1

Comparing Three Approaches to Large-Scale Coordination

Paul Scerri[1], Régis Vincent[2] and Roger Mailler[3]

1. Carnegie Mellon University
2. SRI International
3. Cornell University

Summary. Coordination of large groups of agents or robots is starting to reach a level of maturity where prototype systems can be built and tested in realistic environments. These more realistic systems require that both algorithmic and practical issues are addressed in an integrated solution. In this chapter, we look at three implementations of large-scale coordination examining common issues, approaches, and open problems. The key result of the comparison is that there is a surprising degree of commonality between the independently developed approaches, in particular the use of partial, dynamic centralization. Conversely, open issues and problems encountered varied greatly with the notable exception that debugging was a major issue for each approach.

1.1 Introduction

Coordinating large groups of intelligent robots to perform a complex task in a complex environment requires meeting a range of challenges in an integrated solution. These challenges range from well-known algorithmic issues, e.g., managing the computational complexity of task and resource allocation, to more practical issues, e.g., initialization and deployment of a large number of robots. In the past few years, a small number of systems have been developed that are capable of demonstrating real coordination between large numbers of robots in realistic domains. While extensively leveraging the large body of previous work, these systems required new techniques to deal with the practical complexity of coordinating a large group of robots. In this chapter, 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 and approaches described in this chapter involves at least 100 completely unselfish and cooperative group members. One application required coordination of simulated agents for a complex task, one

involved 100 robots on an exploration and, tracking task and another involved hundreds of sensors for a montoring task. The group members are relatively homogeneous, although there is always some heterogeneity due to location. Thus despite being relatively homogeneous in design, the agents were not always 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. Since the applications involve at least somewhat realistic environments, the approaches were required to address a full spectrum of issues, including many practical challenges often ignored in the multiagent literature. Some of these practical challenges are well known, e.g., dealing with lossy communication or building reliable software, while others were more novel, e.g., working out how 100 robots can enter a building in a reasonable amount of time.

While the approach to each application was developed independently of the others and was underpinned by a diverse set of philosophies and constraints, there was a surprising amount of commonality in both the solutions and the open problems. Two specific, major commonalities were of particular interest. The first was that each approach used some form of *dynamic, partial centralization* to reduce the overall complexity. In particular, 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", were not chosen in advance. The reason for this commonality appears to stem from a need to balance the complexity of key algorithms and the practical limitations of time and communication resources. In situations where coordinated decision making involved a large percentage of the group, developers resorted to various heuristics for controlling resource requirements, and when a small percentage of the group was involved, partial centralization was used. Although the reason for it is unclear, it is noteworthy that no optimal completely distributed algorithms were used, perhaps because in cases where they were applicable partial centralization was more efficient.

Most 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 the lack of top-down structure was at least partially due to the decomposibility of the task, there was more structure to the coordination than the task, indicating that the coordination was not simply designed to mirror the task. Interestingly, none of the approaches were inspired by any particular organizational theory, human or biological. Structure limited the decisions that could be made by an individual, including who that individual could communicate with about what, what tasks the in-

dividual could perform, and protocols for making coordinated decisions. For example, in one of the approaches, the notion of a subteam was strictly defined and carried certain responsibilities that were often not required for best coordinated behavior, but simplified the possible organizations that could occur. Although not explicit in any of the designs, it appears that each approach carefully balanced imposed structure for making the coordination intelligible to a human and flexibility for allowing the group to find the best way to complete a task. The need for intelligibility was key when programming, testing, deploying and improving the system, but the additional structure limited the potential of the team. Future development tools may open the possibility to decrease the amount of structure and, thus, increase the potential 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. This diversity of problems and open issues is especially interesting since the approaches had so much in common. However, it is unclear what to conclude from this, since one might come to the mutually exclusive conclusions that the basic approach was poor and problems manifested themselves in different ways or that the approach was fundamentally good and time was spent on more detailed issues. More applications are required for a definitive conclusion. In each approach, debugging was found to be a major difficulty with only the most rudimentary support available for debugging extremely complex, distributed applications. The most stunning evidence of this problem is that all approaches reported that major bugs went unnoticed for extended periods of time, before being discovered by chance. The bugs went unnoticed because the overall behavior was not accurately predicted in advance, so disappointing performance was attributed to causes other than faulty software.

In the remainder of this chapter, we briefly describe the way each of the three approaches addresses a variety of problems. By showing in detail the commonalities and differences, we provide a fair comparison of the approaches. Finally, open, important problems, identified in the development of the systems, are described to help shape the research agenda for large-scale coordination.

