Comparing Three Approaches to Large Scale Coordination

Paul Scerri, Regis Vincent and Roger Mailler
pscerri@cs.cmu.edu, vincent@ai.sri.com, mailler@cs.umass.edu

Abstract

Coordination of large groups of agents or robots is starting to reach a level of maturity where prototype systems can be built and tested. This process of going from theory or simple simulation to actual implementation of large scale coordination is revealing new issues and encouraging new approaches. However, the field is so new that the central problems and key issues are not yet well known. In this paper, we look at three implementations of large scale coordination to find common issues, approaches and open problems with the aim of setting an agenda for future research.

1. Introduction

In the past few years there have been a small number of systems where large numbers of coordinated agents or robots have been required to perform some complex task. These applications have required completely new techniques to be developed to deal with the complexity that comes about due to the sheer scale of the group. In this paper, we look at three successful approaches to coordination to find commonalities and differences in the techniques used. The aim is to identify ideas that generalize across approaches as well as issues that appear to come up regardless of the approach used.

Each of the applications involves at least one hundred completely unselfish and cooperative group members. The group members were relatively homogeneous, although there was always heterogeneity due to location, thus despite being relatively homogeneous in design the agents were not easily interchangeable. The complex tasks on which the teams were working were relatively decomposable, although constraints (either resource or spatial or both) existed between the decomposed subtasks. In all cases, the coordination algorithms had to deal with many of the issues faced by any multi-agent system, as well as complications due to scale.

Two interesting commonalities were seen between the three approaches. The first commonality was that each approach used some form of dynamic, partial centralization to reduce the overall complexity. Specifically, some decision-making responsibility for a small group of agents was dynamically assigned to an agent particularly able to make those decisions. The form of the centralization varied greatly, from dynamic subteams to dispatchers to mediation. In each case, only a small subset of the team was involved in the centralization and the agents involved, as well as the "center", was not chosen in advance. We conclude that completely centralized coordination is infeasible but that dynamic and localized centralization is a feasible method for large scale coordination.

Likely related to the dynamic localized centralization, the second notable commonality between the three approaches was that the coordination was neither simple and relying on emergent properties nor highly structured with top-down guidance. While this was at least partially due to the way the tasks decomposed into relatively independent pieces, there was more structure in the coordination than in the task. Interestingly, none of the approaches were inspired by any particular organizational theory, human or biological. We hypothesize that some structure is required in order for the developers to be able to manage the complexity of the team, but too much structure unreasonably limits the flexibility of the group.

In contrast to the high degree of commonality between the approaches used, the problems encountered and the major open problems were varied. In two of the approaches, determining appropriate parameters for heuristics was identified as a problem. In two approaches there was unwanted emergent behavior. In one approach, sharing information was a problem. It does not appear that any of the approaches are immune to the problems encountered by the others, only that the specific problems were not induced by the specific applications. While all approaches found the problem of debugging to be time consuming and ungainly, it was achieved with sufficient effort. Though, in what we can perhaps attribute to the newness of the application areas and approaches, in several cases developers did not know there were significant bugs in the code

for some time.

In the remainder of this paper, we briefly describe the way each of the three approaches addresses a variety of problems.

2. Applications and Assumptions

In this section we describe the scope of applications and assumptions for each of the approaches. Each of the applications involves at least 100 cooperative entities and has been tested either in hardware or realistic simulation of hardware. Although specific communication restrictions differ, communication is identified as a much bigger limitation than computation. None of the applications requires optimal performance, instead the focus is on doing a large task robustly.

2.1. Teamwork and Machinetta

Machinetta software proxies are used to develop teams where the team members are assumed to be completely cooperative and willing to incur costs for the overall good of the team[11]. Typically team members will be highly heterogeneous, ranging from simple agents and robots to humans. While the team will be able to leverage reasonably high bandwidth communication channels, we assume the bandwidth is not sufficiently high to allow centralized control. The team will need to achieve complex goals in a complex, dynamic domain. We assume that some decomposition of the complex task into relatively independent sub-tasks can take place.

