Scaling Teamwork to Very Large Teams

Paul Scerri*, Yang Xu+, Elizabeth Liao*, Justin Lai* and Katia Sycara*
* Carnegie Mellon University, + University of Pittsburgh
pscerri@cs.cmu.edu, xuy3@pitt.edu, eliao@andrew.cmu.edu, guomingl@andrew.cmu.edu, katia@cs.cmu.edu

Abstract

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 we apply teamwork to the problem of building teams with hundreds of members, fundamental limitations become apparent. We have developed a model of teamwork that addresses the limitations of existing models as they apply to very large teams. A central idea of the model is to organize team members into dynamically evolving subteams. Additionally, we present a novel approach to sharing information, leveraging the properties of* small worlds networks*. The algorithm provides targeted, efficient information delivery. We have developed domain independant software proxies with which we demonstrate teams at least an order of magnitude bigger than previously published. Moreover, the same proxies proved effective for teamwork in two distinct domains, illustrating the generality of the approach.*

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[6, 9]. Using basic teamwork ideas, many systems have been successfully implemented, including teams supporting human collaboration[2, 17], teams for disaster response[12], for manufacturing[9], for training[19] and for games[11]. While such teams have been very successful, their size has been severely limited. To address larger and more complex problems, we need teams that are substantially bigger but 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 joint activity and of other members of the team. Team members use these models to reason about actions that will aid the achievement of joint goals[8, 19]. However, when the size of a team is scaled up, it becomes infeasible 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[19] cannot be applied nor can Joint Intention's reasoning about committments[8].

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 subgoals 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 in 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 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 subteam: 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 *associates network*. This network connects *all agents in the team and is independent of any relationships due to subteams*. Specifically, the network is a *small worlds network* [20](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 associates network informed is low, due to the small worlds properties of the network, there is high probability for every possible pair of plans some agent will know of both and, thus, can identify inefficiencies due to conflicts between 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.

A side effect of limiting models of joint activities to the members of a subteam is that the overall team loses the ability to leverage the sensing abilities of *all* of 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[7, 22]. 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, [4, 10, 14, 21], current algorithms for sharing information in teams either require infeasibly accurate models of team activities, e.g., STEAM's decision theoretic communication[19], or require that centralized information *brokers* are kept up to date[18, 1] leading to potential communication bottlenecks. We have developed a novel information sharing algorithm that leverages the small worlds properties of the associates network to allow agents to deliver information efficiently despite not knowing who else needs it. The key idea is that each team member builds a model of which of their neighbors in the associates network will most likely either want a particular piece of information or will know who does. These models are inferred from other coordination messages, e.g., for role allocation, and do not require additional communication. Agents then simply propogate information according to this model.

To evaluate our method for building large teams, we have implemented the above approach in software proxies[15] called Machinetta. A proxy encapsulating coordination algorithms works closely with a "domain level" agent and coordinates with other proxies. Although Machinetta proxies build on the successful TEAMCORE proxies[19] and have been used to build small teams[16], 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 exhibited effective, cohesive team behavior. Such teams are an order of magnitude bigger than previously published proxy-based teams[16], hence they represent a significant step forward in building big 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.[1]

2. Building Large Teams

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. Team members detect events in the environment that result in plans to achieve the team's top level goal. The team finds sub-teams 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. Organizing Large Teams

A team A consists of a large number of agents, $A = \{a_1, a_2, ..., a_n\}$. The *associates network* arranges the whole team into a small worlds network defined by $N(t) = \bigcup_{a \in A} n(a)$, where $n(a)$ are the *neighbors* of agent a in the network. The minimum number of *agents* a message must pass through to get from one agent to another via the associates network is the *distance* between those agents. For example, as shown in Figure 1, agents a_1 and a_3 are not neighbors but share a neighbor, hence $distance(a_1, a_3) = 1$. We require