1.2 Applications and Assumptions

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.

1.2.1 Teamwork and Machinetta

Machinetta software proxies are used to develop teams where the members are assumed to be completely cooperative and willing to incur costs for the overall good of the team [20]. Typically, team members will be highly heterogeneous, ranging from simple agents and robots to humans. When a group of agents coordinates via *teamwork* they can flexibly and robustly achieve joint goals in a distributed, dynamic and potentially hostile environment [7, 9]. Key teamwork algorithms have evolved from an extensive body of work on both the theory and practice of teamwork [23, 8, 3]. Teams of heterogeneous actors have potential applications in a wide variety of fields, ranging from supporting human collaboration [1, 22] to disaster response [16] to manufacturing [9] to training [23] to games [10]. To date we have demonstrated teams of 500 software agents [21], in both a UAV simulation [19] and a disaster response simulation, but teams of as many as 200,000 agents are envisioned.

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 that it is better to have fire trucks on reasonable routes to fires, than to delay departure with computationally expensive optimization. While the team will be able to leverage reasonably high bandwidth communication channels, we assume that 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 subtasks can take place.

1.2.2 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 (CENTIBOTS, see figure 1.1), on an indoor search-and-rescue task. In this project, communication was limited and unreliable, and any coordination mechanisms had to deal with the limitations. There are two types of agents in the Centibots system; hence, heterogeneity is not an issue. Similarly, optimality is infeasible, so having a reactive, "good enough" system was the primary aim.

In the scenario, the CENTIBOTS are deployed as a search-and-rescue team for indoor missions. A first set of mapping-capable CENTIBOTS surveys the

Fig. 1.1. 100 Robots used during the January 2004 evaluation.

area of interest to build and share a distributed map highlighting hazards, humans, and hiding places. A second wave of robots, with the capability of detecting an object of interest (e.g. biochemical agents, computers, victims), is then sent. The key goal of 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 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 [11].

Communication is done using an ad-hoc wireless network, which has a maximum *shared* bandwidth of 1 Mpbs. Communication 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. Essentially, there is no chance that all the robots will finish the mission. In addition, resources (i.e. robots) and goals can be added, removed, or disabled at any time, making an adaptable system crucial.

1.2.3 Cooperative Mediation

Scalable, Periodic, Anytime Mediation (SPAM) [12] 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 hundreds to thousands of agents that controlled sensors within a large sensor network as part of the DARPA Autonomous Negotiating Teams (ANTS) program (see figure 1.2).

In this project, sensors were randomly placed in the environment and had to coordinate their internal schedules in order to discover and track moving targets. Each of the sensor platforms had three Doppler-radar-based sensor heads capable of returning amplitude and frequency shift information for objects within their 20-foot range and 120-degree viewable arc. Because of this, multiple, temporally coordinated measurements from different sensors within the network were needed in order to triangulate the precise position of a target at any given time. In addition, each of the sensor platforms was controlled by a Basic stamp micro-controller that was capable of processing the incoming sensor data from only one head at a time. These two factors when combined together formed the basis of a difficult, distributed resource allocation problem that was further complicated by dynamics created by the movement of the targets.

Adding to the complexity of this problem, communications varied from 100 Mbps TCP-based wired networks to 14.4 kbps half duplex, RF-based, multichannel wireless communications. In the latter, message passing was very

Fig. 1.2. Researchers work on a demonstration involving 36 sensors and 3 mobile targets.

unreliable and loss rates of 50% were not uncommon. The communication restrictions combined with the real-time coordination needs made complete centralization out of the question and traditional distributed techniques inadequate.

SPAM has been tested in real-world hardware environments with 36 sensors and in simulated environments with more than 500 sensors.

1.3 Key Algorithms and Principles

Although distinct approaches are used, i.e., teamwork, hierarchical 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.

1.3.1 Machinetta and Teamwork

A key principle in teamwork is that agents have both models of teamwork and models of other team members [21]. The 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 key abstraction in our implementation of teamwork is a Team Oriented Plan, which breaks a complex joint activity down into individual *roles*, with constraints between the roles [18]. Typically, a large team will be executing many team-oriented plans at any time. Dynamically changing subteams form to execute each of the plans. Small amounts of communication occur across subteams, to ensure that sub-teams do not act at cross purposes or duplicate efforts.

