Challenges in Building Very Large Teams

Paul Scerri*, Yang Xu+, Jumpol Polvichai+, Bin Yu*, Steven Okamoto*, Mike Lewis+ and Katia Sycara*

*Carnegie Mellon University
+University of Pittsburgh

Abstract. Coordination of large numbers of unmanned aerial vehicles is difficult due to the limited communication bandwidth available to maintain cohesive activity in a dynamic, often hostile and unpredictable environment. We have developed an integrated coordination algorithm based on the movement of *tokens* around a network of actors. Possession of a token represents exclusive access to the task or resource represented by the token or exclusive ability to propogate the information represented by the token. The movement of tokens is governed by a local decision theoretic model that determines what to do with the tokens in order to maximize expected utility. The result is effective coordination between large numbers of UAVs with very little communication. However, the overall movement of tokens can be very complex and, since it relies on heuristics, configuration parameters need to be tuned for a specific scenario or preferences. To allow rapid tuning of the configuration for a particular scenario, we have developed a neural network model of the relationship between configuration and environment parameters and performance. A human uses the model to rapidly configure a team or even reconfigure the team online, as the environment changes.

1 Intro

Efficient, effective automated or semi-automated coordination of large numbers of cooperative heterogeneous software agents, robots and humans has the potential to revolutionize the way complex tasks are performed in a variety of domains. From military operations, to disaster response[26, 58] to commerce[35] to space[18], automated coordination can decrease operational costs, risk and redundancy while increasing efficiency, flexibility and success rates. To achieve this promise, scalable algorithms need to be found for the key problems in coordination. Unfortunately, these problems, including deciding how to allocate tasks and resources, deciding when and what to communicate and planning, have NP-complete or worse computational complexity and thus require approximate solutions.

While automated coordination is a very active research area (e.g., [56, 61, 19]), previous work has failed to produce algorithms or implementations that meet the challenges of coordinating large numbers of heterogeneous actors in complex environments. Most algorithms developed for key problems do not scale to very large problems (e.g., optimal search techniques[32, 33]), though some

scale better than others (e.g., markets [18]). The rare algorithms that do scale effectively typically either make unreasonable assumptions or are very specialized for a specific problem (e.g., emergent behavior [45]). Algorithms that have been shown to be scalable often rely on some degree of centralization[31], which may not always be desirable or feasible.

The solution relies on three novel ideas. The first novel idea is to use *tokens*, encapsulate both information and control, as the basis for all coordination. For each aspect of the coordination, e.g., each task to be assigned, there is a token representing that aspect. Control information, included with the token, allows actors to locally decide what to do with the token. For example, a task token contains control information allowing an actor to decide whether to perform the task or pass it off for another actor. Movement of tokens from actor to actor, implements the coordination. The effect of encapsulating the piece of the overall coordination with its control information is to ensure that everything required for a particular decision, e.g., whether a particular actor should perform a particular task, is localized to the actor currently holding the respective token. This reduces required communication and avoids many of the problems that arise when required information is distributed. The key is to be able to find control rules and control information that can be encapsulated in the token and allow decisions about that token to be made relatively independantly of decisions about other tokens. We have developed such algorithms for a range of key coordination problems including task allocation[53], reactive plan instantiation[52], resource allocation, information sharing[63] and sensor fusion.

The efficiency of the token based coordination algorithms depends on the routing of the tokens around the network of actors. Individual actors build up local models of what types of things their neighbors in the network are most interested in and use these models to decide where to send a token (if at all)[64]. Previous work has shown that even relatively poor (i.e., often inaccurate) local routing models can dramatically improve overall performance of a particular coordination algorithm[63]. The second novel idea in this work is to exploit the homogeneity of tokens, i.e., the fact that all aspects of coordination are represented with tokens, to allow actors to exploit the movement of tokens for one coordination algorithm to improve the flow of tokens for another coordination algorithm. The inituition is that, e.g., knowing something about a task allocation should help the resource allocation which should in turn help dissemination of important information. For example, if actor A knows that actor B is performing a fire fighting task in Pittsburgh, it can infer that resources physically located in Phillidelphia will not be of interest to actor B and save the overhead of involving actor B in algorithms for allocating those resources. Our results show that exploiting synergies between coordination algorithms dramatically improves overall coordination performance.

The third novel idea in this work is to develop a general purpose *meta-level reasoning* layer, distributed across the team. The meta-reasoning is conceptually "above" the token flows and manipulates the movement of tokens in one of two ways. First, the meta-reasoning layer can extract and manipulate particu-

lar tokens when behavior is not as required. For example, the meta-reasoning layer may notice that a particular task allocation token is unable to find any actor to perform the task it represents and bring that unfilled task assignment to the attention of a human who can decide what to do. Tokens can be safely extracted because everything related to that token is encapsulated within the token. Second, the meta-reasoning layer can configure control parameters on all tokens to manipulate the token flows to maximize current performance requirements. For example, when communication bandwidth is tightly constrained the meta-reasoning can configure the tokens to move less (at a cost of quality of performance.) We use a neural network model of the relationships between control parameters and performance to allow rapid search for parameter settings to meet current performance requirements and online reconfiguration to adapt to changing requirements.

In previous work, we have shown that by leveraging a logical, static network connecting all coordinated actors, some coordination algorithms can be made to effectively scale to very large problems[49]. The approach used the network to relax some of the requirements to communicate and made it possible to apply theories of teamwork[9, 56] in large groups. The key was the *small worlds* property of the network[59] which requires that any two agents are connected by a small number of intermediate agents, despite having a relatively small number of direct neighbors. The token-based algorithms leverage this network when moving around the team

The integrated token-based approach has been implemented both in abstracted simulation environments and as a part of domain independant coordination software called Machinetta[50]. The abstract simulation environments show the effectiveness of the token based ideas across a very wide range of situations. In this simulation environment, the token-based algorithms have been shown to be extremely scalable, comfortably coordinating groups of up to 5000 actors. Machinetta is a public domain software module that has been used in several domains[50, 49, 54], including for control of unmanned aerial vehicles. Machinetta uses the concept of a *proxy*[42, 23] which gives each actor a semi-autonomous module encapsulating the coordination reasoning. The proxies work together in a distributed way to implement the coordination. Experiments with upto 200 Machinetta proxies running the token based algorithms have shown that fully distributed coordination feasible.

2 Problem Description

Target Application: Coordinated Wide Area Search Munitions Our current domain of interest is coordination of large groups of Wide Area Search Munitions (WASMs). 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[7, 12]. A driving example for our work is for teams of WASMs to be launched from 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 making it vulnerable to attack. The WASMs will be launched as the AC-130s enter the battlespace. The WASMs will protect the flight path of the manned aircraft into the area of operations of the special forces, destroying ground based threats as required. Once an 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 each AC-130 responsible for monitoring and controlling the WASMs. Hence, high levels of autonomous operation and coordination are required of the WASMs themselves. However, because the complexity of the battlefield environment and the severe consequences of incorrect decisions, it is expected that human experience and reasoning will be extremely useful in assisting the team in effectively and safely achieving their goals.

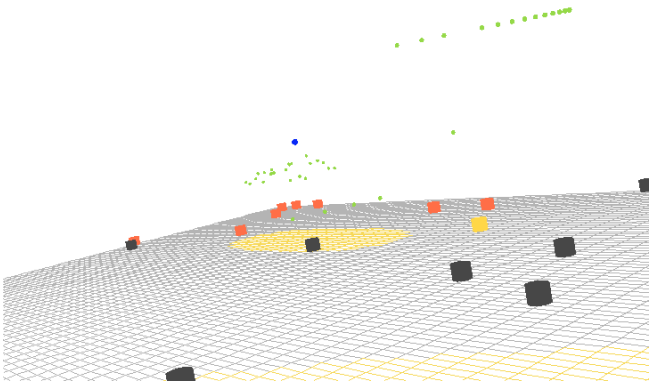

Fig. 1. A screenshot of the WASM coordination simulation environment. A large group of WASMS (small spheres) are flying in protection of a single aircraft (large sphere). Various SAM sites (cylinders) are scattered around the environment. Terrain type is indicated by the color of the ground.