1 A small amount of code was changed to interface to different domain agents.

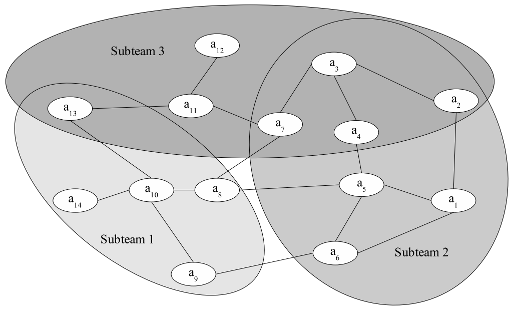


Figure 1. Relationship between subteams and the associates network

that the network be a small worlds network, which imposes two constraints. First, $\forall a \in A, |n(a)| < K$, where K is a small integer, typically less than 10. Second, $\forall a_i, a_j \in A, distance(a_i, a_j) < D$ where D is a small integer, typically less than 10.

Plans and Subteams

The team A has a top level goal, G to which the team *commits*, with the semantics of STEAM. Achieving G requires achieving sub-goals, g_i, that are not known in advance but are a function of the environment. For example, sub-goals of a high level goal to respond to a disaster are to extinguish a fire and provide medical attention to injured civilians. Each sub-goal is addressed with a plan, $plan_i =< g_i, recipe_i, roles_i, d_i >$. The overall team thus has plans $Plans(t) = \{plan_1, \ldots, plan_n\}$, though 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 executing a plan, when it detects that g_i is relevant. $recipe_i$ is a description of the way the sub-goal will be achieved[8] and $roles_i = \{r_1, r_2, r_3, ...r_r\}$ are the individual activities that must be performed in order to execute that $recipe_i$. d_i is the domain specific information pertinant to the plan. For convenience, we write $perform(r, a)$ to signify that agent a is working on role r. We are using LA-DCOP for role allocation[5] which results in a dynamically changing subset of the overall team involved in role allocation. We capture the identities of those agents involved in role allocation with $allocate(plan_i)$.

Mutual Beliefs and Subteams

Agents working on the plan *and* their neighbors in the associates network, make up the *subteam* for the plan (we write the subteam for $plan_i$ as $subteam_i$). Since allocation of team members to roles may change due to failures or changing circumstances, the members of a subteam also change. All subteam members 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. Typically, $subteam_i \cap subteam_j \neq \emptyset$. These subteams are the basis for our coordination framework and leads to scalability in teams.

We distinguish between two sets of agents within the subteam: those that are assigned to roles, $roles_i$, in the plan and those that are not. The subteam members actually assigned to roles in a plan $plan_i$, called the *role executing agents*, $REA(p_i) = \{a | a \in A, \exists r \in roles_i, perform(r, a)\}$ The non-role executing agents are called *weakly goal related agents* $WGRA(p_i) = \{a | a \in A, a \in allocate(p_i) \wedge associate(allocate(p_i)) \wedge associate(REA)\}$.

A key to scaling teamwork is the efficient sharing of information pertaining to the activities of the team members. Using the definitions of subteams, we can provide relaxed requirements on mutual beliefs, making it feasible to build much larger teams. Specifically, agents in REA_i must maintain mutual beliefs over all pieces of information in $plan_i$, while agents only in $WGRA_i$ must maintain mutual beliefs over only g_i and $recipe_i$. Maintaining these mutual beliefs within the subteam requires relatively little communication, and scales very well as more subteams are added.

Conflict Detection

Detecting conflicts or synergies between two known plans is a challenging task[3, 13], but in the context of a large team there is the critical additional problem of ensuring that some team member knows of both recipes. Here we focus on this additional challenge. When we allow an individual agent to commit the team to a goal, there is the possibility that the team may be executing conflicting plans or plans which might be combined into a single, more efficient plan. Once a conflict is detected plan termination or merging is possible due to the fact that the agents form a subteam and thus maintain mutual belief. Since it is infeasible to require that every team member know all plans, we use a distributed approach, leveraging the associates network. This approach leads to a high probability of detecting conflicts and synergies, with very low overheads.

