A Framework for Very Large Teams

Elizabeth Liao, Paul Scerri and Katia Sycara
Carnegie Mellon University
eliao@andrew.cmu.edu, pscerri@cs.cmu.edu, katia@cs.cmu.edu

Abstract

Attempts at scaling previous approaches to team behavior have been largely unsuccessful due to inflexibility of the algorithms. We have developed a framework for teams that addresses the limitations of existing teamwork models for very large teams. The central idea of the model is to organize team members into dynamically evolving subteams and require tight interactions only within subteams. A static network ensures loose coherence across the subteams. Coupled with the use of probabilistic reasoning techniques, our teamwork model allows for the creation of very large teams that can be configured for a wide range of domains.

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[3, 6, 5]. Using basic teamwork ideas, many systems have been successfully implemented, including teams supporting human collaboration[1], teams for disaster response[9], for manufacturing[7], for training[12] and for games[8]. While such teams have been very successful, their size has been severely limited, often less than 10 members. To address larger and more complex problems, teams need to be substantially bigger but retain the desirable properties of teamwork.

A 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[6, 13]. However, when the size of a team is scaled up, it becomes infeasible to maintain up-to-date, detailed models of every other teammate, 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[13] cannot be applied nor can Joint Intention's reasoning about commitments[6]. For example, without accurate models in STEAM[13], an agent may terminate a plan without first informing all other agents working on the plan. To build large teams we must implement the principles of teamwork without relying on accurate models.

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

The model organizes 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 joint activity on which they are working. At the same time, having subteams work on goals in a distributed manner can also increase the number of conflicts and duplication of effort. To ensure cohesion and minimize inefficiency across the whole team, we connect all agents in a static network. This network is independent of any relationships due to subteams. By requiring that agents also keep their neighbors in the network informed of the subgoals of subteams they are members of, there is a high probability inefficiencies can be detected and subsequently addressed. Using this model, we have been able to develop cooperative team behavior in teams up to 1,000 members.

To retain cohesive team behavior but eliminate the need for accurate models when team size is scaled up, the team is organized into dynamic subteams. Dis-

tributed plan creation is implemented by allowing individual agents to create plans on behalf of the team. Only a subset of the team members, called the subteam, work together to achieve these plans. To utilize the resources of the entire team, subteam members can change dynamically to best meet the current challenges, respond to failures, or seize opportunities. Subteams usually have overlapping members which include agents performing tasks on two different plans or agents assisting with enabling interactions between subteams.

When using distributed plan instantiation, individual members can create plans without first informing the rest of the team. A side effect is the possibility of increasing the number of conflicting plans or duplicate copies of the same plan within the team. An agent can detect conflicts by being aware of multiple subteams' activities and hence their plans. To increase the possibility that a single agent knows of multiple plans, we utilize a static network called the associates network. This network connects all agents on the team and is independent of any relationships due to subteams. Specifically, the network is a *small worlds network*[14](see Figure 2). Members in the network are connected so that any two team members are separated by a small number of agents. A neighbor of an agent in the network is called an inform associate. By requiring that agents keep their inform associates up-to-date on subteam activities, there is a high probability of conflict detection. 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 a high probability that for every possible pair of plans, some agent will know of both and, can thus identify inefficiencies. The alternate purpose of the network attempts to efficiently share information by passing domain level information to the correct team members[15].

Although the addition of the associates network enables us to detect conflicts and synergies, it is easier to initially avoid conflicts. To reduce conflict, a rule based method is used to determine when to create a plan. The techniques are the always instantiate, local information and probabilistic instantiation rules. Any remaining conflict still remaining after using these techniques for plan instantiation, will then be handled by the associates network.

In this paper, we report results of coordinating teams of 200 Machinetta proxies[13] that exhibited effective, cohesive team behavior. Such teams are an order of magnitude larger than previously published proxy-based teams[11], 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] and used a simple simulator, called TeamSim to simulate a wide range of domains and team sizes up to 1000 team members.

TeamSim allows observation of behavior resulting from several thousand domain configurations. Outputs were categorized into very good, good, bad and very bad categories. When using C4.5 to classify simulator outputs in a decision tree, several hundred rules involving almost every input parameter was created to describe our algorithm. Since no rule classified a range of a domain parameter as consistently poor, this implies our algorithm is configurable for different domains. However relationships between parameters were complex.