Given the complexity of the domains, tasks and heterogeneity of the team, we typically assume that optimality is not an option. Instead we look for satisficing solutions, that can achieve the goals rapidly and robustly. The assumption is that doing something reasonable is a very good start. For example, in a disaster response domain, we assume it is better to have fire trucks on reasonable routes to fires, than to delay departure with computationally expensive optimization. To date we have demonstrated teams of 200 software agents[12], both in a UAV simulation[10] and a disaster response simulation, but teams of up to 200,000 agents are envisioned.

2.2. Centibots dispatching

Funded by DARPA, the CENTIBOTS project is aimed at designing, implementing, and demonstrating a computational framework for the coordination of very large robot teams, consisting of at least 100 small, resource limited mobile robots (see Figure 1), on an indoor search and rescue task. In this project, communication was limited and unreliable, any coordination mechanisms had to deal with the limitations. There are only two types of agents in the Centibots system, hence heterogeneity is not an issue. Similarily, optimality is infeasible, so having a reactive, "good enough" system was the primary aim.

Figure 1. 100 Robots used during the Januaray 2004 evaluation.

In the scenario, the CENTIBOTS are deployed as an search and rescue team for indoor missions. A first set of mapping capable CENTIBOTS survey the area of interest to build and share a distributed map which highlights hazards, humans, and hiding places. A second wave of robots, with the capability to detect an object of interest (biochemical agents, computers, victims, etc.), is then sent out. The key goal the second wave is to *reliably* search *everywhere* and report any findings to the command center. These robots are then joined by a third wave (possibly being the same robots used during the second wave) of tracking robots that deploy into the area, configuring themselves to effectively sense intruders and share the information among themselves and a command center [5].

Communications is done using an adhoc wireless network, which has a maximum *shared* bandwith of 1Mpbs. Communications is not guaranteed because as the robots move to achieve their own missions, links between the agents are created and lost. Because the robots fail, break, and get lost, planning the entire mission ahead of time is not possible. In practice, it is never the case that all robots complete the mission. In addition, resources (i.e.robots) and goals can be added, removed, or disabled at anytime making an adaptable system crucial.

2.3. Cooperative Mediation

Scalable, Periodic, Anytime Mediation (SPAM) [6] is a *cooperative mediation* based algorithm that was designed to solve real-time, distributed resource allocation problems (RTDRAP). SPAM was developed to coordinate the activities of agents controlling radar based sensor platforms in order to track targets within a real-time environment (see figure 2). Other than controlling their individual sensors, each of the agents is able to take on one or more organizational roles within the sensor network. One of these roles is that of the track manager.

As the track manager, an agent coordinates the activities of enough sensor resources to repeatedly triangulate the position of their target. However, the sensors can only sense one target at a time and they are shared, so the track managers must come to an agreement about which of the sensors they get.

Adding to the complexity of this problem, communications varies from 100 Mbps TCP-based wired networks to 14.4 Kbps half duplex, RF-based, multi-channel wireless communications. In the latter, message passing was very unreliable and loss rates of 50% are not uncommon. The communication restrictions combined with the real-time coordination needs makes complete centralization out of the question and traditional distributed techniques inadequate.

SPAM has been tested in real-world hardware environments with 36 sensor agents and in simulated environments with over 500 sensor agents.

3. Key Algorithms

In this section, we describe the key algorithms and principles underlying each of the approaches. Although distinct approaches are used, teamwork, hierachical dispatching and cooperative mediation, each approach imposes some limited, flexible structure on the overall group. Notice that a central aim of each approach is to efficiently, robustly, and heuristically allocate and reallocate tasks and resources.

3.1. Machinetta and Teamwork

Teamwork is operationalized by STEAM which in turn is based on the principles of joint intentions. A key principle in STEAM is that agents both have models of teamwork and models of other team members. These models are used to reason about which actions to take to achieve team goals. Having explicit models with which the agents can reason, leads to more robustness and flexibility than fixed protocols. The team executes Team Oriented Plans, which break complex tasks down into individual *roles*, with constraints between the roles. Typically, a large team will be executing many team oriented plans at any time. Dynamically changing sub-teams form to execute each of the plans. Limited amounts of communication occur across sub-teams, to ensure that sub-teams do not act at cross purposes or duplicate efforts.