Scalable algorithms required to perform the teamwork were designed with two key ideas in mind. First, we use probabilistic models of team activity and state to inform key algorithms. This actually leverages the size of the team because the probabilistic models tend to be more accurate 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 because 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. Specifically, most key algorithms use *tokens* to encapsulate "chunks" of coordination reasoning [19]. A good example of these principles is in our algorithm for ensuring that the team is not working on conflicting plans. That algorithm uses tokens, for robustness, and the associates network to ensure, *with high probability*, that the team is not working at cross purposes.

These two principles are embodied in the role allocation process that uses a probabilistic model of the current capabilities and tasks of the team to calculate a threshold capability level that a team member performing a role would have in a good overall allocation, and then uses a *token* that moves around the team until an available team member is found with capability above the threshold [5].

1.3.2 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 that 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 hierarchical 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 selected. The manager selection is unimportant as known solutions can be used. The commander assigns a set of goals to each team and the teams' dispatchers assign these to individual robots. When a robot has finished its assigned goals, it notifies the dispatcher, making 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 position, the

[1] Anything that can go wrong will go wrong.

problem is to minimize the search time. This allocation is in theory similar to a multiple traveling salesman problem except that there is no a priori notion of how many salesmen you might have, and a salesman can fail at any time during the traveling. 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.

1.3.3 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 suboptimality 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 that allows the system to converge on mutually beneficial assignments.

This technique represents a new paradigm in distributed problem solving. Unlike current techniques that attempt to limit the information the agents use to make decisions in order to maintain distribution [28, 27], SPAM and more generally cooperative mediation centralize portions of the problem in order to exploit the speed of centralized algorithms.

1.4 Key Novel Ideas

New ideas were required to overcome weaknesses in the principles as approaches were scaled from small numbers of agents to the large numbers needed for the coordination.

1.4.1 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* [21]. As well as the obligation to communicate information to members of its dynamically changing subteam, as required by teamwork, an agent must keep its neighbors in the associates network appraised of 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 [26]. Because of a phenomenon 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 [5]. LA-DCOP extends distributed constraint optimization techniques in several ways to make it appropriate for large, dynamic teams. Most important, 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 [20]. For example, in a disaster response domain, there may be a role for fighting some particular fire that no firefighter 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.

1.4.2 Centibots

The hierarchical dispatching model offers two key interesting qualities. 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 resides physically anywhere on the network. The dispatcher can be running on any team member, and would require only local communication. The second quality is a natural hierarchy can be created to handle a large number of robots. In this configuration, we could have a hierarchy of dispatchers, each responsible for an area of the map, using a subteam 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, achieving a load-balancing system. Like the SPAM system, the Centibots architecture leverage the power of the mediation by centralizing a sub portion of the problem.

1.4.3 Cooperative Mediation

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 including agents with which

they directly share resources. 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. The downside to this heuristic approach, however, is that the mediators have less information and are often unaware of the consequences of their actions on other agents. To combat this effect, SPAM incorporates the use of *conflict propagation* and *conflict dampening*.

As the name implies, conflict propagation occurs whenever a mediator causes conflicts for agents that are outside of one of its sessions. It easy to envision this as the mediator pushing the conflicts onto agents over which it has no control (or responsibility). The key goal of the propagation is to find regions within the global resource problem that are under-constrained and can absorb the conflict. The actual propagation occurs when one the agents that has the newly introduced conflict takes over the role of mediator. These agents can then either absorb the conflict (by finding a satisfying assignment to their subproblem) or can push the conflict onto other agents, which may push it even further.

It is easy to see that conflict propagation alone would have disastrous consequences if it were not for conflict dampening. Conflict dampening is very similar to the min-conflict heuristic presented in [13]. 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 impact 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 because of the effects of friction and gravity.

SPAM also incorporates a number of resource-aware mechanisms that prevent it over-utilizing communication. 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 overburdened, it is dropped from the session. In addition, if an agent notices that it has become a communication hotspot, then it avoids taking the role of mediator until the situation resolves itself. Overall, these mechanisms allow SPAM to tradeoff utility for scalability of communications.

1.5 Software

We describe the major pieces of technology, specifically software, that are used for the coordination in each of the approaches.

1.5.1 Machinetta and Teamwork

The teamwork algorithms are encapsulated in domain-independent *software proxies* [17]. 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 [20]. 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 [21]. 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. The proxy software is freely available on the Internet.

1.5.2 Centibots

The Centibots software makes an extensive use of the Jini [24] architecture. Each robot and each key algorithm is a network service that registers, advertises and interacts independently of its physical location. We have services like the map publisher that aggregates data from the mappers and publishes a map for the other robots, and like the dispatcher that allocates tasks to robots or even the user interface. The result is a very modular, scalable infrastructure. Each robot has its own computer where it runs localization, navigation, path planning, and vision processing algorithms.