2. Building Large Teams

In this section, we provide a detailed model of the organization and coordination of the team. At a high level, team members detect events and details in the environment which is utilized as key information for creating plans to achieve the team's top level goal. Subteams are then formed to work on those plans. Within the subteams, agents maintain accurate models to ensure cohesive behavior. To detect conflicts across subteams, members communicate the goals of the subteams to their inform associates so that conflicts are detected with a higher probability. Depending the on the domain, the number of conflicts and duplicate plans can be very high so we then limit which agents may create plans.

2.1. Plans

The team A has a top level goal, G. The team *commits*, with the semantics of STEAM to G [13]. 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 [10]. 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.

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

Individual agents may commit the team to a sub-goal, provided that it matches a plan template. 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 executing 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 by the library. $Recipe_i$ is a description of the way the sub-goal will be achieved[6] 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 associates network. The identities of those agents involved in role allocation is captured with $allocate(plan_i)$. In the case when 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 ${}^{term}recipe_i$.

2.2. 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[4] 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 therefore 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.

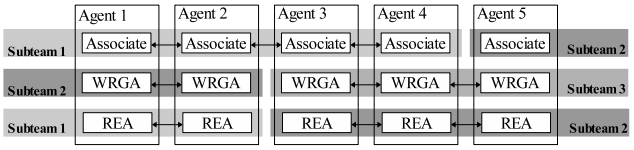


Figure 1. Agent Model

All subteam members, agents performing roles and their inform associates, 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 associates network. These subteams are the basis for our coordination framework and leads to scalability in teams.

2.2.1. Scope of individual team members 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 which are assigned to roles in $plan_i$, we call the *role executing agents*, $REA(p_i) = \{a | a \in A, \exists r \in roles_i, perform(a, r) \in perform(roles_i, m_i)\}$ The non-role executing agents are called *weakly goal related agents* $WGRA(p_i) = \{a | a \in A, a \in allocate(p_i) \wedge associate(allocate(p_i)) \wedge associate(REA)\}$. Figure 1 shows the responsibilities of five agents on three different subteams. As illustrated, an agent may be a member of more than one subteam. Agents 1-4 belong to Subteam 1, Agents 3-5 belong to Subteam 3 and all agents belong to Subteam 2.

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. Mutual beliefs requires each agent to have identical copies of the plan and the environment so that all agents are aware if team members change or a plan is nulled. 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

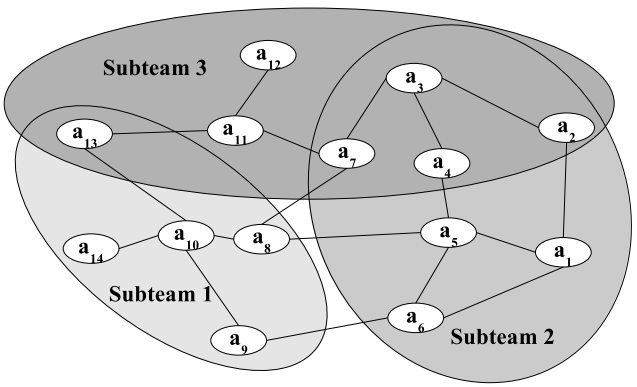


Figure 2. Subteam model

the subteam requires relatively little communication, and scales very well as more subteams are added.

2.3. 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 on the domain setting similar to the rest of our approach.

Team A consists of a large number of cooperative non-homogeneous agents, $A = \{a_1, a_2, ..., a_n\}$. An *associates network* arranges the whole team into a small worlds network defined by $N(t) = \underset{a \in A}{\cup} n(a)$, 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 (see Figure 2), agents a_1 and a_3 are not neighbors but share a neighbor, so $distance(a_1, a_3) = 1$. We require the network, N, to be a small worlds network, which imposes two constraints. First, $|n(a,t)| < K$, where K is a small integer (typically less than 10) and t is time. Second, $\forall a_i, a_j \in A, distance(a_i, a_j) < D$ where D is a small integer, typically less than 10.

Detecting conflicts or synergies between two known plans is a challenging task[2], but in the context of a large team there is the critical problem of ensuring that some team member knows of both conflicting plans. This is the main function of the associates network. We focus on this additional challenge because 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, plans which might be combined into a single, more efficient plan, or duplicate plans. Plan termination or merging of conflicting plans is possible due to fact that the team member who detected the conflict knows the details of both plans. The agent, possibly an associate of one subteam and an REA of the other, would have maintained mutual belief of both subteams. This approach leads to a high probability of detecting conflicts and synergies, with very low overhead.