Many other operations are possible for WASMs, if issues related to coordinating large groups can be adequately resolved. 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 also 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. 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[26], as does intelligent manufacturing[44] and business processes.

The problem of coordination we are dealing with here can be informally described as determining who does what at which time and with which shared resources and information. In the following, we provide a formal description of this coordination problem.

2.1 Team Oriented Plans and Joint Activities

Each member of the team $a \in A$ has a copy of the *Team Oriented Plan* templates, $Templates$ that describe the joint activities that need to be undertaken in particular situations[43]. These templates are defined offline by a domain expert. Each template, $template \in Templates$ has preconditions, $template_{pre}$ under which it should be instantiated into a *joint activity*, α_i. The template may also have parameters, $template_{param}$ that encode specifics of a particular instance. The same template may be instantiated multiple times when with different parameters. The joint activity should be terminated when certain postconditions, $template_{post}$ are met. These postconditions may be a function of $template_{param}$. The templates whose preconditions but not postconditions are satisfied at time t are written $JointActs(t, Templates)$.

A joint activity, α_i, breaks a complex activity down into tasks, $Tasks(\alpha_i) = \{task_i^1, \ldots, task_i^n\}$, each intended to be performed by a single actor. Constraints, $constraints(\alpha_i)$, exist between the tasks including constraints on the sequencing of tasks, the simultaneous (or not) execution of tasks and whether tasks are alternative ways of doing the same thing. The set of roles that should be executed at time t to achieve α_i, given the constraints, at time t is written $CurrTasks(\alpha_i, t) = f(Tasks(\alpha_i), Constraints(\alpha_i))$. Each team member has a capability to perform each task, which is written as $capability(a, task, t)$. This capability may change over time as, e.g., an agent moves around the environment. Notice that the actual value of assigning a particular actor to a particular task depends also on which resources and information that actor has, as described beow. A $template_i$ does not specify which actor should perform which task nor which resources will be used nor what what coordination must take place[43].

For example, in a disaster response domain there may be a template for evacuating a burning building, $template_{evacuate}$. The template will be instan-

tiated when there is a building on fire containing civilians, i.e., $template_{pre} = FireInBuildingX$ and $CiviliansInBuildingX$ and parameterized with the specific building to be evacuated, i.e., $template_{param} = BuildingX$. The template breaks the evacuation job down into individual activities for checking each floor and for broadcasting a message over the building's emergency broadcast system. Fire fighters will have a different inherent capability to clear floors of the building than robots.

Associates Network The *associates network* arranges the whole team into a small worlds network defined by $N(t) = \bigcup_{a \in A} n(a)$, where $n(a)$ are the *neighbors* of agent a in the network. The minimum number of *agents* a message must pass through to get from one agent to another via the associates network is the *distance* between those agents. For example, if agents a_1 and a_3 are not neighbors but share a neighbor $distance(a_1, a_3) = 1$. We require that the network be a small worlds network, which imposes two constraints. First, $\forall a \in A, |n(a)| < K$, where K is a small integer, typically less than 10. Second, $\forall a_i, a_j \in A, distance(a_i, a_j) < D$ where D is a small integer, typically less than 10.

2.2 The Value of Information

Events and circumstances in the environment are represented discretely as beliefs, $b \in Beliefs$. Individual actors will not necessarily know all current beliefs, but typically some subset $K_a \in Beliefs$. If the environment is fully observable then $\bigcup_{a \in A} K_a = Beliefs$ otherwise $\bigcup_{a \in A} K_a \subset Beliefs$. When an actor is assigned to a task (see below) having knowledge of particular beliefs can improve how well the actor can perform the task. The value of a piece of information is dependant on the environment, the task, the actor and time and is written $value(b, a, task, time) \rightarrow \mathcal{R}$. For example, when a robot with vision based sensing is assigned to search a building for trapped civilians, knowledge of where smoke is in the building is less important than to a robot using infrared or acoustic sensors (listening for voices) than using vision. While in general the mapping between tasks, agents, information and value is very complex, in practical applications it is often straightforward to find reasonable, compact approximations.

2.3 Resources

To perform assigned tasks, actors may need *resources*, $Resources = \{r_1, \ldots, r_n\}$. In this work, resources are modeled as being freely assignable to any actor and never being exhausted, however only one actor can have access to a resource at any one time[1]. A task's need for a resource is modeled as being independant of which actor is performing the task. Often there is a set of resources

[1] If multiple actors can access the same resources simultaneously, we represent this is being multiple resources

that are interchangable, in so far as any one of the resources is just as effective as any of the others. Such sets are written $IR = \{r_i, \ldots, r_n\}$. Some interchangable resources, $task_{need} = \{IR_1, \ldots, IR_m\}$, are necessary for execution of the task. This means that without access to at least one resource from each $IR \in task_{need}$ no actor can execute this task. Another set of interchangable resources, $task_{useful} = \{IR_1, \ldots, IR_k\}$ are useful to the execution of the task, although the task can be executed without access to one of the interchanable resources.

Assignment of a resource, r, to actor, a, is written $assigned(r, a)$. The resources assigned to an actor are $resouces(a)$. Since a resource cannot be assigned to more than one agent, we require $\forall a, b \in A, a \neq b, resources(a) \cap resources(b) = \emptyset$.

Consider a task for a fire fighter to extinguish a small fire. To perform this task, any fire fighter must have a hose and access to one of the fire hydrants within range of the fire. All possible hoses are modeled as a single interchangable, necessary resource. Any fire proof suit will allow the fire fighter to get closer to the fire, which improves their ability to fight the fire, but is not essential. All possible fire suits are modeled as interchangable, useful resources.

2.4 Assignments and Optimization

As stated informally above, the basic aim of coordination is to decide who does what, when with which resources. This first requires determining what templates should be instantiated into joint activities, $\alpha_1, \ldots, \alpha_n$. The joint activities define the current set of tasks that need to be assigned to actors. Any templates that are not instantiated, but should be, because their preconditions are satisfied or should be terminated because their postconditions are satisfied, cost the team value. Performance of any tasks for joint activities that should have been terminated provide no value to the team. As described above, the joint activities that should be executed at time t are $JointActs(t, Templates)$.

$Tasks!(t) = \bigcup_{\alpha \in JointActs(t, Templates)} CurrTasks(\alpha, t)$ defines the set of tasks that give value to the team if assigned to capable team members at time t. The set of tasks that are assigned to an actor, a, is written $tasks(a)$. As with resources, tasks should be assigned to only one actor, hence $\forall a, b \in A, a \neq b, tasks(a) \cap tasks(b) = \emptyset$. The value an actor provides to the team is a function of the tasks assigned to it, the resources assigned to it and the information it knows, $contrib(a, tasks(a), resources(a), K_a, t) \rightarrow \mathcal{R}$. This function can be a complex function since interactions between tasks and knowledge can be very intricate, but in practice simple linear functions are often used to approximate it. In the case that the task $t \notin Tasks!(t)$ the agent team can recieve no value for the execution of the task. The overall coordination problem can be described as:

$$\int_{t=0}^{\infty} \sum_{a \in A} contrib(a, tasks(a), resources(a), K_a, t) - communicationCost \quad (1)$$

Typically, communication between actors is not free or is limited in some way (e.g., total volume). Optimization of Equation 1 should be performed taking into account these communications limitations.

3 Algorithms

To implement coordination in a large team we encapsulate anything that needs to be shared in a *token*. Specifically, tokens represent any belief that needs to be shared, any assignable task or any shared resource. Tokens cannot be copied or duplicated, but can be passed from actor to actor along links in the network connecting them. A token, Δ, contains two types of information *content* and *control*. The content component describes the belief, task or resource represented by the token. The control component captures the information that is required to allow each agent decide whether to keep or pass on the token to maximise the expected value to the team. The precise nature of the control component depends on the type of token, e.g., role or resource, and is discussed in more detail below. However, common to all is the path the token has followed through the team, denoted $\Delta.path$. In the remainder of this section, we describe how key coordination algorithms are implemented via the use of tokens. Notice, below when an actor decides to move a token to another actor it calls $Pass$. In the next section, we describe how the $Pass$ function sends the token from a to the $a \in n(a)$ that is most likely to benefit from reciept of the token, e.g., most likley to be able to use the resource represented by a resource token.