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. A simple probability calculation reveals that the probability of overlap between subteams is:

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

where where n = number of agents, k = size of subteam A, m = size of subteam B and ${}_aC_b$ denotes a combination.

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 a conflict is detected because over time more agents are actually involved. In Section 4 we experimentally show that this technique leads to a high probability of detecting conflicts.

3. Sharing Information in Large Teams

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 big 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 that they do not know who requires. In this section, we present an approach to sharing such information, leveraging the small worlds properties of the associates network. The basic idea is to forward information to whichever acquaintance in the associates network is most likely to either need the information or have a neighbor who does.

Agents send information around the team in *messages*. A message consists of four parts, $M =< sender, i, E, count >$. The first two elements, *sender* and i, denote the agent that sent the message and the piece of information being communicated. With this algorithm, we are only interested in delivering domain level information (as opposed to coordination information). So $I = \{i_1, i_2, \ldots, i_n\}$, defines all the information that could be sent, here i is defined according to d_i in Section 2. The last two elements of a message, E and *count*, are used for improving the team's information flow (see below) and determine when to stop forwarding a message, respectively.

For the purposes of information sharing, the internal state of the team member a is represented by $S_a =< H_a, K_a, P_a >$. H_a is the (possibly truncated) history of messages received by the a. $K_a \subseteq I$ is the local knowledge of the agent. If $i \in K_a$, we say $knows(a, i) = 1$, otherwise, $knows(a, i) = 0$. Typically, individual team members will know only a small fraction of all the team knows, i.e., $|K_a| << |I|$. Our algorithms are primarily aimed at routing information in $I - K_a$, since it this information that needs to be shared. Thus, the agents are

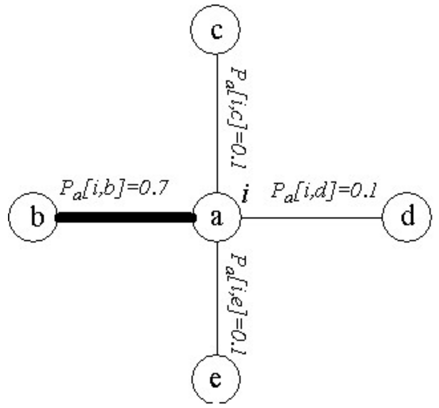


Figure 2. Probability model example

reasoning in advance about how they would route information. For example, a fire fighter might build a model of who might be interested in particular street blockages.

Since the reason for sharing information between teammates is to improve performance of a team, quantifying the importance of a piece of information i to an agent a at time t is needed. Specifically, we use the function $U : I \times A \rightarrow \mathcal{R}$. The importance of the information i is calculated by determining the expected increase in utility for the agent with the information versus without it. That is, $U(a, i) = EU(a, K_a \cup i) - EU(a, K_a)$, where $EU(a, K_a)$ is the expected utility of the agent a with knowledge K_a. When $U(a, i) > 0$, knowledge of i is useful to a, and the larger the value of $U(a, i)$ the more useful i is to a. Formally, the reward for the team is $reward(i) = \frac{\sum_{a \in A} U(a,i) \times knows(a,i)}{\sum_{a \in A} knows(a,i)}$. Notice, that since this calculation is based on knowing the use of a piece of information to each agent, agents cannot compute this locally. Thus, it is simply a metric to be used to measure algorithm performance.

The heart of our algorithm is a model of the relative probabilities that sending a piece of information to a neighbor will lead to an increase in the reward as defined by our objective function. This is P_a in the agent state. P_a is a matrix where $P_a[i, b] \rightarrow [0, 1], b \in N(a), i \in I$ represents the probability that neighbor b is the best neighbor to send information i to. 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. This situation is illustrated in Figure 2. To obey the rules of probability, we require $\forall i \in I, \sum_{b \in N(a)} P^t_a[i, b] = 1$.