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 a conflict is detected. In Section 3, we experimentally show that this technique leads to a high probability of detecting conflicts.

After a conflict is detected, the plan needs to be terminated and 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.

2.3.1. 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 rule for instantiating plans. These rules differ by 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.

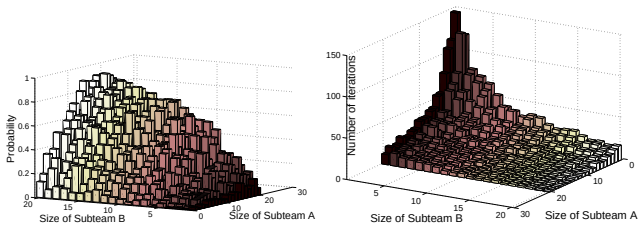
The second rule, the *probabilistic instantiation rule*, requires no knowledge of other team members. This method requires that team members wait a random amount of time before creating the plan. If during that time, it has not been informed by an inform associate that another teammate is creating the 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 plan 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 precondition 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. Results

In this section, we show an empirical evaluation of the above approach with a combination of high and low fidelity experiments.

First, to understand the functionality of the associates network, simulations were run to see the effect of having associates on a dynamic subteam. We want to show that if subteams are chosen at random, then there is a high probability of overlap between any two subteams. This overlap would then translate to improved conflict detection. Two subteams of 1-20 members each were formed of out 200 agents. For each subteam size,



(a) (b)

Figure 3. (a) The probability of having at least one common agent vs. subteam size (b)The average number of times that agents need to be replaced in order to have at least one common agent

members were chosen at random and then checked against each other for any common team members. Figure 3a shows the percent of team member overlap during the simulations. This graph matches closely with the calculated probability $Pr(overlap) = 1 - \frac{(n-k)C_m}{nC_m}$ where n = number of agents, k = size of subteam A and m = size of subteam B. If both teams are mutually exclusive, an agent was chosen at random to replace a current subteam member. Figure 3b shows the average number of times that team members needed to change before a team member overlap occurs.

TeamSim, a simple simulator, was used to analyze the effect of our model in domains with widely varying parameters. 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 would correspond to various domains, include free parameters based on our model and domain parameters. Free parameters are specific to our algorithm and include the associate network density, and plan instantiation rule. A few of the domain parameters included team size, total preconditions, and roles per plan (see Figure 4). Our algorithm is based on the fact that the associates network will detect conflicts with a high probability. In Figure 7, we show that in general, having a higher associate network density will lead to better conflict detection. As higher network density leads to larger subteams and hence a higher probability of overlap as shown above in figure 3a. In Figure 6, a comparison of plan complexity shows that more simple plans will be created. Figure 5 shows a non-linear relationship between an input parameter, team size and an output parameter, messages per agent.

In Machinetta, a proxy encapsulating coordination algorithm works closely with a domain level agent

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

*Parameter for information sharing in large teams[15]
**Instantiation Type(1-Always 2-Local 3-Probabalistic)