1.5.3 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 nondecomposable decision point within the protocol. Transitions between states are event driven and allow the protocol to specify state transitions based on time-outs, message traffic, specific execution conditions, and so on. This allows the 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.

1.6 Key Unexpected Challenges

Challenges were encountered during development that were not expected at the outset. Each approach ran into different, unexpected problems, ranging from sharing information to controlling oscillations.

1.6.1 Machinetta and Teamwork

Two main unexpected challenges occurred during the development of large teams. First, it was often the case that some team member had information that could be relevant to some other member of the team, but did not know to which other team member the information was relevant. For example, in a disaster response domain, an agent may get information about chemicals stored in a particular factory, but not know which firefighters will be attending that fire. Restricting knowledge of current activities to within a subteam provides scalability but 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. To address this problem we made use of the fact that the associates network connecting team members had a small worlds property and allowed an agent to *push* information to its neighbor most likely to be able to make use of that information or know who would [26].

The second unexpected problem encountered was that there were many algorithm parameters that interact with one another in highly nonlinear ways. Moreover, slightly different situations on the ground require substantially different configuration of the algorithm parameters. 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. We are currently developing techniques that use neural networks to model the relationships between parameters and assist the user in finding optimal settings for specific performance requirements and tradeoffs.

1.6.2 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 parts of buildings where this conservative distance was not working. In this case, any robot that enters this communication dead zone will not be able to contact the centralized dispatcher. Our solution was to have the dispatcher living on close-by robots , which was a good improvement but did not 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 could not contact the dispatcher. In this case, at least we would retrieve it.

The second challenge was to determine the number of goals to assign 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. In section 1.5.2 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 varies (one to seven) depending on the number of goals left to be 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 overutilized) we vary the total number of goals to be assigned.

1.6.3 Cooperative Mediation

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

During the development of the SPAM protocol, we explored a method in which each mediator maintained a history of the sensor schedules that were being mediated whenever a session terminated. By doing this, mediators were able determine if they previously may have been in a state that 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 by this technique, which resulted in having the mediator unnecessarily accept a lower-quality solution.

This technique is similar to that applied in [14], 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.

In the final implementation of the SPAM protocol, we allowed the agents to enter into potential oscillation, maintaining almost no prior state from session to session and relied on the environment to break oscillations through the movement of the targets, asynchrony of the communications, timeouts, and so on.

1.7 Open Problems

As with the unexpected problems, each approach has different open problems. 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.

1.7.1 Machinetta and Teamwork

Despite its successes, Machinetta has some critical limitations. Most critically, Machinetta relies on a library of predefined 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 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 ad hoc basis, the general problem of having the team detect that the coordination is failing is important before deploying teams. If such a problem is 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.

1.7.2 Traffic Control in Centibots

Linked to the goal assignment, traffic control for several dozen robots in a small environment is a huge challenge. The assignment should take into consideration the schedule in which each robot will do its 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 dozens of real-life experiments some conflicts becoming literally traffic jams and blocking permanently one access of an area. The only way to

reason about the choke point is as resource and solve the conflict during the assignment by using a method such as SPAM.

1.7.3 Cooperative Mediation

The most interesting open questions for the SPAM protocol deal 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 are strongly tied not only to the size, but to the characteristics of the subproblem that the mediators centralize, a detailed study needs to be conducted to determine the relationship between these two competing factors. Some work has already been done that preliminarily addresses 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 [6]. In addition, recent work on phase transitions in CSPs [2, 4, 15] in part addresses the question of when. It is clear that a great deal of work still needs to be done.

1.8 Evaluation and Metrics

We agree that evaluating the algorithms and the metrics used to measure performance is an immature and difficult science. Clearly, useful and comparable metrics will need to be developed, if sensible comparison is to be performed.

1.8.1 Machinetta and Teamwork

Evaluating teamwork is very difficult. 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 when evaluating the algorithms, we have endeavored to use at least two distinct domains for testing. Moreover, 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, because of the sheer size of the environment and the number of agents, there tends to be high variation in performance, implying that many runs must 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 of 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 no 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, subteam 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.

1.8.2 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. They were tested under controlled conditions, with a single operator in charge of the robot team.

For searching, the evaluation criteria were time to locate object of interests (OOIs), positional accuracy, and false detections. There were four evaluation runs, and the results, in the Table 1.1, 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.

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 1.1. Results of the four evaluation runs.

