Chapter 1

COORDINATING VERY LARGE GROUPS OF WIDE AREA SEARCH MUNITIONS *

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

Yang Xu, Mike Lewis
University of Pittsburgh
xuy3@pitt.edu, ml@sis.pitt.edu

Abstract

Coordinating hundreds or thousands of unmanned aerial vehicles (UAVs), presents a variety of new exciting challenges, over and above the challenges of building single UAVs and small teams of UAVs. We are specifically interested in coordinating large groups of Wide Area Search Munitions (WASMs), which are part UAV and part munition. We are developing a "flat", distributed organization to provide the robustness and flexibility required by a group where team members will frequently leave. Building on established *teamwork* theory and infrastructure we are able to build large teams that can achieve complex goals using completely distributed intelligence. However, as the size of the team is increased, new issues arise that require novel algorithms. Specifically, key algorithms that work well for relatively small teams, fail to scale up to very large teams. We have developed novel algorithms meeting the requirements of large teams for the tasks of instantiating plans, sharing information and allocating roles. We have implemented these algorithms in reusable software proxies using the novel design abstraction of a *coordination agent* that encapsulates a piece of coordination protocol. We illustrate the effectiveness of the approach with 200 WASMs coordinating to find and destroy ground based targets in support of a manned aircraft.

*This research has been supported by AFRL/MNK grant F08630-03-1-0005.

1. INTRODUCTION

Wide Area Search Munitions (WASMs) are a cross between an unmanned aerial vehicle and a munition. With an impressive array of onboard sensors and autonomous flight capabilities WASMs can play a variety of roles in a modern battle field including reconnasiance, search, battle damage assessment, communications relays and decoys. Also being able to play the role of munition makes WASMs a very valuable asset for battlefield commanders. In the foreseeable future, it is envisioned that groups of the order of 100 WASMs will support and protect troops in a battlespace.

Getting large groups of WASMs to cooperate in dynamic and hostile environments is an exciting though difficult challenge. There have been significant successes in automated coordination[5, 8, 18, 31], but the number of entities involved has been severely limited due to the failure of key algorithms to scale to the challenges of large groups. When coordinating small groups of WASMs there are a variety of challenges such as formation flying and avoiding mid-air collisions. However, when we scale up the number of WASMs in the group, a new set of challenges, attributable to the scale of the team, come to the fore. For example, communication bandwidth becomes a valuable commodity that must be carefully managed. This is not to say that the challenges of small teams disappear, only that there are additional challenges. The focus of this chapter, is on those challenges that occur only when the size of the group is scaled up.

Given the nature of the domain, we are pursuing a completely distributed organization that does not rely on any specific entity, either WASM or human, for continued operation. This makes the overall system more robust to enemy activity. Our flat organization builds on well understood theories of *teamwork*[7, 13, 19, 16, 35]. Teamwork has the desirable properties of flexibility and robustness we require. Coordination based on ideas of teamwork requires that a number of algorithms work effectively together. We encapsulate our teamwork algorithms in a domain independant, reusable software proxy[27, 18]. A proxy works in close cooperation with a domain level agent to control a single team member. Specifically, the proxy works in close cooperation with an autopilot to control a single WASM. The proxies communicate among themselves and with their domain agent to achieve coordination.

The proxies execute Team Oriented Plans (TOPs) that break team activities down into individual activities called *roles*. TOPs are specified

a priori, typically by a human designer, and specify the means by which the team will achieve its joint goals. Typically, TOPs are parameterized in templates and can be instantiated at runtime with specific details of the environment. For example, a TOP for destroying a target might have the specific target as a parameter. Importantly, the TOP does not specify who performs which role, nor does the TOP specify low level coordination details. Instead, these generic coordination "details" are handled by the proxies at runtime, allowing the team to leverage available resources and overcome failures.

The proxies must implement a range of algorithms to facilitate the execution of a TOP, including algorithms for instantiating TOPs, allocating roles and sharing relevant information. To build large teams, novel approaches to key algorithms are required. Specifically, we have developed novel approaches to creating and managing team plans, to allocating roles and to sharing information between team members. Our approach to plan instantiation allows any proxy to instantiate a TOP. The team member can then initiate coordination for, and execution of, that TOP and then the whole team (or just a part) can be involved in the coordination and execution.

We are also developing new communication reasoning that works by simply passing pieces of information to group members more likely to know who needs that information. Previous algorithms for reasoning about communication have made assumptions that do not hold in very large groups of WASMs. Specifically, previous algorithms have either assumed that centralization is possible or have assumed that agents have accurate models of other members of the group. Because of a phenomenon called "small world networks"[38] (in human groups this phenomenon is caputured informally by the notion of "six degrees of separation") the result of our simple communication technique is targeted information delivery in an efficient manner. Our algorithm avoids the need for accurate information about group members and functions well even when group members have only very vague information about other group members.

Our implementation of the proxies is based on the abstraction of a *coordination agent*. Each coordination agent is responsible for a "chunk" of the overall coordination and encapsulates a protocol for one aspect of the coordination. We use a separate coordination agent for each plan or sub-plan, role and piece of information that needs to be shared. Specifically, instead of distributed protocols, which provide no single agent a cohesive view of the state of coordination, that state is encapsulated by the coordination agent and moves with that agent. Thus, the proxies can be viewed as a mobile agent platform upon which the coordination

agents execute the TOPs. A desirable side effect of this design abstraction is that it is easier to build and extend complex "protocols" since the complexity of the protocol is hidden in the coordination reasoning, rather than being spread out over many agents.

We are evaluating our approach in a WASM simulation environment that emphasizes the coordination issues, without requiring too much attention to aerodynamic or low-level control issues. We have implemented two different forms of control, centralized and distributed, to allow us to quickly test ideas then perform more detailed validation. Our initial experiments have revealed some interesting phenomena including that very simple target allocation algorithms can perform surprisingly well under some circumstances.

2. WIDE AREA SEARCH MUNITIONS

Wide Area Search Munitions (WASMs) are a cross between an unmanned aerial vehicle and a standard munition. The WASM has fuel for about 30 minutes of flight, after being launched from an aircraft. The WASM cannot land, hence it will either end up hitting a target or self destructing. The sensors on the WASM are focused on the ground and include video with automatic target recognition, ladar and GPS. It is not currently envisioned that WASMs will have an ability to sense other objects in the air. WASMs will have reliable high bandwidth communication with other WASMs and with manned aircraft in the environment. These communication channels will be required to transmit data, including video streams, to human controllers, as well as for the WASM coordination.