The algorithms required to perform the teamwork have been designed with two key ideas in mind. First, we use probabilistic models in all key algorithms. This actually leverages the size of the team because the probabilistic models tend to be more reliable with a large number of agents, since local variation gets canceled out more effectively. The teamwork algorithms are designed to leverage the probabilistic models to make very rapid decisions that are likely to be at least "reasonable". Second, we note that when there are very many team members, Murphy's Law[1] applies, simply due to the fact that everything happens so many times. Creating efficient, lightweight software that is simple enough to be implemented reasonably quickly, yet robust enough to be used in teams with thousands of agents, is as much a function of the algorithms as it is of the actual code. Significant emphasis must be placed on designing algorithms that are sufficiently simple to be straightforward to implement in a very robust manner. These two principles are embodied in the role allocation process, which uses a probabilistic model of the current capabilities and tasks of the team to calculate a threshold capability level that a team member for performing a role would have in a good overall allocation. A *token* representing the role moves around the team until an available team member is found with capability above the threshold[3].

3.2. The Centibots dispatching

Once the Centibots have produced a map as a bitmap image, an abstraction is needed so search goals can be created to ensure all space is searched. The abstraction is done by building a Voronoi diagram from the map and then the voronoi skeleton is abstracted into a graph. This abstraction is solely based on the sensor capabilities of a robot. Once we have all the goals generated, coordination is required to allocate them to a pool of robots.

To coordinate the robots' activities, we use a hierachical dispatching system, where robots can register with multiple *dispatching agents*, one of which is considered "preferred". Teams of robots are formed by a *commander*, and for each team, a manager or dispatcher is

1 Anything that can go wrong will go wrong.

Figure 2. Researchers work on a demonstration involving 36 sensors and 3 mobile targets.

selected. The commander assigns a set of goals to each team and the teams' dispatchers assign these to individuals robots. When a robot has finished its assigned goals, it notifies the dispatcher, marking itself available, and asks for a new goal.

A key problem for Centibots was the strategy used by a dispatcher to assign goals to robots. Since all robots started from the same starting position, the problem is to minimize the search time. This allocation is in theory similar to a multiple travel salesman problem except that there is no a priori notion of how many salesman you might have and salesman can fail at anytime during the travelling. Given these constraints, we found, after trying several techniques, that the best strategy for the dispatcher is to send the robot the farthest away for the first goal and then minimize its movement by taking the closest goals after the first one.

3.3. Cooperative Mediation

SPAM works by having one or more agents concurrently take on the role of mediator. An agent decides to become a mediator whenever it identifies a conflict with a neighbor (both scheduled a sensor for use at the same time) or it recognizes a sub-optimality in its allocation (it could achieve higher utility if it changed its sensor assignment). As a mediator, an agent solves a localized portion (or subproblem) of the overall global problem. In SPAM, this subproblem entails the agents with which the mediator shares sensor resources. As the problem solving unfolds, the mediator gathers preference information from the agents within the session which updates and extends its view and overlaps the context that it uses for making its local decisions with that of the other agents. By overlapping their context, agents understand why the agents within the session have chosen a particular value which allows the system to converge on mutually beneficial assignments.

This technique represents a new paradigm in distributed problem solving. Unlike current techniques which attempt to limit the information the agents use to make decisions in order to maintain distribution [16, 15], SPAM and more generally cooperative mediation, centralizes portions of the problem in order to exploit the speed of centralized algorithms.

The key principle that allows SPAM to be scalable is the heuristic restriction of the size of the subproblem that the mediators are able to centralize. Mediators in SPAM are only allowed to conduct sessions which include agents that they directly share resources with. Although, this prevents the search from being complete, in all but the most tightly constrained problem instances, this technique limits the amount of communication and computation that must occur within any single mediator. However, because mediators use such a myopic view of the overall problem, they often are unaware of the consequences of their actions on other agents. To combat this effect, SPAM incorporates the use of *conflict propagation* and *conflict dampening*.

Conflict propagation works by allowing any agent to take the role of mediator when they have an unresolved resource conflict or wish to improve the utility of their task. Whenever a mediator causes a problem for another agent outside of its local view, that agent gets to take over as the mediator.