Information Sharing Members of a large team will commonly locally sense information that is useful to the execution of another agents tasks. The value of this information to a team member executing a task was formally defined in Section 2.2. However, it is not necessarily the case that the agent locally sensing the information will know which of its teammates needs information or even that a teammate needs it at all. We have developed a token-based algorithm for proactive sharing of such information that efficiently gets the information to any agent that needs it. The algorithm is described in detail in [63]. The control information for an information token is simply the number of hops through the team that the information token will be allowed to make before it is assumed that any team mate that needs the information actually has it. Algorithm 1 provides the pseudo code for local processing of an information token.

Algorithm 1: Information Token Algorithm

```
(1)
        if token.TTL > 0
(2)         token.TTL − −
(3)         Pass(token)
```

Where $token.TTL$ is the number of remaining hops the token can take. Efficient values for $token.TTL$ are determined emperically and tend to be approximately $log(|A|)$.

Template Instantiation and Joint Activity Deconfliction To instantiate a plan template, $template_i$, into a joint activity, α_i, requires that some team member know that the preconditions, $template_{pre}$, for the plan are satisfied. Since preconditions for a particular template may be sensed locally by different team members, belief sharing via information tokens is required to ensure that at least one actor knows all the preconditions. However, if multiple actors get to know the same preconditions, it may happen that the template is instantiated multiple times and the team's effort is wasted on multiple executions of the same plan. Our approach to this problem is described in detail in [29], in this chapter we just provide a brief overview. The approach to avoiding plan duplicates uses two ideas. First, an actor can choose not to instantiate a template (at least for some time), despite knowing the preconditions hold and not knowing of another instantiation. For example, in some cases an actor might wait a random amount of time to see if it hears about another instance before instantiating a plan. Second, once it does instantiate the template into a joint activity it informs each of its neighbours in the associates network. Any actor accepting a role in the joint activity must also inform its neighbors about the joint activity. It turns out that despite only a relatively small percentage of the team knowing about a particular joint activity instance, there is very high probability of at least one team member knowing about both copies of any duplicated team activity. A team member detecting a duplicate plan instantiation is obliged to inform the actors that instantiated the duplicate plans (this information is kept with the information token informing of the initiation of the joint activity) who can then initiate a straightforward deconfliction process.

Task Allocation Once joint activities are instantiated from templates, the individual tasks that make up the joint activity must be assigned to individual actors. Our algorithm for task allocation is extensively described and evaluated in [53]. A task token is created for each task, $t \in Tasks(\alpha)$. The holder of the task token has the exclusive right to execute the task and must either do so or pass the token to a teammate. First, the actor must decide whether it is in the best interests of the team for it to perform the task represented by the token (Alg 2, line 6). A task tokens control information is the minimum capability ($capability(a, task, t)$) an actor must have in order to perform the task, $task$. This threshold is the control component of the token. The token is passed around the team until it is held by an actor with capability above threshold for the task and without other tasks that would prevent it from performing the task. Computing thresholds that maximize expected utility is a key part of this algorithm and is described in [53]. The threshold is calculated once (Alg 2, line 5), when the task arises due to team plan instantiation. A token's threshold therefore reflects the state of the world when it was created. As the world changes, actors will be

able to respond by changing the threshold for newly-created tokens. This allows the team flexibility in dealing with dynamics by always seeking to maximize expected utility based on the most current information available.

Once the threshold is satisfied, the actor must check whether it can perform the task give other responsibilities (Alg. 2, line 9). If it cannot, it must choose a task(s) to reject and pass the respective tokens to a neighbor in the network (Alg. 2, lines 10 and 12). The actor keeps the tasks that maximize the use of its capabilities (performed in the MaxCap function, Alg. 2, line 10), and so acts in a locally optimal manner. Extensions to this algorithm allow efficient handling of constraints between tasks.

Algorithm 2: Task Token

```
(1)   V ← ∅, PV ← ∅
(2)   while true
(3)       token ← getMsg()
(4)       if token.threshold = NULL
(5)           token.threshold ← CalcThreshold(token)
(6)       if token.threshold < Cap(token.value)
(7)           V ← V ∪ token.value

(9)           if Σ_{v∈V} Resources(v) ≥ agent.resources
(10)              out ← V− MaxCap(V)
(11)              foreach v ∈ out
(12)                  PassOn(new token(v))
(13)                  V ← V − out

(15)      else
(16)          PassOn(token) /* threshold < Cap */
```

3.1 Resource Allocation

Efficient teams must be able to assign resources to actors that can make best use of those resources. As described above tasks have both necessary and useful resources. Since there is no global view of which actor is doing which task, the process of allocating resources to tasks is fully distributed. Each shareable, discrete resource is represented by an individual token. As with task tokens, control information on the token is in the form of a threshold. An actor can hold the resource token, and thus have exclusive access to the resource, if it computes its need for the token as being above the threshold. However, unlike task tokens, thresholds for resource tokens are dynamic. When an actor has a resource it slowly increases the threshold (up to some maximum value) and continues checking whether its need for the resource is above that threshold. When the token moves around the team, the threshold is slowly descreases until it is accepted by some agent. The combination of moving the threshold up and

down ensures that whichever actor needs the resource most at a particular point in time gets that resource.

Algorithm 3: ProcessResourceToken

```
(1)   while true
(2)       token ← getMsg()
(3)       if token.threshold < Req(token.resource)
(4)           // Keep the token
(5)           MonitorResourceToken(token)
(6)       else
(7)           Pass(token)
```

Algorithm 4: MonitorResourceToken

```
(1)
      haveToken ← true while haveToken
(2)       sleep()token.threshold ← token.threshold + inc if
          token.threshold > Req(token.resource)
(3)           Pass(token)
(4)           haveToken ← true
```

Sensor Fusion Individual sensors of individual actors may not be sufficient to determine the state of some part of the environment. For example, a single UAV may not have a sufficiently high fidelity sensor suite to independantly determine that an enemy tank is concealed in a forest. However, multiple sensor readings by multiple actors can result in the team having sufficiently high confidence in a determination of the state to take an action. However, in a cooperative *mobile* team an actor will not always have accurate knowledge about where other actors are and hence will not know which team mates might have readings to confirm or refute its own. We encapsulate each sensor reading in an information token and forward the token across the team. Each actor individually performs sensor fusion on the information that it has and creates a new information token with the fused belief when it is able to combine multiple low confidence readings into a single high confidence reading. The key to this algorithm is that despite each token visiting a relatively small number of team members there is high probability that some team member will get to see multiple sensor readings for the same event, if they exist. As with information tokens, the control information for sensor-reading tokens (i.e., information tokens) is the number of additional hops a token should make before assuming it cannot be fused at the current time (i.e., TTL).

4 Synergies Between Algorithms

Efficient token-based coordination depends on how well tokens are routed, i.e., how efficiently they pass from actor to actor to where the are most needed. Since routing decisions are made locally, actors must build local models of the state of the team to make appropriate routing decisions. Notice that *whether* to pass a token on is a function of the control information on the token, but *where* to route a token, if that is the decision, is a function of the local model of state. In this section, we describe an algorithm to maintain the localized decision model by utilizing previously received tokens.

We assume that there is a known relationship between tokens, called *relevance*. We define the relevance relationship between tokens Δ_i and Δ_j as $Rel(\Delta_i, \Delta_j)$. $Rel(\Delta_i, \Delta_j) > 1$ indicates that an agent interested in Δ_i will also be interested in Δ_j, while $Rel(\Delta_i, \Delta_j) < 1$ indicates that an agent interested in Δ_i is unlikely to be interested in Δ_j. If $Rel(\Delta_i, \Delta_j) = 1$ then nothing can be inferred. When an agent receives two tokens for which $Rel(\Delta_i, \Delta_j) > 1$ they are more likely to be usable in concert to obtain a reward for the team. For example, when an actor gets a task token Δ_t and resource token Δ_r representing a necessary resource for the task, $Rel(\Delta_t, \Delta_r) > 1$ and passing them to the same acquaintance is more likely to gain reward for the team than passing them to different acquaintances.