The concept of operations for WASMs are still under development, however, a wide range of potential missions are emerging as interesting. A driving example for our work is for a team of WASMs to be launched from an AC-130 aircraft supporting special operations forces on the ground. The AC-130 is a large, lumbering aircraft, vulnerable to attack from the ground. While it has an impressive array of sensors, those sensors are focused directly on the small area of ground where the special operations forces are operating. The WASMs will be launched as the AC-130 enters the battlespace. The WASMs will protect the flight path of the AC-130 into the area of operations of the special forces, destroying ground based threats as required. Once the AC-130 enters a circling pattern around the special forces operation, the WASMs will set up a perimeter defense, destroying targets of opportunity both to protect the AC-130 and to support the soldiers on the ground. Even under ideal conditions there will be only one human operator on board the AC-130

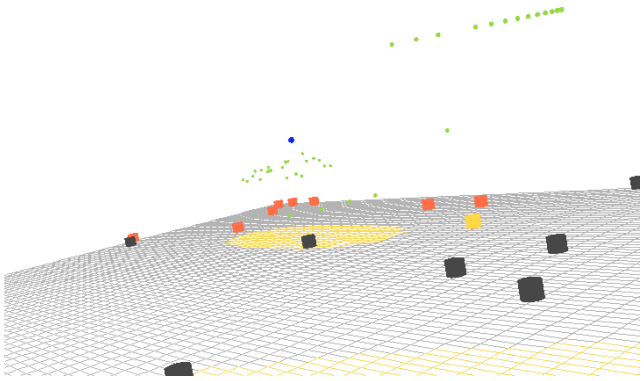


Figure 1.1. A screenshot of the simulation environment. A large group of WASMS (small spheres) are flying in protection of a single aircraft (large sphere). Various SAM sites are scattered around the environment. Terrain type is indicated by the color of the ground.

responsible for monitoring and controlling the group of WASMs. Hence, high levels of autonomous operation and coordination are required of the WASMs themselves.

Many other operations are possible for WASMs. Given their relatively low cost compared to Surface-to-Air Missiles (SAMs), WASMs can be used simply as decoys, finding SAMs and drawing fire. WASMs can be used as communication relays for forward operations, forming an adhoc network to provide robust, high bandwidth communications for ground forces in a battle zone. Since a WASM is "expendible", it can be used for reconnasiance in dangerous areas, providing real-time video for forward operating forces. Many other operations could be imagined in support of both manned air and ground vehicles, if issues related to coordinating large groups can be adequately resolved.

While our domain of interest is teams of WASMs, the issues that need to be addressed have close analogies in a variety of other domains. For example, coordinating resources for disaster response involves many of the same issues[23], as does intelligent manufacturing[29] and business processes. These central issues of distributed coordination in a dynamic environment are beginning to be addressed, but in all these domains current solutions do not efficiently scale to large numbers of group members.

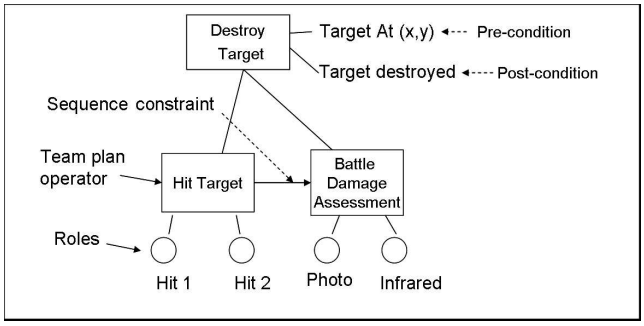


Figure 1.2. An example team plan for destroying a ground based target. There are four roles, that will be instantiated in two stages, destroying the target (which requires that two WASMs hit the target) and the subsequent battle damage assessment (which requires both a photo and infrared imagining).

3. LARGE SCALE TEAMWORK

The job for the proxies is to take the TOP templates, instantiate TOPs as events occur in the environment then manage the execution of the instantiated TOPs. To achieve this a number of algorithms must work effectively together. Events occuring in the environment will only be detected by some agents (depending on sensing abilities). The occurance of these events may need to be shared with other proxies so that a single proxy has all the information required to instantiate a plan. Care must be taken to ensure that there are not duplicate or conflicting team plans instantiated. Events occurring in the environment need to be shared with agents performing roles that are impacted by those events. Once the plans are instantiated, roles need to be allocated to best leverage the team capabilities. Plans also need to be terminated when they are completed, irrelevant or unachievable. Other algorithms, such as ones for allocating resources, may also be required but are not considered here. All the algorithms must work together efficiently and robustly in order for the team to achieve its goals.

Viewed abstractly, the reasoning of the team can be seen as a type of hierarchical reasoning. At the top of the hierarchy are the plans that will be executed by the team. Those plans get broken down into more detailed plans, until the pieces, which we call roles, can be performed by a single team member. The next layers of the hierarchy deal with allocating those roles and finding coalitions for sets of roles that must be performed together. Finally, at the bottom of the hierarchy, is the detailed reasoning that allows team members performing as a part of a coalition to work together effectively. In these small coalitions we can

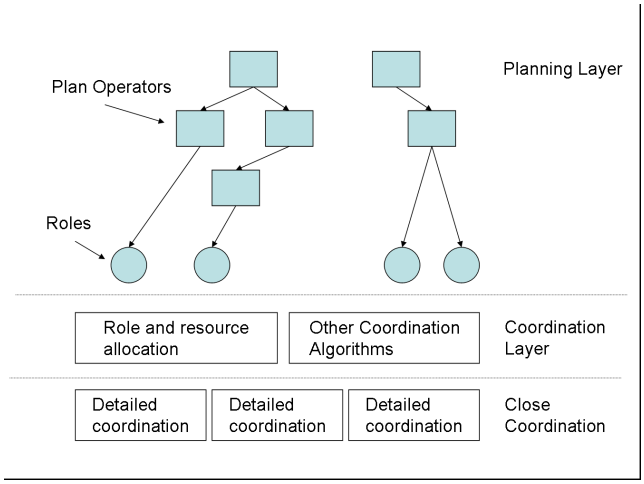


Figure 1.3. Conceptual view of teamwork reasoning hierarchy. At the top, boxes represent team plans which are eventually broken down into individual roles. The roles are sent to the coordination layer which allocates the roles and resources to execute the plans. Finally, at the detailed level, specific sub-teams must closely coordinate to execute detailed plans.

apply standard teamwork coordination techniques such as STEAM. The basic idea is shown in Figure 1.3. The important caveat is that there is no hierarchical reasoning imposed on the team, the hierarchical view is simply a way of understanding what is happening. In the remainder of this section, we describe the proxies, the coordination agents and some of the key algorithms.

3.1. MACHINETTA PROXIES

To enable transitioning our coordination techniques to higher fidelity simulation environments or other domains, we separate the low level dynamic control of the WASM from the high level coordination code. The general coordination code is encapsulated in a *proxy*[18, 36, 26, 32]. There is one proxy for each WASM. The basic architecture is shown in Figure 1.4. The proxy communicates via a high level, domain specific protocol with an intelligent agent that encapsulates the detailed control algorithms of the WASM. Most of the proxy code is domain independent and can be readily used in other domains requiring distributed control. The proxy code, known as Machinetta, is a substantially extended and updated version of the TEAMCORE proxy code[36]. TEAMCORE

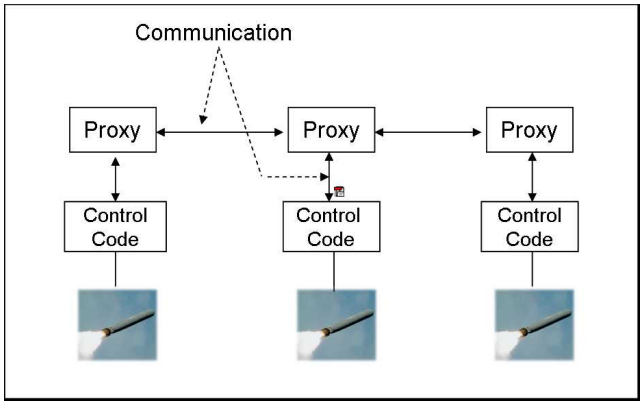


Figure 1.4. The basic system architecture showing proxies, control code and WASMs being controlled.

proxies implement *teamwork* as described by the STEAM algorithms [35], which are in in turn based on the theory of *joint intentions*[19, 7].