Conflict dampening is very similar to the min-conflict heuristic presented in [7]. When an agent mediates, it gathers information about the impact of particular assignments from each of the agents involved in the session. This allows the mediator to choose solutions that minimize the effect of their decision making on agents outside of its view. Overall the effects of conflict propagation and dampening can be visualized as ripples in a pond that eventually die down due to the effects of friction and gravity.

4. Software

In this section, we describe the major pieces of technology, specifically software, that are used for the coordination in each of the approaches.

4.1. Machinetta and Teamwork

The teamwork algorithms are encapsulated in *software proxies*. Each member of the team works closely with its own proxy. The proxy handles all the routine coordination tasks, freeing the agent to focus on specific domain level tasks. The proxy communicates with the domain level agent (or robot or person) via an agent specific, high level protocol. Adjustable autonomy reasoning is applied to each decision, allowing either the agent or the proxy to make each coordination decision. Typically, all decisions are made by the proxy on behalf of agents or robots, but when the proxy is working with a person, key decisions can be transferred to that person. The current version of the proxies is called Machinetta and is a lightweight Java implementation of the successful SOAR-based TEAMCORE proxies. The proxies have been successfully tested in several domains including coordination of UAVs, disaster response, distributed sensor recharge and personal assistant teams. The proxy code can be freely downloaded from the web. The application dependent aspects of the proxies, specifically the communication code and the interface to the agents, are implemented as "pluggable" modules that can be easily changed for new domains, thus improving the applicability of the proxies.

4.2. Centibots

The Centibots software makes an extensive use of the Jini[13] architecture. Each robot and each key algorithm is a network service that register, advertise and interact independently of its physical location. We have services like the map publisher which aggregates data from the mappers and publishes a map for the other robots, like the dispatcher which allocates task to robots or even the user interface. The result is very modular, scalable infrastructure. Each robot has its own computer where it runs localization, navigation, path planning and vision processing algorithms.

4.3. Cooperative Mediation

The SPAM protocol is implemented both within simulation and as part of more complex agents designed to work on sensor hardware. The protocol itself is composed of several finite state machines (FSMs) that are written in Java. Each state in the FSM encapsulates a non-decomposable decision point within the protocol. Transitions between states are event-driven and allow to protocol to specify state transitions based on time-outs, message traffic, specific execution conditions, etc. This allows to protocol to be time and resource aware, modifying its behavior based on the current environmental conditions.

SPAM is currently being considered for use in a number of domains including real-time airspace deconfliction and the control of sensors for severe weather tracking.

5. Key Unexpected Challenges

In this section we describe the challenges that were encountered during development but were not expected at the outset. Interestingly, each approach ran into different, unexpected problems, ranging from sharing information to controlling oscillations.

5.1. Machinetta and Teamwork

Two main unexpected challenges occurred during the development of large teams. First, it is often the case that some team member has information that may be relevant to some other member of the team, but it does not know to which other team member the information is relevant. For example, in a disaster response domain, the agent may get information about chemicals stored in a particular factory, but not know which fire fighters will be attending that fire. Encapsulating knowledge of current activities to within a subteam reduces the ability of other team members to provide potentially relevant information. Previous approaches, including blackboards, advertisement mechanisms and hierarchies, do not immediately solve this problem in a manner that can effectively scale.

The second key problem is that there are many algorithm parameters that interact with one another in highly non-linear ways. Moreover, slightly different situations on the ground require substantially different configuration of the algorithm parameters. Currently, determining appropriate values for all parameters for a given domain is as much art as science and typically requires extensive experimentation. When the situation changes significantly at runtime, an initially appropriate configuration of algorithm parameters can end up being poor. Techniques that allow the team to reconfigure these parameters online is a major open problem.

5.2. Centibots challenges

The two main challenges we had to face are the instability of the communications and the number of goals to be assigned per agent. In this project, the communication was coordinated assuming a very conservative range for the wireless network. Unfortunately we have encountered more than once part of buildings where this conservative distance was not working. In this case, any robot who enters this communication dead zone will not be able to contact the centralized dispatcher. Our solution was to have the dispatcher living on robots close-by, which was a good improvement but didn't completely solve the problem. As a result, we had to implement a low-level behavior where the robot, after waiting a known timeout, will return to its original starting position if it couldn't contact the dispatcher. In this case, at least we would retrieve them.

