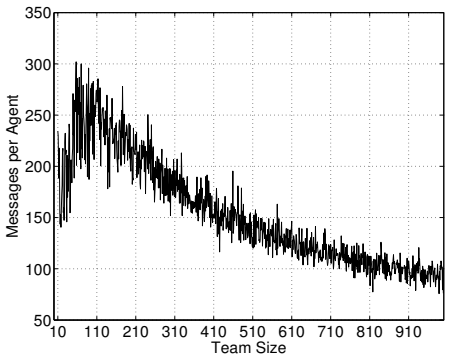


Fig. 12. Messages per Agent as Team Size is increased

6 Summary

In this paper, we have presented an approach to building large teams that has allowed us to build teams of an order of magnitude larger than those discussed in previously published work. To achieve these unprecedented scales, fundamentally new ideas were developed and new, more scalable algorithms were implemented. Specifically, we presented an approach to organizing the team based on an acquaintance network with dynamically evolving subteams. Potentially inefficient interactions between subteams were detected by sharing information across a network independent of any subteam relationships. We leveraged the social network properties of these networks to very efficiently share domain knowledge across the team. While much work remains to be done to fully understand and be able to build large teams, this work represents a significant step forward.

Acknowledgments

This research was supported by AFSOR grant F49620-01-1-0542 and AFRL/MNK grant F08630-03-1-0005.

References

1. Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 2002.
2. Albert-Laszla Barabasi and Eric Bonabeau. Scale free networks. *Scientific American*, pages 60–69, May 2003.
3. Mark H. Burstein and David E. Diller. A framework for dynamic information flow in mixed-initiative human/agent organizations. *Applied Intelligence on Agents and Process Management*, 2004. Forthcoming.

4. Hans Chalupsky, Yolanda Gil, Craig A. Knoblock, Kristina Lerman, Jean Oh, David V. Pynadath, Thomas A. Russ, and Milind Tambe. Electric Elves: Agent technology for supporting human organizations. *AI Magazine*, 23(2):11–24, 2002.
5. Eithan Ephrati, Martha Pollack, and Sigalit Ur. Deriving multi-agent communication through filtering strategies. In *Proceedings of IJCAI '95*, 1995.
6. Alessandro Farinelli, Paul Scerri, and Milind Tambe. Building large-scale robot systems: Distributed role assignment in dynamic, uncertain domains. In *Proceedings of Workshop on Representations and Approaches for Time-Critical Decentralized Resource, Role and Task Allocation*, 2003.
7. Joseph Giampapa and Katia Sycara. Coversational case-based planning for agent team coordination. In *Proceedings of the fourth International coference on Case-based Reasoning*, 2001.
8. C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-agent Systems*, 2003.
9. Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom*, pages 56–67, 2000.
10. N. Jennings. The archon systems and its applications. Project Report, 1995.
11. N. R. Jennings. Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *Intl. Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.
12. Nick Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75:195–240, 1995.
13. Kam-Chuen Jim and C. Lee Giles. How communication can improve the performance of multi-agent systems. In *Proceedings of the fifth international conference on Autonomous agents*, 2001.
14. Hiraoki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.
15. Yong-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *MobiCom*, pages 66–75, 1998.
16. Elizabeth Liao, Paul Scerri, and Katia Sycara. A framework for very large teams. In *AAMAS04 Workshop on Coalitions and Teams*, 2004.
17. M.E.J.Newman. The structure and function of complex networks. SIAM Review, Vol. 45, No. 2, 2003.
18. S. Milgram. The small world problem. In *Psychology Today, 22*, 1967.
19. R. Nair, T. Ito, M. Tambe, and S. Marsella. Task allocation in robocup rescue simulation domain. In *Proceedings of the International Symposium on RoboCup*, 2002.
20. K. Sycara P. Scerri and M Tambe. Adjustable autonomy in the context of coordination. In *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*, 2004.
21. David Pynadath and Milind Tambe. Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, 2002.

22. David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems, Special Issue on Infrastructure and Requirements for Building Research Grade Multi-Agent Systems*, page to appear, 2002.
23. D.V. Pynadath, M. Tambe, N. Chauvat, and L. Cavedon. Toward team-oriented programming. In *Intelligent Agents VI: Agent Theories, Architectures, and Languages*, pages 233–247, 1999.
24. P. Scerri, D. V. Pynadath, L. Johnson, Rosenbloom P., N. Schurr, M Si, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.
25. Paul Scerri, Yang Xu, Elizabeth Liao, Justin Lai, and Katia Sycara. Scaling teamwork to very large teams. In *AAMAS*, pages 888–895, 2004.
26. Katia Sycara and Micheal Lewis. Team cognition. In *Chapter Intelligent Agents into Human Teams. Erlbaum Publishers*, 2003.
27. Katia Sycara, Anandeep Pannu, Mike Williamson, and Keith Decker. Distributed intelligent agents. *IEEE Expert: Intelligent Systems and thier applications*, 11(6):36–45, December 1996.
28. Milind Tambe. Agent architectures for flexible, practical teamwork. *National Conference on AI (AAAI97)*, pages 22–28, 1997.
29. Milind Tambe, Wei-Min Shen, Maja Mataric, David Pynadath, Dani Goldberg, Pragnesh Jay Modi, Zhun Qiu, and Behnam Salemi. Teamwork in cyberspace: using TEAMCORE to make agents team-ready. In *AAAI Spring Symposium on agents in cyberspace*, 1999.
30. Duncan Watts and Steven Strogatz. Collective dynamics of small world networks. *Nature*, 393:440–442, 1998.
31. Yang Xu, Mike Lewis, Katia Sycara, and Paul Scerri. Information sharing in large scale teams. In *In AAMAS 2004 workshop on Challenges in the Coordination of Large Scale Multi Agents Systems*, 2004.
32. P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.
33. J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz. Cast: Collaborative agents for simulating teamwork. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1135–1142, 2001.
34. Bin Yu, Paul Scerri, Katia Sycara, Yang Xu, and Michael Lewis. Proactive information delivery and fusion in mobile sensor networks. In *submitted to IPSN*, 2005.
35. Feng Zhao and Leonidas Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann Publishers, 2004.