Coordination Agents. In a dynamic, distributed system protocols for performing coordination need to be extremely robust. When we scale the size of a team to hundreds of agents, this becomes more of an issue than simply writing bug-free code. Instead we need abstractions and designs that promote robustness. Towards this end, we are encapsulating "chunks" of coordination in *coordination agents*. Each coordination agent manages one specific piece of the overall coordination. When control over that piece of coordination moves from one proxy to another proxy, the coordination agent moves from proxy to proxy, taking with it any relevant state information. We have coordination agents for each plan or subplan (PlanAgents), each role (RoleAgents) and each piece of information that needs to be shared (InformationAgents). For example, a RoleAgent looks after everything to do with a specific role. This encapsulation makes it far easier to build robust coordination.

Coordination agents manage the coordination in the network of proxies. Thus, the proxy can be viewed simply as a mobile agent platform that facilitates the functioning of the coordination agents. However, the proxies play the additional important role of providing and storing local information. We divide the information stored by the proxies into two categories, domain specific knowledge, K, and the coordination knowledge of the proxy, CK. K is the information this proxy knows about the state of the environment. For example, the proxy for a WASM knows its own location and fuel level as well as the the location of some tar-

gets. This information comes both from local sensors, reported via the domain agent, and from coordination agents (specifically InformationAgents, see below) that arrive at the proxy. CK is what the proxy knows about the state of the team and the coordination the team is involved in. For example, CK includes the known team plans, some knowledge about which team member is performing which role and the TOP templates. At the most abstract level, the activities of the coordination agents involve moving around the proxy network adding and changing information in C and CK for each agent. The content of K as it pertains to the local proxy, e.g., roles for the local proxy, govern the behavior of that team member. The details of how a role is executed by the control agent, i.e., the WASM, are domain (and even team member) dependant.

A *Factory* at each proxy is responsible for creating coordination agents as required.[4] It creates a *PlanAgent* when the pre-conditions of a plan template are met and an *InformationAgent* when a new piece of domain information is sensed locally by the proxy allowing the team to share information sensed locally by a proxy. The algorithm is shown in Figure 1.5.

Factory
loop
Wait for state change
foreach template ∈ TOP Templates
if matches (*template*, K)
Create PlanAgent(*template*, K)
end for
if new locally sensed information in K
Create InformationAgent (new information)
end loop

Figure 1.5. Algorithm for a proxy's factory.

3.2. TEAM ORIENTED PLANS

The basis of coordination in the Machinetta proxies are a *Team Oriented Plans* (TOP) [28]. A TOP describes the joint activities that must take place for the team to achieve its goals. At any point in time, the team may be executing a number of TOPs simultaneously. TOPs are instantiated from TOP *templates*. These templates are designed before the team begins operation, typically by humans to ensure compliance

[4] *Factory* is a software engineering term for, typically, an object that creates other objects

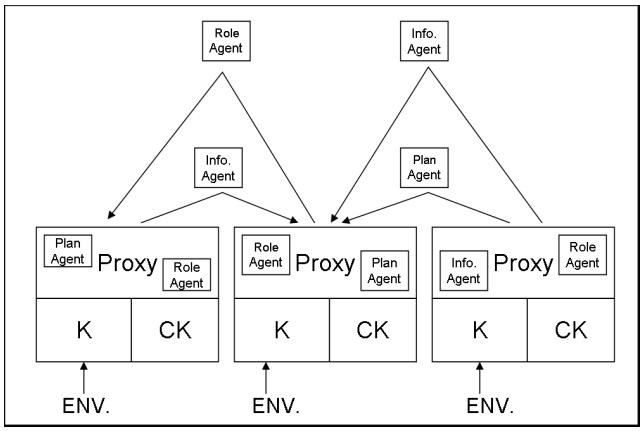


Figure 1.6. High level view of the implementation, with coordination agents moving around a network of proxies.

with established doctrine or best practices. A TOP is a tree structure, where leaf nodes are called *roles* and are intended to be performed by a single team member. For example, a typical TOP for the WASM domain is to destroy a ground based target as shown in Figure 1.2. Such a plan is instantiated when a ground based target is detected. The plan is terminated when the target is confirmed as destroyed or the target becomes irrelevant. The plan specifies that the roles are to actually hit the target and to perform battle damage assessment. The battle damage assessment must be performed after the target has been hit. The coordination algorithms built into the proxies handle the execution of the TOP, hence the plan does not describe the required coordination nor how the coordination needs to be performed. Instead the TOP describes the high level activities and the relationship and constraints between those activities.

Plan Monitoring with PlanAgents. A *PlanAgent* is responsible for "managing" a plan. This involves instantiating and terminating roles as required and stopping execution of the plan when the plan either succeeds, becomes irrelevant or is no longer achievable. These conditions are observed from K in the proxy state. Currently, the PlanAgent must simply match conditions using string matching against post-conditions in the template, but we can envision more sophisticated reasoning in the future.

Because plans are instantiated in a distributed manner, the PlanAgents need to ensure that there are not other plans that are attempting to achieve the same goal (e.g., hit the same target) or other plans that

may conflict. We discuss the mechanisms by which a PlanAgent can avoid these conflicts below. To facilitate the conflict avoidance (and detection) process, as well as keeping the team appraised of ongoing activities, the first thing a PlanAgent does is create an InformationAgent to inform the other proxies (who will update CK.)

If the PlanAgent does not detected any conflicts, it executes its main control loop until the plan becomes either irrelevant, unachievable or is completed. For each role in the plan, a *RoleAgent* is created. RoleAgents are coordination agents that are responsible for a specific role. We do not describe the RoleAgent algorithms in detail here, see [12] for details. Suffice it to say that the RoleAgent is responsible for finding a team member to execute that role. As the plan progresses, the required roles may change, in which case the PlanAgent must terminate the current RoleAgents and create new RoleAgents for the new roles. It is also possible that a previously undetected plan conflict is found and one plan needs to be terminated. The PlanAgents responsible for the conflicting plans jointly determine which plan to terminate (not shown for clarity). When the plan is completed, the PlanAgent terminates any remaining RoleAgents and finishes. The overall algorithm is shown in Figure 1.7.