The second challenge was to determine the number of goals to assigned to a robot. There was no way to know a priori how many robots would be part of the mission, therefore a fair division of the number of goals was not possible. We have seen in section 4.2 we have shown that the most effective dispatching would require an assignment of several close-by goals, the key question is how many. Since the number of robots assigned to the mission is unknown (robots assigned will break and the commander may reassign others in the middle of the mission), the solution we use is an empirical function. The number of goals assigned vary (between 7 to 1) depending of the number of goals left to assigned. At the end of each run we collect the number of goals fulfilled by each robot and we collect each ending time, if there is a large variation (meaning some robots were under-utilized and others were over-utilized) we vary the total number of goals to be assigned.

5.3. Cooperative Mediation

Because the SPAM protocol operates in a local manner, a condition known as oscillation can occur. Oscillation is a condition that is caused by repeated searching the same parts of the search space because of the limited view that the agents maintain through the problem solving process.

We explored a method in which each mediator maintained a history of the sensor schedules that were being mediated over whenever a session terminated. By doing this, mediators were able determine if they have previously been in a state which caused them to propagate in the past. To stop the oscillation, the propagating mediator lowered its solution quality to force itself to explore different areas of the solution space. It should be noted that in certain cases oscillation was incorrectly detected using this technique, which resulted in having the mediator unnecessarily lower its objective level.

This technique is similar to that applied in [8], where a *nogood* is annotated with the state of the agent storing it. Unfortunately, this technique does not work well when complex interrelationships exist and are dynamically changing. Because the problem changes continuously, previously explored parts of the search space need to be constantly revisited to ensure that an invalid solution has not recently become valid. Currently, we allow the agents to enter potential oscillation, maintaining no prior state other than objective levels from session to session and rely on the environment to break oscillations through the movement of the targets, asynchrony of the communications, timeouts, etc.

6. Key Novel New Ideas

In this section we describe the new ideas that were required specifically to scale up the number of agents involved in the coordination. Specifically, we show the ideas that were required to overcome weaknesses in the principles as approaches were scaled from small numbers of agents to large numbers.

6.1. Machinetta and Teamwork

There are a variety of novel ideas in the Machinetta proxies. To maintain cohesion and minimize conflicted effort, the whole team is connected via a static, scale free *associates network*[12]. As well as obligations to communicate information to members of their dynamically changing sub-team, an agent must keep its neighbors in the associates network appraised of some key information. The network allows most conflicted or duplicated efforts to be quickly and easily detected and resolved. Movement of information around the team, when team member(s) requiring the information are not known in advance, also leverages the associates network. Every time information is communicated, the agent receiving the information updates a model of where it might send other information, based on information received to date. Because of a phenomena known as *small worlds networks*, information passed around a network in this manner can be efficiently sent to the agent(s) requiring the information.

Allocating roles in team oriented plans to best leverage the current skill set of the team is accomplished by a novel algorithm called LA-DCOP[3]. LA-DCOP extends distributed constraint optimization techniques in several ways to make it appropriate for large, dynamic

teams. Most importantly, LA-DCOP uses probabilistic models of the skills of the team and the current roles to be filled to estimate the likely skill of an agent filling a role in a "good" allocation. To take advantage of human coordination reasoning, when it is available, we represent all coordination tasks explicitly as *coordination roles* and allow the proxy to meta-reason about the coordination role[11]. For example, in a disaster response domain, there may be a role for fighting some particular fire that no fire fighter is able to fill, the proxies can recognize this and send the role to some person and allow that person to determine what action to take.

6.2. Centibots

The hierarchical dispatching model offers two key interesting qualities. On the positive side, the communication is minimal since the dispatcher is eavesdropping on the status message. Assuming the status message is required then using a centralized dispatching will outperform any distributed methods. The drawback is the need of communication between the team of robots and the dispatcher. We assume that the dispatcher is a network service that reside physically anywhere on the network, the dispatcher can be running on any team member and would only require local communication. In this configuration, we could have a hierarchy of dispatcher, each responsible for an area of the map, using a sub team of robots. Each robot can already subscribe to several dispatchers. If a dispatcher has completed all its goals, then it can release its assets for other dispatchers to use.