4.1 Updating Decision Model according to Previous Tokens

Each actor maintains a matrix $P[\Delta, a] \rightarrow \mathcal{R}$ that estimates that for each possible token, the probability that each of its associates would be the best to pass that token to. For example, $P[\Delta_k, a] = 0.2$ indicates that the actor estimates that the probability associate a is the best of its associates to pass token Δ_k to is 0.2. Notice that in the implementation we do not actually store the entire matrix but calculate it as needed, but in the following we assume so for clarity.

The update function of agent α's P_α based on an incoming token Δ_j, written as $\delta_P(P_\alpha[\Delta_i, b], \Delta_j)$ leverages Bayes' Rule as following:

$$\forall b \in n(\alpha), \forall \Delta_i, d = first(n(a), \Delta_j.path)$$

$$\delta_P(P_a[\Delta_j, b], \Delta_i) = \begin{cases} P_a[\Delta_j, b] \times Rel(\Delta_i, \Delta_j) & \text{if } \Delta_i \neq \Delta_j, b = d \\ P_\alpha[\Delta_j, b] & \text{if } \Delta_i \neq \Delta_j, b \neq d \\ \varepsilon & \text{if } \Delta_i = \Delta_j, b \in \Delta_j.path \cap n(\alpha) \end{cases} \quad (2)$$

$first$ extracts from the recorded path of the token the acquataince of the actor that earliest had the token. P is then normalized to ensure $\sum_{b \in N(\alpha)} P_\alpha[\Delta_j, b] = 1$. The first case in Eqn. 2 is the most important. The probability that d is the best agent to receive Δ_i is updated according to $Rel(\Delta_i, \Delta_j)$. The second case in the equation changes the probability of sending that token to agents other than the sender in a way that ensures the subsequent normalization has the desired

effect. Finally, the third case encodes the idea that an actor should typically not pass a token back from where it came. Details about how *Rel* is computed to ensure appropriate behavior can be found in [64].

5 Human-in-the-Loop

The token-based process described above works effectively at controlling large teams. However, for real-world teams it is essential to have a human-in-the-loop, controlling the behavior of the team. The need for such control stems from two key reasons. First, the heuristics used to coordinate the team will not always work effectively and sometimes human "common sense" will be required to ensure appropriate behavior. Second, the human may have preferences for tradeoffs given the current situation, e.g., a willingness to trade off the quality of task allocation provided bandwidth is reduced. These two rationales for human control imply an ability for control at both a high and low level and over a wide range of aspects of behavior. Fortunately, the homogeneity of the token-based algorithms allows an effective and general control layer to be built on top of the control flows providing powerful control for a human user (or users.) The effect is to allow meta-reasoning over the token-based coordination. The specific approach we have developed has two components, one for high level control and another for more detailed control.

5.1 High-Level Control

The high level control component allows the user to tradeoff high level performance measures such as the message bandwidth versus performance task allocation versus resource allocation. To do this we need a model of the interaction between the environment algorithm configuration and performance. However, the relationship turns out to be very complex, denying straightforward means of modeling it. Moreover, non-determining leads to a relatively high standard deviation in performance. To represent the highly non-linear relationship between the environment, configuration and performance of the team, we used multilayer feed-forward neural networks, capable of representing any arbitrary function[38]. With inspirations from the idea of dynamic rearrangement [13], we use the concept, called *Dynamic Neural Networks* [39][40], which allows all internal nodes in the network to act stochastically and independently even though all external input data remain unchanged.

We trained the multilayer feed-forward neural network using genetic algorithms because of the high standard deviation of the function being modelled. Moreover, in genetic algorithms, the unit of adaptation is not an individual agent, but a population of agents, which is excellent for dealing with very huge and noisy training data set. The fitness function was the average of square error between target output and actual output as follows:

$$\sum_{d \in D} \sum_{p \in P} (O^d_{p,t} - O^d_{p,a})^2 / sizeof(D).$$

Where D is the set of training data ($d \in D$), P is the set of system performance measures ($p \in P$), $O^d_{p,t}$ is the target output of the p th performance parameter of the data entry d, and $O^d_{p,a}$ is the actual output of the p th performance parameter of the data entry d. The genetic algorithm training function attempts to minimize this function. The learning process converged to 20 percent error quickly and slowly converged to 15 percent error after that. Future work will look at making more accurate models.

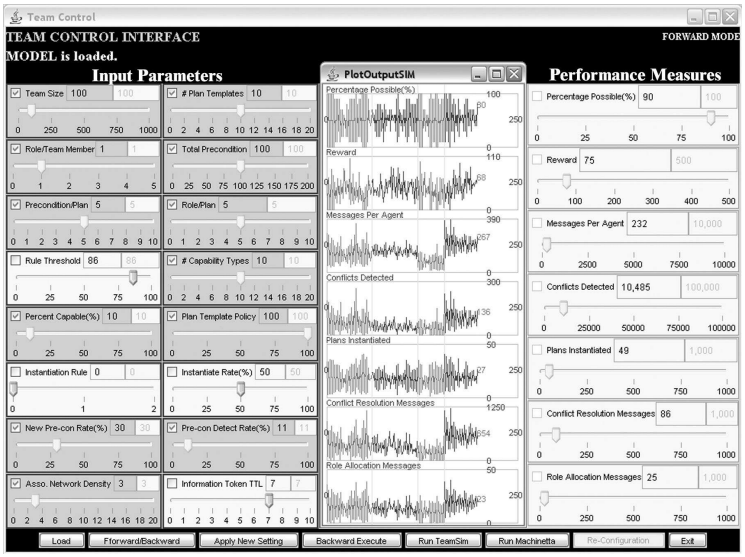

Fig. 2. The team control interface for online and offline control. Input parameters are shown on the left side, performance measures are shown on the right side. Check boxes for performance measures are used to specify the constraints for finding configurations.

Team Control Interface A user interface, shown in Figure 2 was developed for working with the dynamic neural network model. There are two key interaction modes: *input-to-output* where the model shows the expected performance of a particular setup; and *output-to-input* where the model shows the optimal configuration to achieve a specified performance profile. In input-to-output mode the interface simply provide inputs to the neural network and displays the output, but the output-to-input mode is more complex.

Output-to-Input Mode Using the team neural network in "reverse", the interface allows a user to change output parameters and receive a configuration that best meets some specific performance constraints both in input and output. The user

specifies which performance features to constrain and what values they should have. In order to find input parameters that meet output requirements, the interface performs a search over the changeable configuration parameters to find a configuration that gives the required performance tradeoffs. Notice that this usage of the neural network allows various coordination algorithms to be traded off against each other automatically. For example, if the user requests a descrease in bandwidth usage, the neural network can determine which algorithms to limit that bandwidth usage of to have the least impact on overall performance.

The user interface can be connected to an executing team allowing the user to monitor system performance and to change configuration during execution. Special data collection tokens sample the team to determine current performance measures and the state of the environment. When the user specifies a new performance tradeoffs or the environment changes, the neural network determines the best configuration for meeting the users needs and sends information tokens to all actors to get the new configuration initiated.

5.2 Addressing Specific Problems

Technically, because tokens completely encapsulate pieces of the overall coordination, it is feasible to examine individual tokens to determine whether that particular aspect of the coordination is working correctly. If not, or if the user has some particular preference for how that particular detaileds aspect should work, then the individual token can be extracted and modified (or its task taken over by a human.) Because of the independance of tokens, it is possible to extract any single token without effecting the behavior of the others. However, it is infeasible to have a *human* monitor all tokens and determine which are not performing to their satisfaction. Our approach is to instead have a model of expected token behavior and bring tokens to the attention of a human when the tokens behavior deviates from this model. Conceptually, this process corresponds to identifying details of coordination that may be problematic and bringing them to the attention of a human.