PlanAgent
Wait to detect conflicts between plans
if conflict detected *then* end
else
Create InformationAgent to inform others of plan
Instantiate initial RoleAgents
while ($\neg$*irrelevant* & $\neg$*complete* & $\neg$*unachievable*)
Wait for change in K or CK
Check if RoleAgents need to be terminated
Instantiate new RoleAgents if required
if newly detected plan conflicts *then*
Terminate this plan or conflicting
end if
end while
end if
Terminate all RoleAgents

Figure 1.7. Algorithm for a PlanAgent

Instantiating Team Oriented Plan Templates. The TOP templates typically have open parameters which are instantiated with specific domain level information at run time. Specifically, the *Factory* uses

K to match against open parameters in plan templates. The matching process is straightforward and currently involves simple string matching.[6] The *Factory* must also check CK to ensure that the same TOP has not been previously instantiated. When the team is very large, it is infeasible to have all team members agree on which plan to instantiate or even for all team members to know that a particular plan has been instantiated. For example, in a team with 100 members, it may take on the order of minutes to contact all members, significantly delaying execution of the plan. However, this is what is typically required by teamwork models. Instead, we allow any proxy that detects all the preconditions of a plan to instantiate that plan. Hence, notice that when a factory at any proxy notices that preconditions are met, the TOP is instanted immediately and a PlanAgent is created (see below).

Avoiding Conflicting Plans. While the distributed plan instantiation process allows the team to instantiate plans efficiently and robustly, two possible problems can occur. First, the team could instantiate different plans for the same goal, based on different preconditions detected by different members of the team. For example, two different plans could be instantiated by different factories for hitting the same target depending on what particular team members know or sense. Second, the team may initiate multiple copies of the same plan. For example, two WASMs may detect the same target and different factories instantiate identical plans to destroy the same target. While our algorithms handle conflict recognition and resolution (see PlanAgent algorithm), minimizing conflicts to start with minimizes excess communication and wasted activity. When a PlanAgent is created for a specific plan, the first thing it does is "wait to detect conflict". This involves checking CK to determine whether there are conflicting plans, since CK contains coordination knowledge and will contain information about the conflicting plans,. Clearly, there may be conflicting plans the proxy does not know about, because they are not in CK, and thus there may be a conflict, not immediately apparent to the PlanAgent.

We are currently experimenting with a spectrum of algorithms for minimizing instantiations of conflicting plans. Each of the algorithms implements the "Wait to detect conflict" part of the PlanAgent algorithm in a different way. At one end of the spectrum we have a specific, deterministic rule based on specific information about the state of the team. We refer to this instantiation rule as the *team status instantiation*

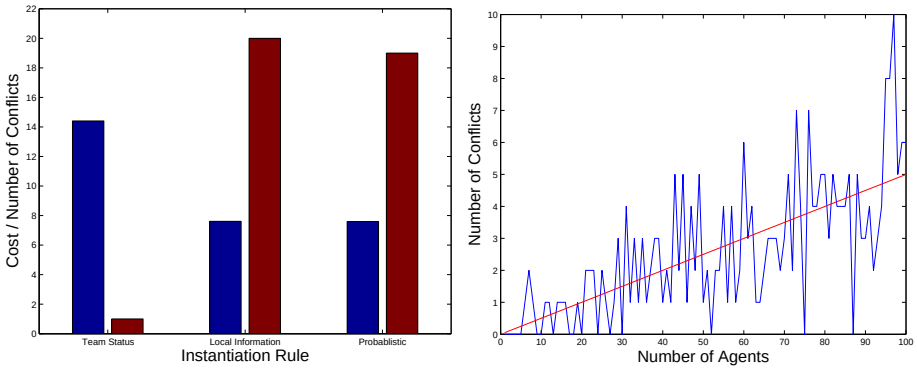
[6] We can envision more sophisticated matching algorithms and even runtime planning, however to date this has not been required.

rule. When using this rule, we attached a mathematical function to each TOP. The value of that function can be computed from information in K. For example, the function attached to the TOP for destroying a target is based on distance to the target. Unless the PlanAgent computes that the local proxy has the highest possible value for that function, it should not proceed. The advantage of this rule is that there will be no conflicts, provided that K is accurate. The disadvantage of the rule is that many InformationAgents must move around the proxies often to keep K up-to-date.

At the other end of the spectrum, we have a probabilistic rule that requires no information about other team members. This rule, which we refer to as the *probabilistic instantiation rule*, requires that the PlanAgent wait a random amount of time, to see whether another team member instantiates that plan (or a conflicting plan.) Thus, InformationAgents for newly instantiated TOPs at other proxies have some time to reach the proxy, update CK and avoid a costly conflict. The advantage of this rule, is that no information is required about other team members to use this rule, thus reducing the volume of InformationAgents required. There are two disadvantages. First, there may be conflicting plans instantiated. Second, there may be a significant delay between detection of pre-conditions and the instantiation of the plan depending on how long the PlanAgents wait.

In between these two extremes, we define another rule, which we refer to as the *local information rule*, that requires that a proxy must detect some of the TOP's preconditions locally, in order to instantiate the plan. Specifically, at least one of the TOPs preconditions must have come into K directly from the environment, rather than via an InformationAgent. Although this will lead to conflicting plans when multiple proxies locally sense preconditions, it is easier to determine where the conflicts might occur and resolve them quickly. Specifically we can look for proxies with the ability to locally sense information, e.g., those in a specific part of the enviornment. The major disadvantage of this rule is that when a TOP has many preconditions the team members that locally detect specific preconditions may never get to know all the preconditions and thus not instantiate the plan.

Figure 1.8(a) shows the result of a simple simulation of the three instantiation rules. We used simple models of the environment to work out how often InformationAgents must move around in order to implement the three rules. This "cost" is indicated by the left-hand column and uses a logarithmic scale. The right hand column shows the number of plan conflicts that result. A conflict occurs when two or more PlanAgents proceed before they have been informed that the other has



(a) **(b)**

Figure 1.8. (a) The number of plan instantiations as we increase the number of agents using the probabilistic instantiation rule. The straight line represents the average of a large number of runs. The jagged line shows output from specific runs, highlighting the high variance. (b) The number of plan instantiations using the three different rules. In this simulation, there were 200 agents and a message took 600ms to be transmitted. For the probabilistic instantiation rule, the PlanAgent would wait upto 10s.

proceeded. Clearly, the *team status* rule gives a different tradeoff between conflicts and cost than the other rules. Notice that the precise behavior of the *probabilistic rule* depends on the specific parameter settings. Figure 1.8(b) shows how many conflicts result from this approach as we increase the number of PlanAgents. The precise slope of the line depends on the amount of time the PlanAgent is willing to wait and the length of time it takes to communicate that the PlanAgent has been instantiated.

3.3. INFORMATION SHARING

Information or events sensed locally by a agent will often not be sensed by other agents in the team. In some cases, however, that information will be critical to other members of the team, hence should be communicated to them. For example, consider the case where one agent detects that a ground target has moved into some trees. It needs to inform the WASM that is tasked with destroying that target, but will typically not know which WASM that is or whether any WASM is or whether the WASM has already been informed of the move (perhaps many times). A successful information sharing algorithm needs to deliver information where it is required without over loading the communication network.

Previous algorithms for sharing information in a multiagent system have made assumptions that do not hold in very large groups of WASMs (or large teams in general). Specifically, algorithms either assume that centralization is possible[33] or assume that agents have accurate models of other members of the group[35]. Often techniques for communication assume that an agent with some potentially relevant information will have an accurate model of the rest of the group. The model of the group is used to reason about which agents to communicate the information to (and whether there is utility in communicating at all[35, 26]). However, in large groups, individual agents will have very incomplete information about the rest of the group, making the decision about to whom to communicate some information much more difficult. Moreover, both as a design decision and for practical reasons, communication in a centralized way is not appropriate.