In Algorithm 1, the function CHOOSE selects a neighbor according to which to send the message, according to the probabilities in P. Notice, that this function

```
ALGORITHM 1: INFORMATION SHARE($S_a$)
(1)  while true
(2)    $m \leftarrow getMsg$
(3)    $S_a \leftarrow \delta(m, S_a)$
(4)    if $m.count < MAX\_STEPS$
(5)      INC(m.count)
(6)      $next \leftarrow$ CHOOSE($P[i,m]$)
(7)      $m.sender \leftarrow next$
(8)      SEND(m)
```

can choose any neighbor, with likelihood proportional to their probability of being the best to send to, rather than always sending to the agent with the highest probability, which leads to some additional robustness when inferences are wrong. δ is the function the agent uses to update its state when it receives a message and is defined below. As a piece of information gets propagated around the associates network, the counter is incremented. Once this counter reaches MAX_STEPS the information propagation is stopped. While this is a simple stopping condition the agent does not have enough information to do a more optimal calculation.

3.1. Building a Network Model

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. Specifically, when a message arrives, the agent state, S_a, is updated by the transition function, δ, which has four parts, δ_H, δ_K, δ_P^I and δ_P^E. First, the message is appended to the history, $\delta_H(m, H_a) = H_a \cup m$. Second, the information contained in the message is added to K_a, $\delta_H(m, K_a) = K_a \cup m.i$. The details of how δ_P^I and δ_P^E update P_a will be described below.

Intuitively, if agent a tells agent b about a fire at 50 Smith St, when agent b has 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. The basic idea is that received information can be interpreted as *evidence* for which neighbor to send other information to.

Underlying any algorithm that exploits the relationships between pieces of information must be a model of those relationships. We write this function as $rel(i,j) \rightarrow [0,1], i,j \in I$, where 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).

Applying the basic idea of Bayes' Rules, we can define δ_P^I based on a message received from b in the following way:

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

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 the equation is the most important. 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. The second case simply changes the probability of sending that information to agents other then the sender in a way that ensures the normalization works. The third case encodes the idea that you would not want to send a piece of information to an agent that sent it to you.

Consider the following example:

$$P_a^t = \begin{array}{c} \\ i \\ j \\ k \end{array} \begin{array}{c} \begin{array}{cccc} b & c & d & e \end{array} \\ \left[\begin{array}{cccc} 0.6 & 0.1 & 0.2 & 0.1 \\ 0.4 & 0.2 & 0.3 & 0.1 \\ 0.4 & 0.4 & 0.1 & 0.1 \end{array} \right] \end{array}$$

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 with information j arrives from agent b, $m =< b, j, \emptyset, 1 >$. Applying δ_P^I to P_a we get the following result:

$$P_a^{t'} = \begin{array}{c} \\ i \\ j \\ k \end{array} \begin{array}{c} \begin{array}{cccc} b & c & d & e \end{array} \\ \left[\begin{array}{cccc} 0.643 & 0.089 & 0.179 & 0.089 \\ \varepsilon & 0.333 & 0.5 & 0.167 \\ 0.211 & 0.526 & 0.132 & 0.132 \end{array} \right] \end{array}$$

The effects on P are intuitive: (i) j will likely not be sent back to b, i.e., $P_a[i,b] = \varepsilon$; (ii) the probability of sending i to b is increased because agents wanting j probably also want i; (iii) the probability of sending k to b is decreased, since agents wanting j probably do not want k.

3.2. Sharing Models to Improve Efficiency

By adding a small amount of information to each message, i.e., $e \in E$ in $M =< sender, i, E, count >$, the agents can share their models and further improve performance. Notice that there are many ways to achieve this, here we present one technique that gives good results, with low computational overhead.