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 a system is not an easy task.

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

1.8.3 Cooperative Mediation

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

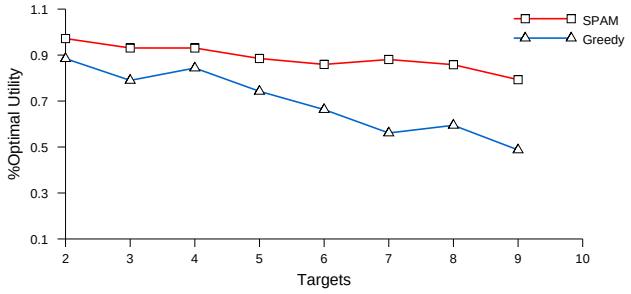

Fig. 1.3. Percentage of optimal utility for SPAM and greedy solutions.

We implemented two alternative methods for comparison. The first, which are called *greedy*, involved having 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 less than 10 targets. Overall, SPAM performed nearly optimally under various amounts of resource contention (see figure 1.3). Independent analysis of the protocol was also conducted in [25], which verified these findings.

1.9 Testing and Debugging

Testing and debugging of the approaches 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 pouring over logfiles, which is clearly inappropriate if such systems are to be widely used.

1.9.1 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 that development tools in general, and testing and debugging support specifically, may be the biggest impediment to the deployment of even larger teams.

1.9.2 Centibots

Debugging is especially difficult because overall the system is behaving correctly. In one experiment, we had 66 robots in use at one time, producing over 1 MB of logs and debugging information per minute. We ran our experiment for more than 2 hours. In Centibots, we have a very sophisticated logging mechanism that writes every event, every message and information in an SQL database. By using the database, it is possible to replay an entire experiment. We also built SQL scripts that can extract statistics such as average running time per robot, average traveling time per robot, and number of goals fulfilled per robot that are very useful to the debugging process. Unless the system is performing very strangely, noticing the presence of bugs is extremely hard. In fact, one bug persisted for more than a year before being detected and fixed, leading to a dramatic improvement in performance.

1.9.3 Cooperative Mediation

Even with specialized simulation environments, testing and debugging coordination protocols that operate in the large is very difficult. On reasonably small problems involving tens of agents, noncritical problems often went unnoticed for long periods of time. We encountered a number of problems in trying to debug and test SPAM.

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

1.10 Conclusion

We have presented three initial attempts at performing large-scale coordination among robots or agents. We have shown striking similarities between the

approaches that raise interesting scientific questions that must be addressed in a principled way. Critically, design of the coordination seems to be driven more by the difficult challenge of developing the software to implement it than by principles or theory. It will be important, for the field to move forward, to balance (or mitigate) development complexity with algorithmic performance in a better way than has been done so far. If these challenges can be met, the promise of large-scale coordinating is very exciting.

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. 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.
3. Philip R. Cohen and Hector J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.
4. Joseph Culberson and Ian Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1–2):227–264, 2001.
5. 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.
6. 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.
7. J. Giampapa and K. Sycara. Conversational case-based planning for agent team coordination. In *Proceedings of the Fourth International Conference on Case-Based Reasoning*, 2001.
8. Barbara Grosz and Sarit Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996".
9. Nick Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75:195–240, 1995.
10. 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.
11. 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.
12. 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.

13. 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.
14. 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, *Preproceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 181–193, 2001.
15. Remi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400:133–137, 1999.
16. 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.
17. David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems, Special Issue on Infrastructure and Requirements for Building Research Grade Multi-Agent Systems*, page to appear, 2002.
18. 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.
19. 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.
20. 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.
21. P. Scerri, Yang. Xu, E. Liao, J. Lai, and K. Sycara. Scaling teamwork to very large teams. In *Proceedings of AAMAS'04*, 2004.
22. K. Sycara and M. Lewis. *Team Cognition*, chapter Integrating Agents into Human Teams. Erlbaum Publishers, 2003.
23. Milind Tambe. Agent architectures for flexible, practical teamwork. *National Conference on AI (AAAI97)*, pages 22–28, 1997.
24. Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.
25. Guandong Wang, Weixiong Zhang, Roger Mailler, and Victor Lesser. *Analysis of Negotiation Protocols by Distributed Search*, pages 339–361. Kluwer Academic Publishers, 2003.
26. Y. Xu, M. Lewis, K. Sycara, and P. Scerri. Information sharing in very large teams. In *In AAMAS'04 Workshop on Challenges in Coordination of Large Scale MultiAgent Systems*, 2004.
27. 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.
28. 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.