In practice, autonomously identifying coordination problems that might be brought to the attention of a human expert is imprecise. Rather than reliably finding poor coordination, the meta-reasoning must find *potentially* poor coordination and let the humans determine the actually poor coordination. (Elsewhere we describe the techniques that are used to ensure that this does not lead to the humans being overloaded[51].) Notice that while we allow humans to attempt to rectify problems with agent coordination, it is currently an open question whether humans can actually make better coordination decisions than the agents. For example, when a task token travels to many actor repeatedly, it may be that no actor has the required capability for the task or that the task is overloaded. A human might cancel the task or find an alternative way of acheiving the same goal.

6 Implementation

To evaluate the token-based approach we have developed both an abstracted simulator and a fully distributed implementation called Machinetta. The abstract simulator, called *TeamSim* represents tasks, information and resources as simple objects and uses simple queues for messages. It allows very rapid prototyping of algorithms and extensive testing to be performed.

Machinetta is an approach to building generic coordination software based on the concept of a *proxy*[22, 43]. Each team member is given its own proxy which encapsulates generic coordination reasoning. Plan templates are specified in XML and given to all the proxies. The proxy interacts with its team member via an abstracted interface that depends on the type of actor, e.g., for a robot it might be a simple socket while for a human it may be a sophisticated GUI. The proxies coordination together, using the token-based algorithms described above to implement the coordination. Machinetta proxies have been demonstrated to perform efficient, effective coordination with up to 500 distributed team members. They have been tested in several distinct domains and will be used as a part of a U.S. Airforce flight test in September 2005.

7 Results

In this section we present results of the individual token algorithms, the synergistic use of the algorithms and the human in the loop control. For results utilizing Machinetta refer to [50, 52, 49, 54]. Note that these results have for the most part been previously published elsewhere but are collected here to present a cohesive picture of the approach.

7.1 Task Allocation

To test the token based task allocation, we developed a simple simulator where actors are randomly given capabilities, independant of information or resources, for each of 5 types of task, with some percentage of actors being given zero capability for a type of task. For each time step that the agent has the task, the team receives ongoing reward based on the agent's capability. Message passing is simulated as perfect (lossless) communication that takes one time step. As the simulation progresses, new tasks arise spontaneously and the corresponding tokens are distributed randomly. The new tasks appear at the same rate that old tasks disappear, thus keeping the total number of tasks constant. This allows a single, fixed threshold for all tasks to be used throughout the experiment. Each data point represents the average from 20 runs.

Figure 3 shows the performance of the algorithm against two competing approaches. The first is DSA, which is shown to outperform other approximate distributed constraint optimization algorithms for problems like task assignment [32, 16]; we choose optimal parameters for DSA [65]. As a baseline we also compare against a centralized algorithm that uses a "greedy" assignment[5]. Results

are shown using two different thresholds for the task tokens, T=0.0 and T=0.5. Figure 3(a) shows the relative performance of each algorithm as the number of agents is increased. The experiment used 2000 tasks over 1000 time steps. The y-axis shows the total reward, while the x-axis shows the number of agents. Not surprisingly, the centralized algorithm performs best but not dramatically better than the token based approach. The token based approach performs significantly better with a threshold of 0.5 than with no threshold. The real key to the comparison, however, is the amount of communication used, as shown in Figure 3(b). Notice that the y-axis is a logarithmic scale; thus the token based approach uses approximately four orders of magnitude fewer messages than the greedy algorithm and six orders of magnitude fewer messages than DSA. The token-based approach performs better than DSA despite using far less communication and only marginally worse than a centralized approach, despite using only a tiny fraction of the number of messages.

7.2 Information Sharing Results

To evaluate the information sharing algorithms, we arranged agents into a network and randomly picked one agent as the source of a piece of information i and another as a sink (i.e., for the sink agent $U(i)$ is very large). The sink agent sent out 30 information tokens (with $TTL = 150$) with information with a high Rel to i. Then the source agent sent out i and we measured how long it takes to get to the sink agent. In Figure 4(b) we show a frequency distribution of the time taken for a network with 8000 agents. While a big percentage of messages arrive efficiently to the sink, a small percentage get "lost" on the network, illustrating the problem with a probabilistic approach. However, despite some messages taking a long time to arrive, they all eventually did and faster than if moved at random. We also looked in detail at exactly how many messages must be propogated around the network to make the routing efficient (Figure 5). Again using 8000 agents we varied the number of messages the sink agent would send before the source agent sent i onto the network. Notice that only a few messages are required to dramatically affect the average message delivery time.

7.3 Sensor Fusion Results

To evaluate the sensor fusion approach we use a random network of 100 nodes. Nodes are randomly chosen as the source of relevant sensor readings. Information tokens propogate the sensor readings through the network and we measure the probability of getting a successful fusion given a fixed TTL ("hops" on x-axis of graph). Figure 6 shows two cases, one where all three sensor readings must be known to a single actor for fusion to be successful (labeled 3/3) and one where three of five readings must be known to a single actor for successful fusion (labeled 3/5). Notice that a relatively small TTL is required to have high probability of successful fusion.

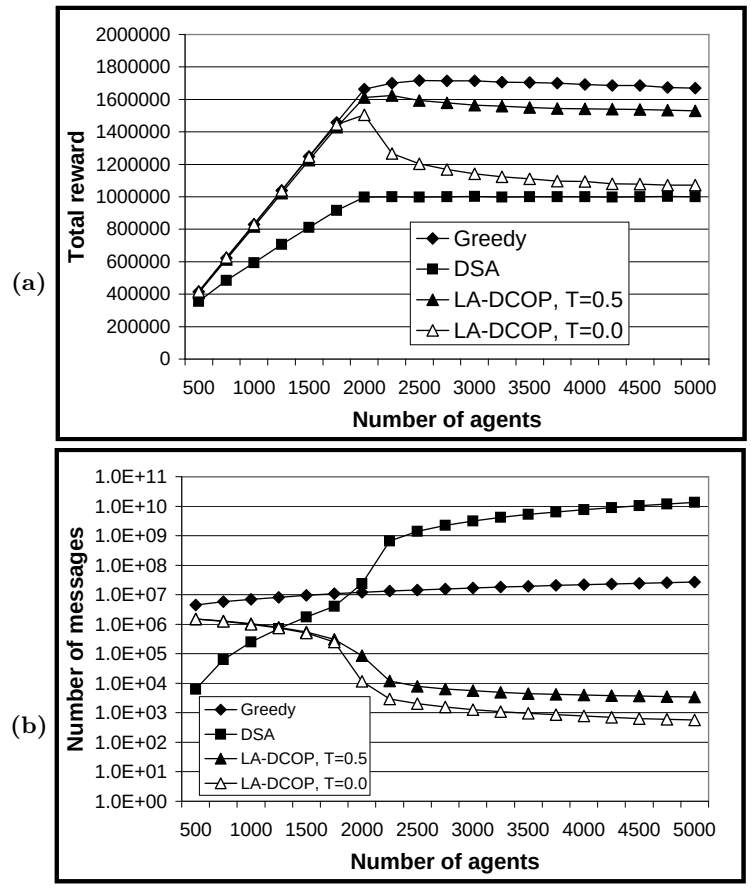

Fig. 3. (a) comparing the reward versus the number of agents. (b) the number of messages sent versus the number of agents