We are developing new communication reasoning that reduces the need to know details about other team members by exploiting the fact that, even in very large groups, there is a low *degree of separation* between group members. We assume the agents have point-to-point communication channels with a small percentage ($< 1\%$) of other group members. Having a low degree of separation means that a message can be passed between any two agents via a small number of the point-to-point connections. Such networks are known as *small worlds networks* [38]. In a small worlds network, agents are separated from any other agent by a small number of links. Such networks exist among people and are popularized by the notion of "six degrees of separation"[1]. When agents are arranged in a network, having a small number of neighbours relative to the number of members in the team, the number of agents through which a message must pass to get from any agent to any other, going only from neighbour to neighbour, is typically very small.

The intuition behind our approach is that agents can rapidly get information to those requiring it simply by "guessing" which acquaintance to send the information to. The agent attempts to guess which of its neighbours either require the information or are in the best position to get the information to the agent that requires it. In a small worlds network, an agent only needs to guess correctly slightly more often than it guesses wrong and information is rapidly delivered. Moreover, due to the low degree of separation, there only needs to be a small number of correct "guesses" to get information to its destination. Since the agents are working in a team, they can use information about the current state of the coordination to inform their guesses. While members of large teams will not have accurate, up-to-date models of the team, our hypothesis

is that they will have sufficiently accurate models to "guess" correctly often enough to make the algorithm work.

InformationAgents are responsible for delivering information in our proxy architecture. Thus, these "guesses" about where to move next are made by the InformationAgents as they move around the network. The basic algorithm is shown in Figure 1.9. The InformationAgent guesses where to move next, moves there, updates the proxy state and moves on. This process continues until the information is likely to be out of date or the InformationAgent has visited enough proxies that it believes there are unlikely to be more proxies requiring the information. In practice, we typically stop an InformationAgent after it has visited a fixed percentage of the proxies, but we are investigating more optimal algorithms.

InformationAgent
while Worth Continuing
Guess which link leads closer to proxy requiring information
Move to that proxy
Add information to proxy state (either K or CK)
end while

Figure 1.9. Algorithm for an InformationAgent

To test the potential of the approach we ran an experiment where proxies are organized in a three dimensional grid. One proxy is randomly chosen as the source of some information and another is randomly picked as the sink for that information. For testing, a probability is attached to each link, indicating the chance that passing information down that link will get the InformationAgent a smaller number of links from the sink. (These probabilities need to be inferred in the real proxies, see below for details.) In the experiment shown in Figure 1.10(a) we adjust the probability on links that actually lead to an agent requiring the information. For example, for the "59%" setting, links that lead closer to the sink agent have a probability of 0.59 attached, while those that lead further away have a 0.41 probability attached. The InformationAgent follows links according to their probability, e.g., in the "59%" setting, it will take links that lead it closer to the sink 59% of the time. Figure 1.10(a) shows that the information only needs to move closer to the target slightly more that 50% of the time to dramatically reduce the number of messages required to deliver information efficiently to the sink. To test the robustness of the approach, we altered the probability on some links so that the probability of moving further from the sink was actually higher than moving toward it. Figure 1.10(b) shows that

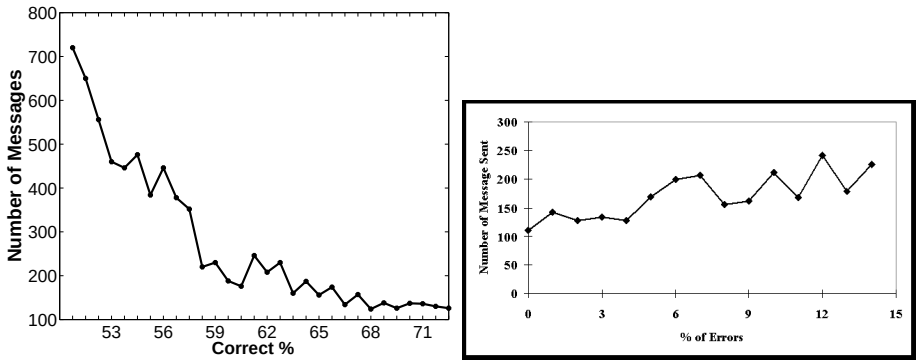


Figure 1.10. (a) The number of messages required to get a piece of information from one point in a network to another as we increase the likelihood that agents pass information closer to the target. There were 800000 agents arranged in a three dimensional grid. (b) The total number of messages required as the percentage of agents with probabilities indicating the wrong direction to send the information.

even when a quite large percentage of the links had these "errornous" probabilities, information delivery was quite efficient. While this experiment does not show that the approach works, it does show that if the InformationAgents can guess correctly only slightly more than 50% of the time, we can get targeted, efficient information delivery.

Sharing Information with InformationAgents. An initial approach to determining where InformationAgents should travel relies on inferring the need for one piece of information from the reciept of another piece. To understand the motivation for the idea, consider the following example. When a proxy receives a message about a role that is being performed at coordinates (1,1) from neighbour a, it can infer that if it found out about a SAM site at coordinates (1,2), passing that information to neighbour a is likely to get the information to a proxy that needs it. Notice, that it need not be the neighbour a that actually needs the information, but it will at least likely be in a good (or better) position to know who does. These inferences can be inferred using Bayes' Rule. In the following, we present a model of the small worlds network and an algorithm, based on Bayes' Rule, for updating where an InformationAgent should move next.

Proxy Network Models. Our proxy network model is composed of three elements, A, N and I, where A are the proxies, N is the network between the agent and I is the information to be shared. The team consists of a large number of proxiess, $A(t) = \{a_1, a_2, ..., a_n\}$.

N denotes the communication of network among proxy team. An proxy a can only communicate directly with a very small subset of its team mates. The acquaintances, or neighbours, of a at time t are written $n(a,t)$ and the whole network as $N(t) = \bigcup_{a \in A(t)} n(a,t)$. A message can be transferred from proxies that are not neighbours by passing through intermediate proxies but proxies will not necessarily know that path. We define the minimum number of proxies a message must pass through to get from one agent to another as the *distance* between those agents. The maximum distance between any two proxies is the network's "degree of separation". For example, if proxies a_1 and a_2 are not neighbours, but share a neighbour $distance(a_1, a_2) = 1$. We require the network, N, to be a small worlds network, which imposes two constraints. First, $|n(a,t)| < K$, where K is a small integer, typically less than 10. Second, $\forall a_i, a_j \in A, distance(a_i, a_j) < D$ where D is a small integer, typically less than 10. While N is a function of time, we assume that it typically changes slowly relative to the rate messages are sent around the network.

I is the alphabet of information that the team knows, $I = CK \cup K$. $i \in I$ denotes a specific piece of information, such as "There is a tank at coordinates (12, 12)".

The internal state of the team member a is represented by $S_a =< H_a, P_a, K_a >$. H_a is the history of messages received by the proxy. In practice, this history may be truncated to leave out old messages for spaces reasons. $K_a \subseteq I$ is the local knowledge of the proxy (it can be derived from H_a). If $i \in K_a$ at time t we say $knows(a, i, t)$.