Intuitively, the idea is as follows. Whenever agent b is sending a message to a it can also share part of its model, so that future information can be more effectively routed through the network. Specifically, if b decides that it is well placed to route information i can add additional information to a message to b, letting it know to send i to it, if it ever receives it. Conversely, if b knew it was not well placed to route i it could add information that told a not to send it i, if it received it. The key to the efficiency of this technique is that b is sending key parts of an accumulated model, hence with many such messages the whole team can quickly get accurate models of P_i and, thus, route information effectively.

Specifically, we can determine what e an agent b should send in a message to a in the following way. First, we sum the evidence that the agent has received from each of its neighbors about where to send each piece of information. Specifically, we calculate $Q_b^i = \sum_{d \in N(b)} \prod_{j \in K_b \, from \, d} 2 \times rel(i,j)$. The result can be interpreted as the value of routing information i through b. We choose to send the information that will provide maximum value. Specifically we send model information such that:

$$argmax_{|\cup i| \leq m} \sum_{i \in K_b} \big(\sum_{c \in (N(b)-a)} Q_b^i \big)$$

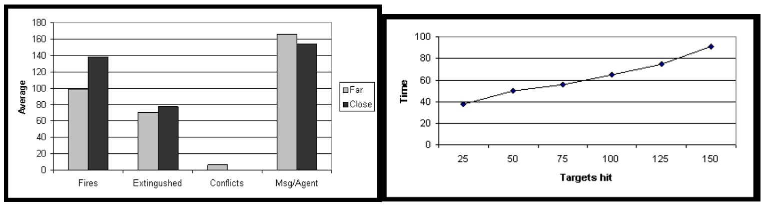
When an agent receives extra model information in the form of Q, it must update P accordingly. First we define $rel'(i, e_b(i)) = \frac{Q_b^i}{2 \times \, Q_a^i}$, as the local relationship between i and $e_b(i)$. $rel'(i,j)$ is the value of routing information i through b, from a's perspective. We use $rel'(i,j)$ as a power factor to update P_a. The we can write the update function the agent uses to update P based on e as follows:

$$\forall i,j \in I, b \in N(a) \quad \delta_P^I(P_a[i,b], m =< c, j, e_c(i), l >)$$
$$= \begin{cases} P_a[i,b] + k \times rel'(i, e_b(i)) & \text{if } b = c \\ P_a[i,b] & \text{if } b \neq c \end{cases}$$

k is a weighting factor that captures how strongly a lets P be influenced by the incoming information. The best value to use for k must be determined empirically. Then, as in the previous section, P must be normalized.

To continue the example from above,

$$P_a^t = \begin{array}{c} \\ i \\ j \\ k \end{array} \begin{array}{cccc} b & c & d & e \\ \end{array} \left[\begin{array}{cccc} 0.643 & 0.089 & 0.179 & 0.089 \\ \varepsilon & 0.333 & 0.5 & 0.167 \\ 0.211 & 0.526 & 0.132 & 0.132 \end{array} \right]$$



(a) (b)

Figure 3. Coordinating 200 agents in (a) disaster response simulation (average on y-axis, fires, extinguished, conflicts and messages per agent on x-axis); and (b) simulated UAVs in a battlespace (time on y-axis, targets hit on x-axis).

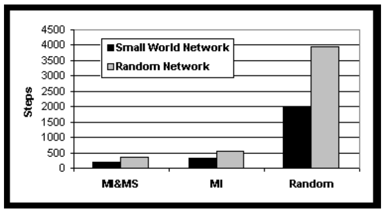
when a message $m =< b, j, \{ \, Q_b^i = 5\}, 1 >$, and $Q_a^i = 4$, $k = 0.1$
then

$$P_a^{t'} = \begin{array}{c} \\ i \\ j \\ k \end{array} \begin{array}{cccc} b & c & d & e \\ \end{array} \left[\begin{array}{cccc} 0.683 & 0.079 & 0.158 & 0.079 \\ \varepsilon & 0.333 & 0.5 & 0.167 \\ 0.211 & 0.526 & 0.132 & 0.132 \end{array} \right]$$

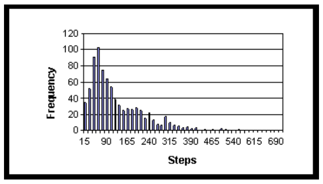
When a receives the extra information about i from b, it increases the value of sending information i to agent b, as shown in the first row of the array. As the extra information has no relationship with j and k, the second and third rows are not changed.