6.3. Cooperative Mediation

The SPAM algorithm has several key features that deal specifically with the question of scale. The first is to limit the scope of the problems that the agents work on. Although not a new idea, by restricting the context that agents use to make decisions, their problem solving becomes more focused.

The second is to the use of conflict propagation and dampening. When combined together, the effects of making local changes travel a very short distance from the source. This focuses of the problem solving effort within the areas that have interdependencies that can benefit the most.

SPAM also incorporates a number of resource-aware mechanisms which prevent it over-utilizing communications. In particular, SPAM monitors the state of the communications links between itself and other agents and when it notices that one of the agents in the session has become over-burdened, it is dropped from the session. In fact, if the mediator notices that it has become a communication hot-spot, then it avoids going into session entirely.

7. Open Problems

In this section we describe the key open problems for each of the approaches. As with the unexpected problems, each approach has different open problems. Importantly, even though most of the problems appear to be reasonably approach independent, e.g., traffic control in Centibots, neither of the other approaches has specific solutions to that problem, suggesting that the problems may be general ones.

7.1. Machinetta and Teamwork

Despite its successes, Machinetta has some critical limitations. Most critically, Machinetta relies on a library of pre-defined team oriented plan templates. While some constructs exist for expressing very limited structure in the plans, these constructs are hard to use. In practice, to write successful Machinetta plans, the domain must be easily decomposable into simple, relatively independent tasks. The ability to write and execute more complex plans is a pressing problem.

While the probabilistic heuristics used by Machinetta are typically very effective and efficient, occasionally an unfortunate situation happens and the resulting coordination is very poor. Sometimes the coordination will be unsuccessful or expensive because the situation is particularly hard to handle, but sometimes it will be that the particular heuristic being used is unsuited to the specific situation. Critically, the agents themselves cannot distinguish between a domain situation that is difficult to handle and a case where the coordination is failing. For example, it is difficult for a team to distinguish between reasonable role allocation due to a dynamic and changing domain and "thrashing" due to a heuristic not being suited to the problem. While individual problems, such as thrashing, can be solved on an adhoc basis, the general problem of having the team detect that the coordination is failing is important before deploying teams. If such a problem was detected, the agents may be able to reconfigure their algorithms to overcome the problem. However, as mentioned above, determining how to configure the algorithms for a specific situation, is also an open problem.

7.2. Traffic Control in Centibots

Linked to the goal assignment, traffic control for several dozen of robots in a small environment is a huge challenge. The assignment should take into consideration the schedule in which each robot will do their tasks to prevent deadlocks. For a robot, a doorway is a very narrow choke-point and only one robot can go through at one time. When more than two robots try to enter and exit the same room at the same time, you have a conflict. Currently we are not managing this problem, luck and local avoidance is how we solve it. We have seen in our dozen of real life experimentation some conflict becoming literally traffic jam and blocking permanently one access of the area. The only reasonable way is to reason about the choke point as resources and solve the conflict during the assignment using methods like SPAM.

7.3. Cooperative Mediation

The most interesting open questions for the SPAM protocol deals with the when, why, and whom for extending the view of the mediators given different levels of environmental dynamics and interdependency structures. Because the optimality and scalability of the protocol is so strongly tied to the not only the size, but the characteristics of the subproblem that the mediators centralize, a detailed study needs to be conducted in order to understand the relationship between these two competing factors. Some work has already been done that preliminarily address these questions. For example, the whom and why to link questions were in part addressed in the *texture measures* work of Fox, Sadeh, and Baycan [4]. In addition, recent work on phase transitions in CSPs [1, 2, 9] in part addresses the question of when. It is clear that a great deal of work still needs to be done.

8. Evaluation and Metrics

In this section, we describe the techniques for evaluating our algorithms and the metrics used to measure performance. All authors agree this is an immature and difficult science. Clearly, useful and comparable metrics will need to be developed, if sensible comparison is to be performed.