The matrix P is the key to our information sharing algorithm.

$$P : I \times N(a) \rightarrow [0, 1]$$

P maps an proxy and piece of information to a probability that that proxy is the best to pass that piece of information to. To be "best" means that passing the information to that proxy will most likely get the information to a sink. For example, if $P[i_1, a_2] = 0.9$, then given the current state of a_1 suggests that passing information i_1 to proxy a_2 is the best proxy to pass that information to. To obey the rules of probability, we require:

$$\forall i \in I, \sum_{b \in N(a)} P[i, b] = 1$$

Using P, when the proxy has a piece of information to send, it chooses a proxy to send the message to according to the the likelihood sending to that proxy is the best. Notice, that it will not always send to the best proxy, but will choose an proxy relative to its probability of being the best.

The state of a proxy, S_a, gets updated in one of three ways. First, local sensing by the proxy can add information to K_a. Second, over time the information in K_a changes as information becomes old. For example, information regarding the location of an enemy tank becomes more uncertain over time. Maintaining K_a over time is an interesting and difficult challenge, but not the focus of this paper, hence we ignore any such effects. Finally, and most importantly, K_a changes when a message m is sent to the proxy a from another proxy b at time t, $sent(m, a, b, t)$. In the case that m contains a piece of information i, we define a transition function, δ, that specifies the change to the proxy state. Two parts of the δ function, namely the update to the history part of the state, $H_a(t+1) = H_a(t) \cup m$, and the knowledge part of the state, $K_a(t+1) = K(t) \cup i$, are trivial. The other part of the transition function, the update to P_a due to message m, is written δ_P. This is the most difficult part of the transition function, and is the key to the success of the algorithm. The function is discussed in detail in later sections.

The reason for sharing information between team mates is to improve the individual performance and hence the overall performance. To quantify the importance of a piece of information i to a proxy a at time t we use the function $R : I \times A \times t \rightarrow \mathcal{R}$. The importance of the information i is calculated by determining the expected increase in utility of the proxy with the information versus without it. That is, $R(a, i, t) = EU(a, K+i) - EU(a, K-i)$, where $EU(a, K)$ is the expected utility of the proxy a with knowledge K. When $R(a, i, t) > 0$, it means that the specific information i supports a's decision making. The larger the value of $R(a, i, t)$ the more a needs the information.

$O(A, I, N)$ is the objective function:

$$reward(A, t) = \frac{\sum_{a \in A(t)} r(a, i, t)}{\sum_{a \in A(t)} knows(a, i, t)}$$

The numerator sums the reward recieved for getting the information to proxies that need it, while the denominator gives the total number of agents to whom the information was given. Intuitively, the objective function is maximized when information is transferred to as many as possible proxies that need that information and as few as possible of those that do not.

Updating Proxy Network Models. The key question for the algorithm is how we define δ_P, i.e., how we update the matrix P when a new message arrives. To update where to send a piece of information j based on a message containing information i, we need to know the relationship, if any between those pieces of information. Such relationships are domain dependant, hence we assume that a relationship function, $rel(i,j) \rightarrow [0,1]$, is given. The intuition caputured by rel is that if $rel(i,j) > 0.5$ then an agent interested in i will also be interested in j, while if $rel(i,j) < 0.5$ then an agent interested in i is unlikely to be interested in j. For example, if i corresponds to a particular event in the environment, if j corresponds to an event near the event at i, we can expect $rel(i,j) > 0.5$, otherwise we expect a smaller value. If there is no relationship between i and j, then $rel(i,j) = 0.5$.

Utilizing Bayes' rule, we interpret a message containing information i arriving from a proxy b as *evidence* that proxy b is the best associate to pass information j to. Specifically, we can define define δ_P as follows:

$$\delta_P(P, recv(i,a)) = Pr(P[j,b]|recv(i,a)) \times P[j,b]$$

where

$$Pr(P[j,b]|recv(i,a)) = \begin{cases} rel(i,j) \times \frac{2}{|N|} & \text{if } a = b \\ \frac{1}{|N|} & \text{otherwise} \end{cases}$$

After δ_P has been applied, P must again be normalized:

$$P'[i,a] = \frac{P[i,a]}{\sum_{b \in N(a)} P[i,b]}$$

4. RESULTS

The most important aspect of our results is that we have run a team of 200 simulated WASMs, controlled by proxies in a simulation of a mission to protect a manned aircraft. Such a team is an order of magnitude bigger than previously published teams. The proxies are implemented in Java and 200 ran on two 2GHz linux machines with 1Gb of RAM on each machine. In the following, we present selected runs from experiments with this scenario, plus the results of experiments using a simpler centralized controller that mimics the coordination, but is more lightweight.

The first experiment compared three different algorithms for allocating WASMs to targets. We compared two centralized algorithms with our distributed allocation. The first centralized algorithm was very simple, allocating the closest available WASM to every newly discovered

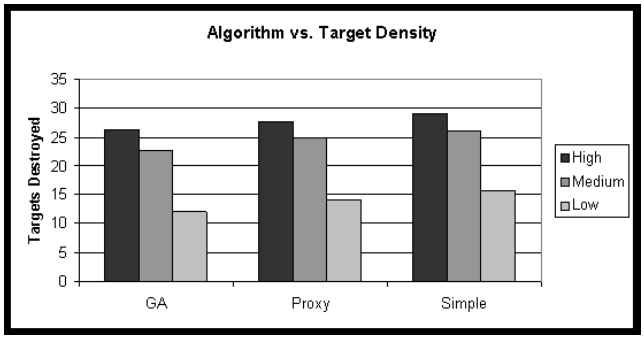
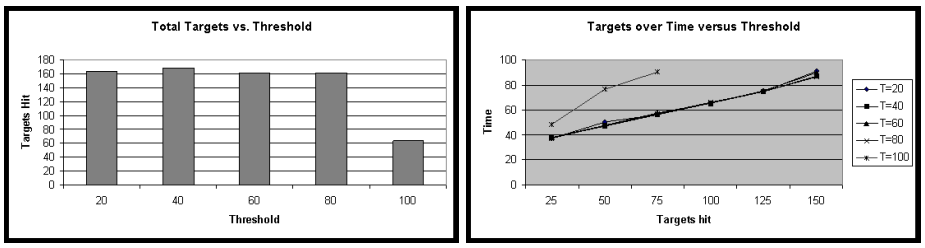


Figure 1.11. Comparing the number of targets hit by three different role allocation algorithms under three different target densities.

target. The second centralized algorithm was a genetic algorithm based approach. Figure 1.11 shows the number of randomly distributed targets destroyed by each of the algorithms in a fixed amount of time. For each algorithm we tried three different levels of target density, few targets spread out to many targets in a small area. Somewhat surprisingly, the simple algorithm performed best, followed by our distributed algorithm, finally followed by the genetic algorithm. It appears the the random distribution of targets is especially amenable to simple allocation algorithms. However, notice that the performance of the distributed algorithm is almost as good as the simple algorithm, despite having far lower communication overheads. We then performed more detailed experiments with the distributed algorithm, varying the threshold for accepting a role to destroy a target. The threshold is inversely proportional to the distance of the WASM to the target. A team member will not accept a role unless its capability is above the threshold and it has available resources. Figure 1.12(a) shows that unless the threshold is very high and WASMs will not go to some targets, the number of targets hit does not vary. Even the rate of targets hit over time does not change much as we vary the thresholds, see Figure 1.12(b).