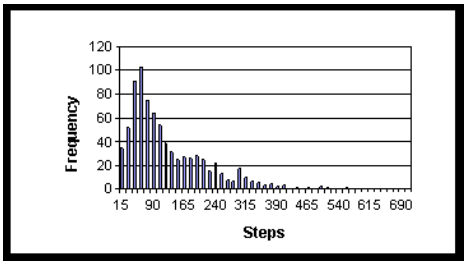

Fig. 4. Distribution of number of steps required

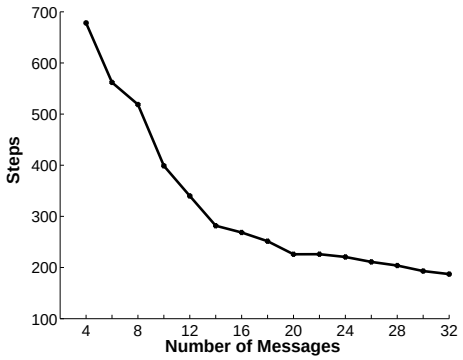

Fig. 5. Association between number of relative messages and delivery time

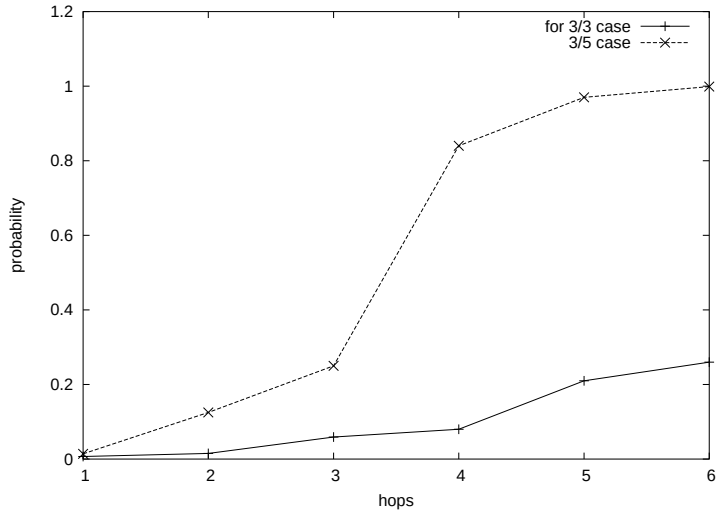

Fig. 6. The probability of a successful fusion (y-axis) by at least one actor given a specific TTL (x-axis) when three of five or three of three readings must be known by a single actor to perform fusion.

7.4 Token-Based Algorithms Working Together

To evaluate the synergies between algorithms, due to the use of the P model, we configured TeamSim to simulate a group of 400 distributed UAVs searching a hostile area. Simulating automatic target detection rates 200 pieces of information, e.g., SAM sites, were randomly "sensed" by UAVs and passed around the team. Fifty plan templates, each with four independent preconditions were used on each of 100 trials. Each plan template had four tasks to be performed. Thresholds for the tasks tokens were set such that UAVs needed to be near the target or reconnasaince site to accept the task. Shared resources were airspace that the UAVs needed to fly through to complete their tasks. One resource token was created for each "voxel" of airspace. When all four tasks in a plan were completed, the team recieved a reward of 10. A maximum reward of 500 units (10 units x 50 plans) was possible.

Five variations of the integrated algorithm were compared. The most integrated algorithm used all types of plan tokens to update P. The least integrated algorithm moved tokens randomly to associates when it was decided to move a token. Three intermediate variations of the algorithm used only one type of token, resource, role or information, tokens to update the local routing model, P. Figure 7 shows the reward received by the team on the y-axis and the time on the x-axis. The Figure shows that the team recieved more reward, faster when using the integrated algorithm. Moreover, Figure 8 shows that less messages (on the y-axis) were required to get a fixed amout of reward (on the x-axis) for the integrated approach. Both Figures show that using any type of tokens to build a routing model is better than using no routing model at all. Finally, Figure 9 shows that the algorithm scales well with increased team size. In this case, the y-axis shows the average number of messages per agent required to achieve a certain amount of reward. Notice, there is some indication that the average number of messages goes down as the team gets bigger, but more work is required to determine under what conditions this holds and what the key reasons for it are.

7.5 Meta-Reasoning

We have evaluated both the low and high level aspects of the human-in-the-loop control of the large teams.

High Level Control Using TeamSim we were able to verify that the user was able to reconfigure a team online and get required changes in performance tradeoffs. The interface is connected directly with TeamSim, so that users can set team configurations and monitor team performance measures online. The user configures the team at the start of the mission. When performance changes are requested the offline features of the team performance model areneural network is used to find suitable reconfigurations. The team control interface and reconfiguration assistance were evaluated over 10 scenarios. Scenarios were selected to provide situations that would require users to reconfigure their team in order to meet performance targets. For example, in a mission involving a very large team

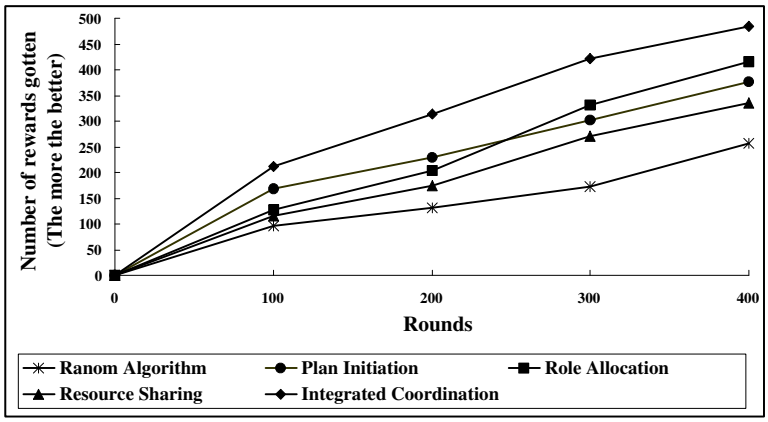

Fig. 7. Team can get more rewards in the coordination of token-based algorithm

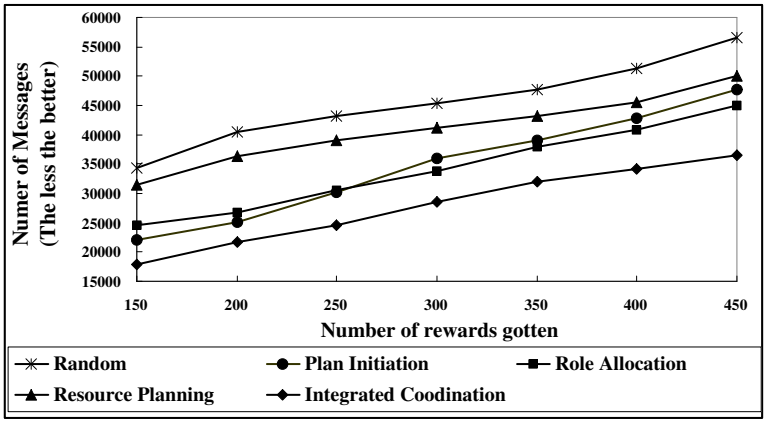

Fig. 8. Team can cost less messages in the coordination of token-based algorithm

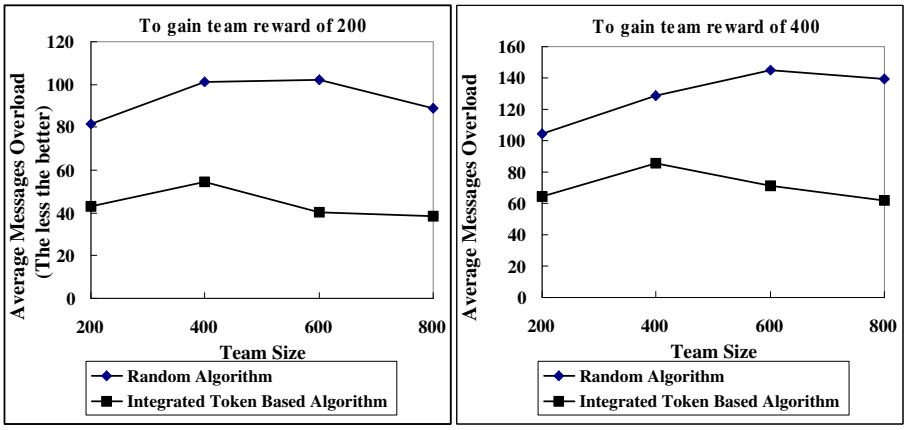

Fig. 9. Efficiency for coordinating different size of teams

of 300 agents the user might be requested at some point in the mission to reduce the number of messages per agent or increase the number of plans instantiated. Performance measures are recorded throughout the execution. The data presented here represents 4 hours of runtime with a user in the loop. At step 1, the initial team configuration is set. At step 2, the user is asked to increase level of rewards obtained by the team disregarding other performance measures. Using the output-to-input feature of the team performance model the user finds a new coordination configuration that increases reward performance and reconfigures the team. At step 3 network communication bandwidth is reduced limiting the time-to-live for information tokens to 2 hops requiring another team reconfiguration to lessen the degradation in performance. At step 4, the user is again asked to reconfigure to improve reward performance. Results for six of the performance measures are shown in Figure 10. The bold lines show average values for the configured system while the lighter lines indicate the values predicted by the output-to-input model. The jagged lines show the moment to moment variation in the actual performance measures. Despite the high variability of team performance measures the model qualitatively predicts the effects of reconfiguration on average performance values across all six measures.