Figure 4. Parameter Table

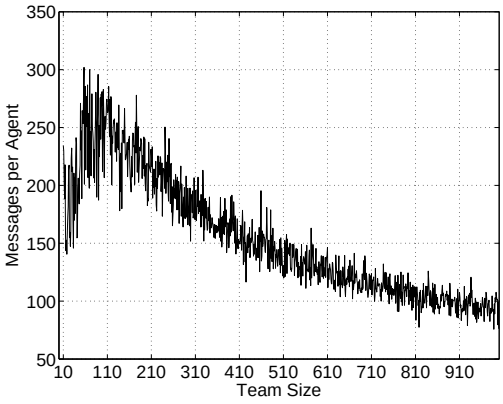


Figure 5. Messages per Agent as Team Size is increased

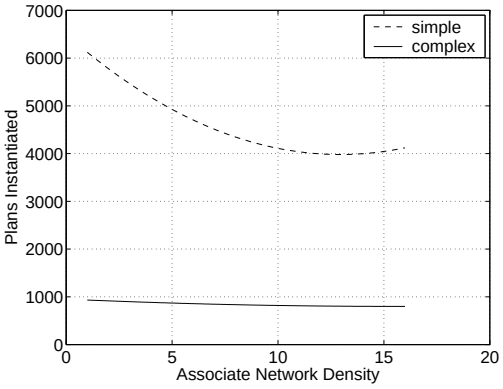


Figure 6. A comparison of simple and complex plans

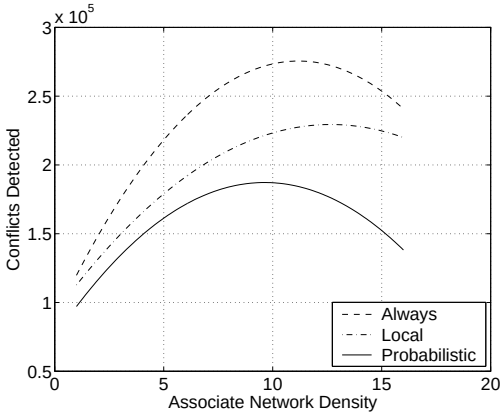


Figure 7. A comparison of the number of conflicts detected using different instantiation rules

and coordinates with other proxies. Although Machinetta proxies build on the successful TEAMCORE proxies[13] and have been used to build small teams[11], they were not able to scale to large teams without the fundamentally new algorithms and concepts described previously.

In Figure 8, we show the results of an experiment using 200 Machinetta proxies running the coordination algorithms described in Section 2. 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 required that individual fires 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 extended 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 3rd set of columns, 'Conflicts'). Although these conflicts were resolved, it took an non-trivial amount of time and slightly lowered the team's ability to fight fires. Resolving conflicts also increased the number of messages required (see 4th set of columns), though most of the differences in the number of messages can be attributed to more fire fighters sensing fires and spreading that infor-

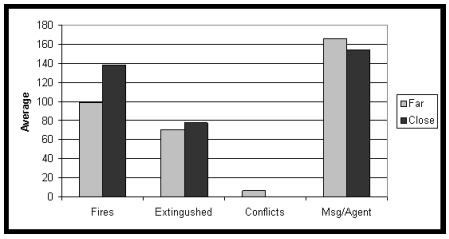


Figure 8. Machinetta Proxy Simulation

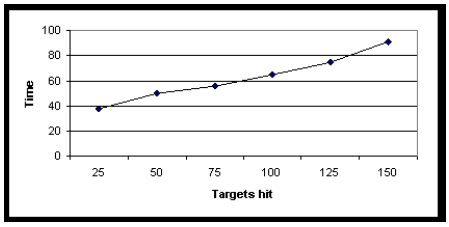


Figure 9. Simulated UAVs

mation. The experiment showed that the overall number of messages required to effectively coordinate the team was extremely low, which was partially due to the fact that no low level coordination between agents was required (as a result of using one fire truck per plan).

Figure 9 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.

4. Conclusion and Future Work

In this paper, we describe and implement a scalable framework for team coordination. With dynamic subteams and a static network, we show that cohesive team behavior can be established for team of sizes up to 1000 members. Using this model, resources of the entire team can be utilized without keeping accurate models of the entire team. Due to the distributed nature of the team, agents can independently create plans which increase the possibility of multiple copies of the same plan. Plan conflicts are minimized by a rule-based instantiation rule while the static associates network handles the conflicts that remain.

Through implementation of Machinetta proxies and TeamSim simulations, we have shown that our algorithm can be tuned to a wide variety of domains. However, the data observed shows that the relationship between input and output parameters is non-linear as shown in 5. An interesting area of investigation is to discover the cause of the non-linearity and to create an accurate model that will determine the best free parameters for a given domain.

Acknowledgments

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

References

[1] 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.

[2] B. Clement and E. Durfee. Scheduling high level tasks among cooperative agents. In *Proceedings of the 1998 International Conference on Multi-Agent Systems (ICMAS'98)*, pages 96–103, Paris, July 1998.

[3] K. Decker and J. Li. Coordinated hospital patient scheduling. In *Proceedings of the 1998 International Conference on Multi-Agent Systems (ICMAS'98)*, pages 104–111, Paris, July 1998.

[4] 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.

[5] Barbara Grosz and Sarit Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996".

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[13] Milind Tambe. Agent architectures for flexible, practical teamwork. *National Conference on AI (AAAI97)*, pages 22–28, 1997.

[14] Duncan Watts and Steven Strogatz. Collective dynamics of small world networks. *Nature*, 393:440–442, 1998.

[15] 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.