In our second experiment, we used the centralized version of our teamwork algorithms to run a very large number of experiments to understand how WASMs should coordinate. The mission was to protect a manned aircraft and the output measure was the closest distance an undestroyed target got to the manned aircraft (higher is better) which followed a random path. The WASMs had two options, stay with the aircraft or spread out ahead of the aircraft path. We varied six parameters, giving them low, medium and high values and performed over 8000 runs.



(a) **(b)**

Figure 1.12. (a) The number of targets hit as the threhold is varied. Threshold is the minimum capability of an WASM assigned a target and is inversely proportional to the WASMs distance from the target. (b) The time taken to hit a specific number of targets as the threshold is varied.

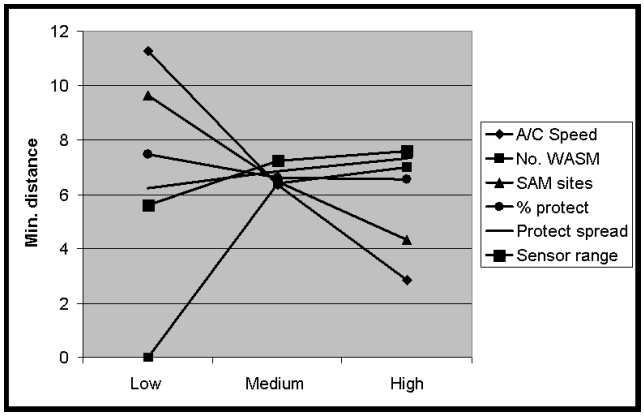


Figure 1.13. Effects of a variety of parameters on the minimum distance a SAM site gets to a manned aircraft the WASMs are protecting.

The first parameter was the speed of the aircraft relative to the WASM (A/C Speed). The second parameter was the number of WASMs (No. WASM). The third parameter was the number of targets (SAM sites). The fourth parameter was the percentage of WASMs that stayed with the aircraft versus the percentage that spread out looking for targets. The fifth parameter is the distance that the WASMs which stayed with the aircraft flew from it (Protect Spread). Finally, we varied the WASM sensor range. Figure 1.13 shows the results. Notice the speed of the aircraft relative to the WASMs is one of the most critical factors, alongside the less surprising Number of WASMs.

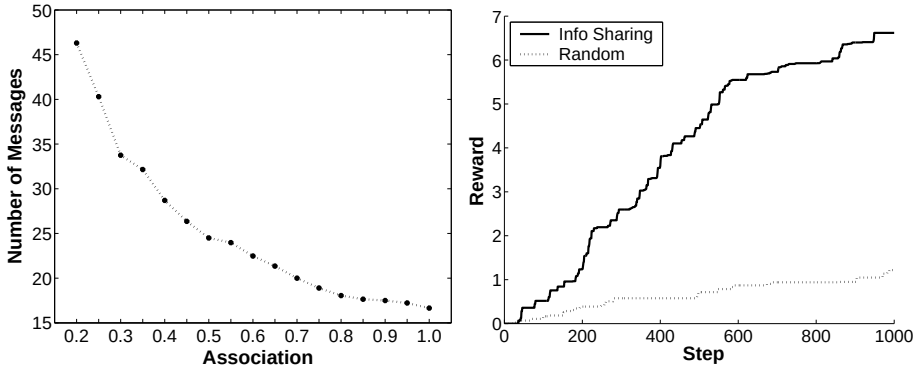


Figure 1.14. (a) The reduction in the number of messages as the association between information received and information to be sent increases. (b) The reward received over time, based on our information sharing algorithm and on a random information passing algorithm.

Finally, we ran two experiments to evaluate the information sharing algorithm. In the first experiment, we arranged around 20000 agents in a small worlds network. Then we passed 150 pieces of information from a particular source randomly around the network. After these 150 pieces of information had been sent, we created a new piece of information randomly and applied our algorithms to get it to a specific sink agent. In Figure 1.14(a) we show the average number of steps taken to deliver the message from the source to the sink as we varied the strength of the relationship between the information originally sent out and the new piece of information. As expected, the stronger the relationship between the originally sent information and the new information the better the information delivery. In the second experiment, we started information from various sources, moving them 150 steps, as in the first experiment. In this case, there were multiple "sinks" for the piece of information that we randomly added. The reward recieved, based on the objective function above, is proportional to the ratio of the number of agents receiving the information that wanted it and the number that did not need it. Figure 1.14(b) shows that our algorithm dramatically outperforms random information passing. While important work remains, the initial information sharing experiments show the promise of our approach.

5. RELATED WORK

Coordination of distributed entities is an extensively studied problem[7, 6, 21, 25, 34]. A key design decision is how the control is distributed amoung the group members. Solutions range from completely centralized[11], to hierarchical[10, 17] to completely decentralized[39]. While there is not yet definitive, empirical evidence of the strengths and weaknesses of each type of architecture, it is generally considered that centralized coordination can lead to behavior that is closer to optimal, but more distributed coordination is more robust to failures of communications and individual nodes[2]. Creating distributed groups of cooperative autonomous agents and robots that must cooperate in dynamic and hostile environments is an huge challenge that has attracted much attention from the research community[22, 24]. Using a wide range of ideas, researchers have had moderate success in building and understanding flexible and robust teams that can effectively act towards their joint goals[5, 8, 18, 31].

Tidhar [37] used the term "team-oriented programming" to describe a conceptual framework for specifying team behaviors based on mutual beliefs and joint plans, coupled with organizational structures. His framework also addressed the issue of team selection [37] — team selection matches the "skills" required for executing a team plan against agents that have those skills. Jennings's GRATE* [18] uses a teamwork module, implementing a model of cooperation based on the joint intentions framework. Each agent has its own *cooperation level* module that negotiates involvement in a joint task and maintains information about its own and other agents' involvement in joint goals. The Electric Elves project was the first human-agent collaboration architecture to include both proxies and humans in a complex environment[5]. COLLAGEN [30] uses a proxy architecture for collaboration between a single agent and user. While these teams have been successful, they have consisted of at most 20 team members and will not easily scale to larger teams.

Jim and Giles[20] have show that communication can greatly improve multiagent system performance greatly by analyzing a general model of multi-agent communication. However, these techniques rely on a central message board. Burstein implemented a dynamic information flow framework and proposed an information delivery algorithm based on two kinds of information communication: Information Provision advertisements and Information Requirements advertisements[4]. But its realization was based on broadcast or using middle agents as brokers who respond to all the information disseminated. Similar research can be found in Decker and Sycara's RETSINA multiagent system[9, 13]

which defined information agent and middle agent who was supposed to be able to freely delivery information with any of the others without delay. Such approaches, while clearly useful for some domains, are not applicable to large scale teams.

Yen's CAST proposed a module that expedites information exchange between team members based on a shared mental model, but almost the same shortcoming exists because in a huge team who is working in an ad-hoc environment, any team member can only sense a very limited number of teammates' status as well as their mental[42]. Xuan[41] and Goldman[15] proposed a decentralized communication decision model in multi-agent cooperation based on Markov decision processes (MDP). Their basic idea is that an explicit communication action will incur a cost and they assume the global reward function of the agent team and the communication cost and reward are known. Xuan used heuristic approaches and Goldman used a greed meta-level approaches to optimize the global team function. Moveover, Goldman[14] put forward a decentralized collaborative multiagents communication model and mechanism design based on MDP, which assumed that agents are full-synchronized when start operating, but no specific optimal algorithm was presented. Furthermore, there are no experiment result was shown that their algorithm can work on huge team very well.