Low Level Control To remove the need for many hours of human input, the interfaces for manipulating individual tokens were augmented with code that made decisions as if they were made by the human. These "human" decisions were made between five seconds and two minutes after control was transferred to the human. The experiments involved a team of 80 WASMs operating in a large environment. The primary task of the team was to protect a manned aircraft by finding and destroying surface-to-air missile sites spread around the environment. Half the team spread out across the environment searching for targets while the

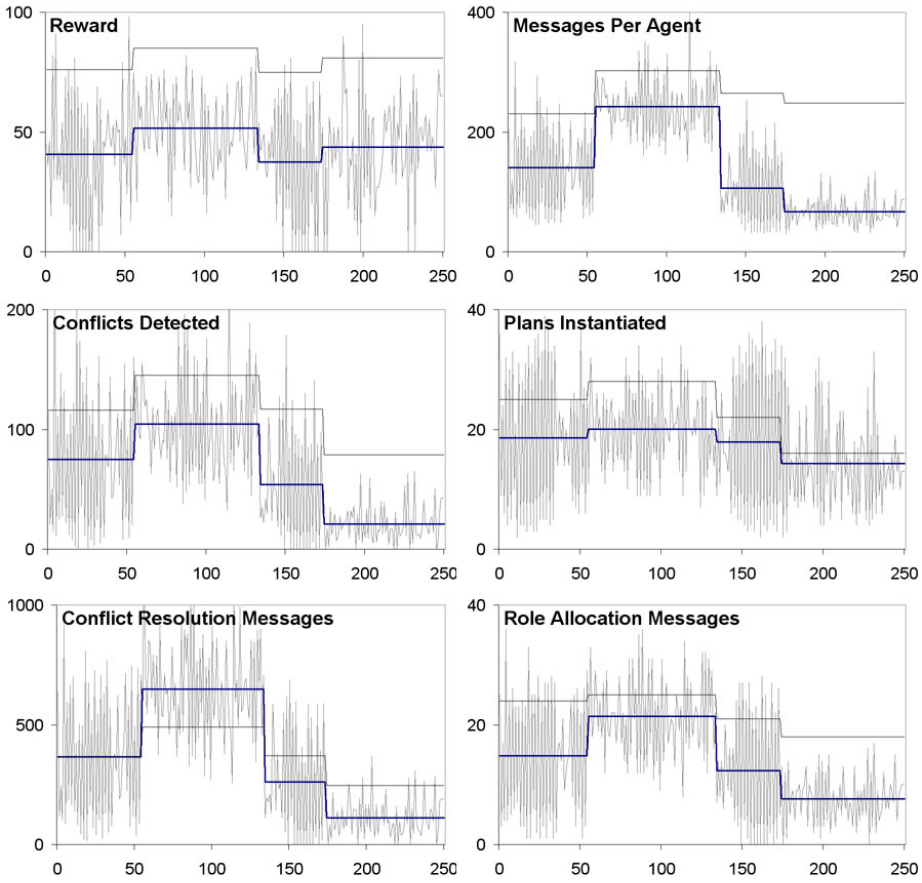

Fig. 10. Six performance measures recorded from TeamSim are ploted during the mission with 3 times of reconfiguration. Thick lines show the average values of actual performance measures of each configuration setting. Thin lines are the predicted values by the user interface.

other half stayed near the manned aircraft destroying surface-to-air sites as they were found near the aircraft. Plans were simple, requiring a single WASM to hit each found target. If a target was not hit within three minutes of being found, this was considered abnormal plan execution and meta-reasoning would be invoked. Meta-reasoning was also invoked when a WASM was not allocated to hit any target for five minutes. These times are low, but reasonable since the simulation ran at approximately four times real-time. Finally, meta-reasoning was invoked when no WASM was available to hit a found target. Two human commanders were available to make meta-reasoning decisions (although, as discussed above there were not "real" human commanders).

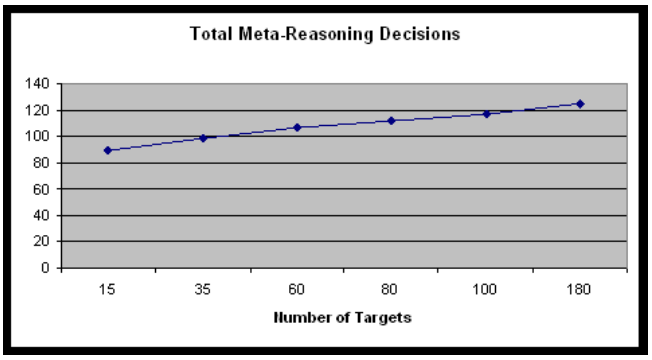

Fig. 11. The number of meta-reasoning decisions to be made as the number of targets in the environment increases.

Six different scenarios were used, each differing the number of surface-to-air missile sites. Each configuration was run ten times, thus the results below represent around 30 hours of simulation time (120 hours of real-time). As the number of missile sites increases, the team will have more to do with the same number of WASMs, thus we expected more meta-reasoning decisions.

Figure 11 shows that the total number of meta-reasoning decisions does increase with the number of targets. Over the course of a simulation, there are around 100 meta-reasoning decisions or about one per agent. However, as Figure 12 shows, only about 20% of these get transferred to a human. The large number of decisions that are made autonomously is primarily because humans are not available to make those decisions. This suggests work may need to be done to prioritize decisions for a user, to prevent high priority decisions being left to an agent, while the user is busy with low priority decisions. However, an appropriate solution is not obvious, since new decisions arrive asynchronously and it will likely not be appropriate to continually change the list of decisions the human is working on. Finally, notice in Figure 13 that a large percentage of the meta-decisions are to potentially cancel long running plans. The large number of

such decisions illustrates a need to carefully tune the meta-reasoning heuristics in order to avoid overloading the system with superfluous decisions. However, in this specific case, the problem of deciding whether to cancel a long running plan was the most appropriate for the human, hence the large percentage of such decisions for the human is reasonable.

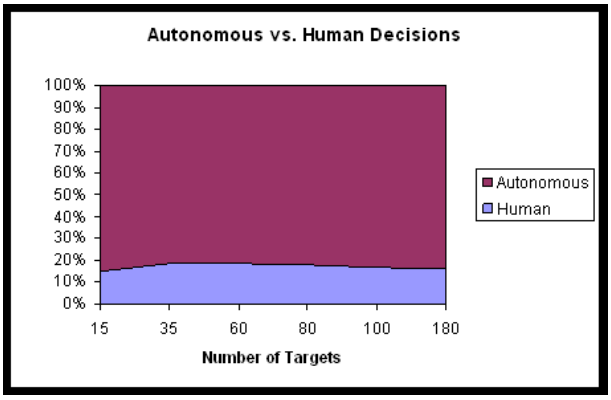

Fig. 12. The percentage of decisions transferred to humans versus the percentage made autonomously.

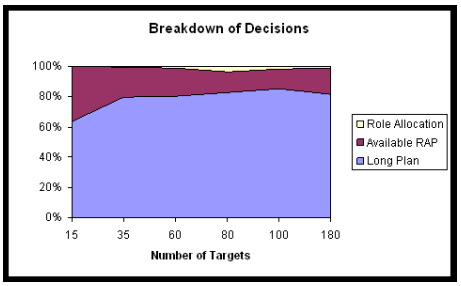

Fig. 13. Ratios of different types of meta-reasoning decisions presented to the user.