8.1. Machinetta and Teamwork

Evaluating teamwork is a very difficult task. While success at some particular domain-level task is clearly a good sign, it is a very coarse measure of coordination ability and thus, it is only one aspect of our evaluation. To ensure that we are not exploiting some feature of the domain, we have endeavored to use at least two distinct domains for testing. Typically it is infeasible to test head-to-head against another approach, hence we are limited to varying parameters in the proxies. For the larger teams, a single experiment takes on the order of an hour, severely limiting the number of runs that can be performed. Unfortunately, due to the sheer size of the environment and the number of agents, there tends to be high variation in performance, implying that many runs need to be performed to get statistically significant results. Even determining what to measure in an experiment is a difficult decision. We measure things like number of messages, number plans created, roles executed and scalability, although it is not clear how some of these numbers might be compared to other algorithms. Typically, we measure global values, such as the overall number of messages rather than local values such as the number of messages sent by a particular agent.

Since there are not modeling techniques available for mathematically analyzing the algorithms performance, we have developed a series of simple simulators that allow specific algorithms to be tested in isolation and very quickly. These simulators typically also allow comparison against some other algorithms. Currently we have simple simulators for role allocation, sub-team formation and information sharing. Performing very large numbers of experiments with these simulators we are able to understand enough about the behavior of the algorithms to perform much more focused experimentation with the complete Machinetta software.

8.2. Centibots evaluation

This project was driven by the challenge problem set by DARPA and in this sense the evaluation was independently done by a DARPA team that has measured the behaviors of the Centibots software to solve the search and rescue mission not purely the coordination. For a week in January 2004, the Centibots were tested at a $650m^2$ building in Ft. A.P. Hill, Virginia. We were tested under controlled conditions, with a single operator in charge of the robot team.

For searching, the evaluation criteria were time to locate OOI(s), positional accuracy, and false detections. There were four evaluation runs, and the results are shown in the Table 1. They show that the team was highly effective in finding the object and setting up a guard perimeter. Note that we used very simple visual detection hardware and algorithms, since we had limited computational resources on the robots – false and

missed detections were a failure of these algorithms, rather than the spatial reasoning and dispatching processes.

The results were not focused on the coordination portion but measured the overall performance of the system to solve the search and rescue mission. As explained in the next section, extracting meaningful data from such system is not an easy task.

8.3. Cooperative Mediation

The SPAM protocol was implemented and tested within a working sensor network but, most the development and analysis of the protocol has been done in simulation.

The primary metrics used to measure SPAM were the number of targets being effectively tracked during a fixed period of time, the number of messages being used per agent, and the social utility being obtained. For this problem, social utility is defined as the sum of the individual utilities for each target with penalties assigned for ignoring objects.

We implemented two alternative methods for comparison. The first, we called greedy, involved have each agent request all possible sensing resources to track its target potentially overlapping with the requests of other agents. The utility calculation treated these overlaps as subdivided sensor time for each of the tracks. We also implemented algorithms to calculate the optimal utility and optimal number of tracks. Because these algorithms took so long to find the optimal solution however, we were forced to restrict the size of the problems to under 10 targets. Overall, SPAM performed nearly optimally under various amounts of resource contention. Independent analysis of the protocol was also conducted in [14] which verified our findings.

9. Testing and Debugging

In this section we describe some of the practical issues related to testing and debugging the approaches. This area is perhaps the most unexpectedly difficult area, despite the sophisticated basic approaches and the relatively straightforward algorithms used, debugging always degenerated into a process of poring over logfiles, which is clearly inappropriate if such systems are to be widely used.

9.1. Machinetta and Teamwork

Testing and debugging Machinetta teams is extremely difficult. Probabilistic reasoning and complex, dynamic domains lead to occasional errors that are very hard to reproduce or track down. We have extensive logging facilities that record all the decisions the proxies make, but without tool support determining why something failed can be extremely difficult and time-consuming. Simple simulators play a role in allowing extensive debugging of protocols in a simplified environment, but the benefit is limited. We believe, development tools in general, and testing and debugging support specifically, may be the biggest impediment to the deployment of even larger teams.