Bui[3] and Wie[40] solved the information sharing problems in novel ways. In Bui's work, he presented a framework for team coordination under incomplete information based on the theory of incomplete information game that agents can learn and share their estimates with each other. Wie's RHINO used a probability method to coordinate agent team without explicit communication by observing teammates' action and coordinating their activities via individual and group plan inference. The computational complexity of these approaches makes them inapplicable to large teams.

6. CONCLUSIONS AND FUTURE WORK

In this Chapter we have presented a novel approach and initial results to the challenges presented by coordination of very large groups of WASMs. Specifically, we presented Machinetta proxies as the basic arhictecture for flexible, robust distributed coordination. Key coordination algorithms encapsulated by the proxies were presented. These algorithms, including plan instantiation and information sharing, address new challenges that arise when a large group is required to coordinate. These novel algorithms replace existing algorithms that fail to scale when the group involves a large number of entities. We imple-

mented the proxies using the novel abstraction of coordination agents, which gave us high levels of robustness. With the novel algorithms and architecture we were able to execute scenarios involving 200 simulated WASMs flying coordinated search and destroy missions.

Our initial experiments reveal that while our algorithms are capable of dealing with some of the challenges of the domain, many challenges remain. Perhaps more interestingly, new unexpected phenomena are observed. Understanding and dealing with these phenomena will be a central focus of future efforts. Further down the track, the coordinated behavior must be able to adapt strategically in response to the tactics of the hostile forces. Specifically, it should not be possible for enemy forces to exploit specific phenomena of the coordination, the coordination must react to such attempts by changing their coordination. Such reasoning is currently far beyond the capabilities of large teams.

References

[1] Albert-Laszla Barabasi and Eric Bonabeau. Scale free networks. *Scientific American*, pages 60–69, May 2003.

[2] Johanna Bryson. Hierarchy and sequence vs. full parallelism in action selection. In *Intelligent Virtual Agents 2*, pages 113–125, 1999.

[3] H. H. Bui, S. Venkatesh, and D. Kieronska. A framework for coordination and learning among team members. In *Proceedings of the Third Australian Workshop on Distributed AI*, 1997.

[4] Mark H. Burstein and David E. Diller. A framework for dynamic information flow in mixed-initiative human/agent organizations. *Applied Intelligence on Agents and Process Management*, 2004. Forthcoming.

[5] 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.

[6] D. Cockburn and N. Jennings. *Foundations of Distributed Artificial Intelligence*, chapter ARCHON: A Distributed Artificial Intelligence System For Industrial Applications, pages 319–344. Wiley, 1996.

[7] Philip R. Cohen and Hector J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

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

[9] K. Decker, K. Sycara, A. Pannu, and M. Williamson. Designing behaviors for information agents. In *Procs. of the First International Conference on Autonomous Agents*, 1997.

[10] Vincent Decugis and Jacques Ferber. Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. In *Proceedings of the Second International Conference on Autonomous Agents*, 1998.

[11] T. Estlin, T. Mann, A. Gray, G. Rapideau, R. Castano, S. Chein, and E. Mjolsness. An integrated system for multi-rover scientific exploration. In *Proceedings of AAAI'99*, 1999.

[12] 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.

[13] Joseph Giampapa and Katia Sycara. Team oriented agent coordination in the RETSINA multi-agent system. In *Proceedings of Agents02*, 2002.

[14] C. V. Goldman and S. Zilberstein. Mechanism design for communication in cooperative systems. In *Fifth Workshop on Game Theoretic and Decision Theoretic Agents*, 2003.

[15] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-agent Systems*, 2003.

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

[17] Bryan Horling, Roger Mailler, Mark Sims, and Victor Lesser. Using and maintaining organization in a large-scale distributed sensor network. In *In Proceedings of the Workshop on Autonomy, Delegation, and Control (AAMAS03)*, 2003.

[18] N. Jennings. The archon systems and its applications. Project Report, 1995.

[19] N. R. Jennings. Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *Intl. Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.

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

[21] David Kinny. The distributed multi-agent reasoning system architecture and language specification. Technical report, Australian Artificial intelligence institute, Melbourne, Australia, 1993.

[22] 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.

[23] Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsushi Shinjoh, and Susumu Shimada. Robocup rescue: Searh and rescue in large-scale disasters as a domain for autonomous agents research. In *Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics*, volume VI, pages 739–743, Tokyo, October 1999.

[24] John Laird, Randolph Jones, and Paul Nielsen. Coordinated behavior of computer generated forces in TacAir-Soar. In *Proceedings of the fourth conference on computer generated forces and behavioral representation*, pages 325–332, Orlando, Florida, 1994.

[25] V. Lesser, M. Atighetchi, B. Benyo, B. Horling, A. Raja, R. Vincent, T. Wagner, P. Xuan, and S. Zhang. The UMASS intelligent home project. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 291–298, Seattle, USA, 1999.

[26] David Pynadath and Milind Tambe. Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, 2002.

[27] 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.

[28] 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.

[29] Paul Ranky. *An Introduction to Flexible Automation, Manufacturing and Assembly Cells and Systems in CIM (Computer Integrated Manufacturing), Methods, Tools and Case Studies*. CIMware, 1997.

[30] C. Rich and C. Sidner. COLLAGEN: When agents collaborate with people. In *Proceedings of the International Conference on Autonomous Agents (Agents'97)"*, 1997.

[31] P. Rybski, S. Stoeter, M. Erickson, M. Gini, D. Hougen, and N. Papanikolopoulos. A team of robotic agents for surveillance. In *Proceedings of the fourth international conference on autonomous agents*, pages 9–16, 2000.

[32] P. Scerri, D. V. Pynadath, L. Johnson, Rosenbloom P., N. Schurr, M Si, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[33] Daniel Schrage and George Vachtsevanos. Software enabled control for intelligent uavs. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, Hawaii, August 1999.

[34] Munindar Singh. Developing formal specifications to coordinate hetrogeneous agents. In *Proceedings of third international conference on multiagent systems*, pages 261–268, 1998.

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

[36] Milind Tambe, Wei-Min Shen, Maja Mataric, David Pynadath, Dani Goldberg, Pragnesh Jay Modi, Zhun Qiu, and Behnam Salemi. Teamwork in cyberspace: using TEAMCORE to make agents team-ready. In *AAAI Spring Symposium on agents in cyberspace*, 1999.

[37] G. Tidhar, A.S. Rao, and E.A. Sonenberg. Guided team selection. In *Proceedings of the Second International Conference on Multi-Agent Systems*, 1996.

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

[39] Tony White and Bernard Pagurek. Towards multi swarm problem solving in networks. In *Proceedings of the International conference on multi-agent systems*, pages 333–340, Paris, July 1998.

[40] Michael Van Wie. A probabilistic method for team plan formation without communication. In *Proceedings of the fourth international conference on Autonomous agents*, 2000.

[41] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.

[42] J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz. Cast: Collaborative agents for simulating teamwork. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1135–1142, 2001.