4. Experimental Results

In this section, we present empirical evidence of the above approach with a combination of high and low fidelity experiments. In Figures 3(a) and (b), we show the results of an experiment using 200 Machinetta proxies running the coordination algorithms described in Section 2. 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 had only one role 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



(a)



(b)

Figure 4. (a) Small worlds network vs. Random network (b) Distribution of number of steps required

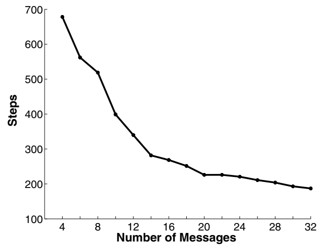


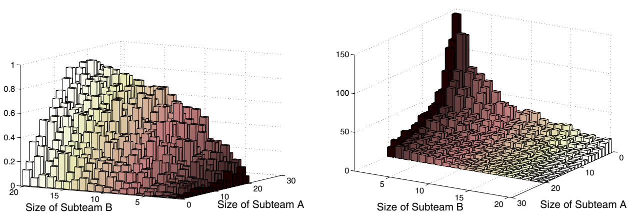
Figure 5. Association between number of relative messages and delivery time

”‘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 an 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), though 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 were required (since one fire truck per plan). Figure 3(b) 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. Taken together, the experiments in the two domains show not only that our approach effective at coordinating very large teams but also suggests that it is reasonably general.

While experiments with large teams show the feasability of the approach, it is extremely difficult to isolate specific factors affecting performance. Hence, to better understand the key algorithms we used Matlab to experiment with abstract problems. First, we tested our information sharing algorithms on very large teams using different types of network, a small worlds network and a network with random links. We arranged 32000 agents into a network and randomly picked one agent as the source of a piece of information i and another as a sink (i.e., for the sink agent $U(i)$ is very large). The sink agent sent out 30 messages with information related strongly related to i with $MAX\_STEPS = 300$. Then the source agent sent out i and we measured how long it takes to get to the sink agent. In the figure, MI indicates model inferring algorithm and MS indicate the model sharing algorithm. The result is shown in Figure 4(a). As anticipated, the two algorithms together perform best and they performance is best on a small worlds network. Using a similar setup, we then measured the variation in the length of time it takes to get a piece of information to the sink. In Figure 4(b) we show a frequency distribution of the time taken for a network with 8000 agents and $MAX\_STEP = 150$. While a big percentage of messages arrive efficiently to the sink, a small percentage get “lost” on the network, illustrating the problem with a probabilistic approach. However, despite some messages taking a long time to arrive, they all eventually did and faster than if moved at random.

Next we looked in detail at exactly how many messages must be propogated around the network to make the routing efficient (Figure 5). Again using 8000 agents we varied the number of messages the sink agent would send before the source agent sent i onto the network. Notice that only a few messages are required to dramatically affect the average message delivery time.

To understand the functionality of the associates network, simulations were run to see the effect of having associates on a dynamically changing subteam. We wanted to demonstrate that if the subteams have common members (associates), then conflicts between subteams can be detected more easily. Two subteams, each composed of 1-20 members, were formed from a group of 200. For each subteam size, members were chosen at random and then checked against the other subteam for any common team members. Figure 6a shows the calculated percentage of team member overlap when the subteam are initially formed during the simulation. This graph matches closely with the calculated probability $Pr(overlap) = 1 - \frac{_{(n-k)}C_m}{_nC_m}$. Since subteams are dynamic, in the case that both teams are mutually exclusive, an team member was chosen at random to replace a current subteam member. Figure 6b shows the average number of times that team members needed to be replaced before a common team member was found.