9.2. Centibots

Debugging is especially difficult because overall the system can appear to be behaving correctly, although it is not. In one experiment, we had 66 robots in use at one time producing over 1 megabyte of logs and debug information per minute. We ran our experiment for more than 2 hours. In Centibots we have a very sophisticated logging mechanism that write every events, messages and informations in a SQL database. Using the database is possible to replay entirely the experimentation. We also built some SQL script that will extract statistics like average running time per robot, average traveling time per robot, number of goals fulfilled per robot that were helpful to debug. Unless the system is really performing strangely noticing the presence bugs is extremely hard.

9.3. Cooperative Mediation

Even with specialized simulation environments, testing and debugging coordination protocols that operate in the large is very difficult. Even on reasonably small problems involving 10's of agents, non-critical problems often go unnoticed for long periods of time. We encountered a number of problems trying to debug and test SPAM.

In the end, countless hours were spent pouring over many, large log files, adding additional debugging text, rerunning, etc. We did develop several graphical displays that helped to identify pathologies (or emergent behaviors) that could only be witnessed by viewing the system's performance from a bird's eye perspective. It's fairly clear that a combination of macro and micro debugging methods are essential to developing systems of this type.

10. Conclusion

Applications that involved hundreds or thousands of agents are starting to appear as working prototype systems. This paper presents three such systems, each using a novel approach to deal with problems that oc-

1 Caused by a misconfigured tracking filter, fixed before the next run.

Run	Mapping Time	Map Area	Search Robots	Search Time False Pos	Position Error / Topo Error
1	22 min	96%	66	34 min / 0	11 cm / none
2	26 min	97%	55	76 min / 1	24 cm / none
3	17 min (2 robots)	95%	43	16 min / 0	20 cm / none
4	19 min (2 robots)	96%	42	Missed / 2	NA
Avg.	21 min	96%	51	30 min / 0.75	14 cm / none

Table 1. Results of the 4 evaluation runs.

cur as coordination is scaled up. While the approaches has some similarities, particularly the use of dynamic, partial centralization, they also have significant differences. The common problems encountered by the three approaches, provide something of a look at the key open problems for scaling coordination. One surprising, key issue was the critical need for new testing and debugging techniques.

References

[1] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

[2] Joseph Culberson and Ian Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1–2):227–264, 2001.

[3] 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.

[4] Mark S. Fox, Norman Sadeh, and Can Baycan. Constrained heuristic search. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 1, pages 309–316, Detroit, MI, August 1989. Morgan Kaufmann.

[5] Kurt Konolige, Dieter Fox, Charlie Ortiz, Andrew Agno, Michael Eriksen, Benson Limketkai, Jonathan Ko, Benoit Morisset, Dirk Schulz, Benjamin Stewart, and Regis Vincent. Centibots: Very large scale distributed robotic teams. In *Proc. of the International Symposium on Experimental Robotics*, 2004.

[6] Roger Mailler, Victor Lesser, and Bryan Horling. Cooperative Negotiation for Soft Real-Time Distributed Resource Allocation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, pages 576–583, Melbourne, July 2003. ACM Press.

[7] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[8] Pragnesh Jay Modi, Hyuckchul Jung, Milind Tambe, Wei-Min Shen, and Shriniwas Kulkarni. Dynamic distributed resource allocation: A distributed constraint satisfaction approach. In John-Jules Meyer and Milind Tambe, editors, *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 181–193, 2001.

[9] Remi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.

[10] P. Scerri, E. Liao, Yang. Xu, M. Lewis, G. Lai, and K. Sycara. *Theory and Algorithms for Cooperative Systems*, chapter Coordinating very large groups of wide area search munitions. World Scientific Publishing, 2004.

[11] 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 *The Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[12] P. Scerri, Yang. Xu, E. Liao, J. Lai, and K. Sycara. Scaling teamwork to very large teams. In *Proceedings of AAMAS'04*, 2004.

[13] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[14] Guandong Wang, Weixiong Zhang, Roger Mailler, and Victor Lesser. *Analysis of Negotiation Protocols by Distributed Search*, pages 339–361. Kluwer Academic Publishers, 2003.

[15] Makoto Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP-95)*, Lecture Notes in Computer Science 976, pages 88–102. Springer-Verlag, 1995.

[16] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.