(a) (b)

Figure 6. (a) The probability of having at least one common agents vs. subteam size (b) The average number of times that agents need to be replaced in order to have at least one common agents

5. Summary

In this paper, we have presented an approach to building large teams that has allowed us to build teams an order of magnitude bigger than previously published. To achieve this fundamentally new ideas were developed and new more scalable algorithms implemented. Specifically, we presented an approach to organizing the team based on dynamically evolving subteams. Potentially inefficient interactions between subteams were detected by sharing information across a network independant of any subteam relationships. We leveraged the small worlds 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] M. H. Burstein and D. E. Diller. A framework for dynamic information flow in mixed-initiative human/agent organizations. *Applied Intelligence on Agents and Process Management*, 2004. Forthcoming.

[2] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric Elves: Agent technology for supporting human organizations. *AI Magazine*, 23(2):11–24, 2002.

[3] B. Clement and E. Durfee. Scheduling high level tasks among cooperative agents. In *Proc. of ICMAS'98*, pages 96–103, 1998.

[4] E. Ephrati, M. Pollack, and S. Ur. Deriving multi-agent communication through filtering strategies. In *Proceedings of IJCAI '95*, 1995.

[5] A. Farinelli, P. Scerri, and M. Tambe. Building large-scale robot systems: Distributed role assignment in dynamic, uncertain domains. In *Proc. of Workshop on Representations and Approaches for Time-Critical Decentralized Resource, Role and Task Allocation*, 2003.

[6] J. Giampapa and K. Sycara. Conversational case-based planning for agent team coordination. In *Proc. of the Fourth Int. Conf. on Case-Based Reasoning*, 2001.

[7] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proc. of AAMAS'03*, 2003.

[8] 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.

[9] N. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75:195–240, 1995.

[10] K. Jim and C. L. Giles. How communication can improve the performance of multi-agent systems. In *Proceedings of the fifth international conference on Autonomous agents*, 2001.

[11] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, , and H. Matsubara. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.

[12] R. Nair, T. Ito, M. Tambe, and S. Marsella. Task allocation in robocup rescue simulation domain. In *Proc. of the International Symposium on RoboCup*, 2002.

[13] M. Paolucci, O. Shehory, and K. Sycara. Interleaving planning and execution in a multiagent team planning environment. *Journal of Electronic Transactions of Artificial Intelligence*, May 2001.

[14] D. Pynadath and M. Tambe. Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In *Proc. of AAMAS'02*, 2002.

[15] D. V. Pynadath and M. Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *JAAMAS, Special Issue on Infrastructure and Requirements for Building Research Grade Multi-Agent Systems*, 2002.

[16] P. Scerri, D. V. Pynadath, L. Johnson, P. Rosenbloom, N. Schurr, M Si, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *Proc. of AAMAS'03*, 2003.

[17] K. Sycara and M. Lewis. *Team Cognition*, chapter Integrating Agents into Human Teams. Erlbaum Publishers, 2003.

[18] K. Sycara, A. Pannu, M. Williamson, and K. Decker. Distributed intelligent agents. *IEEE Expert: Intelligent Systems and thier applications*, 11(6):36–45, 1996.

[19] M. Tambe. Towards flexible teamwork. *JAIR*, 7:84-123, 1997.

[20] D. Watts and S. Strogatz. Collective dynamics of small world networks. *Nature*, 393:440–442, 1998.

[21] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proc. of Agents'01*, 2001.

[22] J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz. Cast: Collaborative agents for simulating teamwork. In *Proc. of IJCAI'01*, pages 1135–1142, 2001.