8 Related Work

Coordination of distributed entities is an extensively studied problem[9, 8, 24, 28, 55]. A key design decision is how the control is distributed among the group

members. Solutions range from completely centralized[14], to hierarchical[11, 20] to completely decentralized[60]. 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[3]. Creating distributed groups of cooperative autonomous agents and robots that must cooperate in dynamic and hostile environments is a huge challenge that has attracted much attention from the research community[25, 27]. 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[6, 10, 22, 47].

Tidhar [57] 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 [57] — team selection matches the "skills" required for executing a team plan against agents that have those skills. Jennings's GRATE* [22] 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[6]. COLLAGEN [46] 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.

8.1 Small Worlds Networks

Research on social networks began in physics[59, 2]. Gaston [17] investigate the team formation on type of social network structures can dramatically affect team abilities to complete cooperative tasks. In particular, using a scale-free network structure for agent team will facilitate team formation by balancing between the number of skill-constrained paths available in the agent organization with the effects of potential blocking. Pujol [41] compared the merits of small world network and scale free network in the application of emergent coordination.

Task Allocation Numerous task allocation algorithms have been proposed, although most do not consider costs (find only satisfying allocations) or scale very poorly with team size, or both. Symbolic matching techniques[57, 36] ignore costs completely which can have disastrous effects on team performance. Combinatorial auctions[21] are one approach that seek to minimize costs, but are impractical for very large teams due to the exponential number of possible bids and bottlenecks formed by centralized auctioneers. Forward-looking, decision-theoretic models[33] can exploit task decomposition to optimally allocate and reallocate tasks, but also do not scale to very large teams due to the exponential size of the search space.

Complete distributed constraint optimization algorithms[32, 31] can find optimal solutions but require impractically long running times and unacceptably high amounts of communication. Some incomplete distributed constraint optimization algorithms[65] can be scaled to large teams, but these may also suffer from a high amount of communication, and has been outperformed by our approach in previous evaluations[34].

Swarm-based approaches[48, 1, 4] provide a distributed, highly scalable way to perform task allocation and reallocation. Interestingly enough, these algorithms also rely on threshold-based computations. However, swarm algorithms rely directly on locally sensed stimuli to adjust thresholds and make decisions, while under our approach actors may use arbitrary information obtained locally or from other actors. This additional level of flexibility can be leveraged for better performance through synergistic interactions with the other algorithms presented here.

Human Control The approach of using sensitivity analysis of multilayer neural networks to provide inverse relationship from output to input have been applied in several areas. Especially, Ming Lu et al. [30] demonstrated a simple algorithm for using a sensitivity analysis of neural networks and X. Zeng et al. [62] provide theoretical results.

Peter Eggenberger et al. [13] investigated and introduced the idea of dynamic rearrangement of biological nervous systems. Their approach allows neural networks to have an additional mechanism to dynamically change their synaptic weight modulations and neuronal states during execution. [15] presented another idea of dynamic network that dynamically modifying network structure. The algorithms start with zero or small number of hidden nodes and later the network change its structure by the number of hidden nodes to find the structure that fit well with the target system. A. Parlos et al. [37] proposed a hybrid feedforward/feedback neural network for using to identify nonlinear dynamic systems. Dynamic backpropagation learning is demonstrated as the dynamic learning algorithm.

9 Conclusions and Future Work

In this chapter, we have presented a novel approach to coordination based on the idea of tokens. We have showed how such algorithms can be very effective for scalable coordination, particularly when they are combined into an integrated algorithm. The homogeneity of the token-based approach allowed us to build a general meta-reasoning layer over the top of the flows of tokens. This meta-reasoning layer gives a user powerful tools for ensuring that the team fulfills their requirements. Future work will examine how to inject fault tolerance into these algorithms and how the precise details of the associates network affect behavior.

Acknowledgements

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

References

1. William Agassounon and Alcherio Martinoli. Efficiency and robustness of threshold-based distributed allocation algorithms in multiagent systems. In *Proceedings of AAMAS'02*, 2002.
2. Albert-Laszla Barabasi and Eric Bonabeau. Scale free networks. *Scientific American*, pages 60–69, May 2003.
3. Johanna Bryson. Hierarchy and sequence vs. full parallelism in action selection. In *Intelligent Virtual Agents 2*, pages 113–125, 1999.
4. M. Campos, E. Bonabeau, G. Therauluz, and J.-L. Deneubourg. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 2001.
5. C. Castelpietra, L. Iocchi, D. Nardi, M. Piaggio, A. Scalzo, and A. Sgorbissa. Coordination among heterogenous robotic soccer players. In *Proceedings of IROS'02*, 2002.
6. 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.
7. Richard Clark. *Uninhabited Combat Air Vehicles: Airpower by the people, for the people but not with the people.* Air University Press, 2000.
8. 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.
9. Philip R. Cohen and Hector J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.
10. 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.
11. 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.
12. Defense Science Board. Defense science board study on unmanned aerial vehicles and uninhabited combat aerial vehicles. Technical report, Office of the Under Secretary of Defense for Acquisition, Technology and Logistics, 2004.
13. P. Eggenberger, A. Ishiguro, S. Tokura, T. Kondo, and Y. Uchikawa. Toward seamless transfer from simulated to real worlds: A dynamically-rearranging neural network approach. In *Proceeding of 1999 the Eighth European Workshop in Learning Robot (EWLR-8)*, pages 44–60, 1999.
14. 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.
15. S. E. Fahlman and C. Lebiere. The Cascade-Correlation Learning Architecture. In Touretzky (ed.), editor, *Advances in Neural Information Processing Systems 2*. Morgan-Kaufmann.

16. Stephen Fitzpatrick and Lambert Meertens. *Stochastic Algorithms: Foundations and Applications, Proceedings SAGA 2001*, volume LNCS 2264, chapter An Experimental Assessment of a Stochastic, Anytime, Decentralized, Soft Colourer for Sparse Graphs, pages 49–64. Springer-Verlag, 2001.
17. M. Gaston and M. desJardins. The communicative multiagent team decision problem: analyzing teamwork theories and models. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference*, 2005.
18. Dani Goldberg, Vincent Cicirello, M Bernardine Dias, Reid Simmons, Stephen Smith, and Anthony (Tony) Stentz. Market-based multi-robot planning in a distributed layered architecture. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, volume 2, pages 27–38. Kluwer Academic Publishers, 2003.
19. Barbara Grosz and Sarit Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996".
20. 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.
21. L. Hunsberger and B. Grosz. A combinatorial auction for collaborative planning, 2000.
22. N. Jennings. The archon systems and its applications. Project Report, 1995.
23. N. Jennings, E. Mamdani, I Laresgoiti, J. Perez, and J. Corera. GRATE: A general framework for cooperative problem solving. *Intelligent Systems Engineering*, 1(2), 1992.
24. David Kinny. The distributed multi-agent reasoning system architecture and language specification. Technical report, Australian Artificial intelligence institute, Melbourne, Australia, 1993.
25. 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.
26. 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.
27. 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.
28. 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.
29. E. Liao, P. Scerri, and K. Sycara. A framework for very large teams. In *AAMAS'04 Workshop on Coalitions and Teams*, 2004.
30. Ming Lu, S. M. AbouRizk, and U. H. Hermann. Sensitivity analysis of neural networks in spool fabrication productivity studies. *Journal of Computing in Civil Engineering*, 15(4):299–308, 2001.
31. R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS'04*, 2004.

32. Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of Autonomous Agents and Multi-Agent Systems*, 2003.
33. R. Nair, M. Tambe, and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of the second International Joint conference on agents and multiagent systems (AAMAS)*, 2003.
34. Steven Okamoto. Dcop in la: Relaxed. Master's thesis, University of Southern California, 2003.
35. Committee on Visionary Manufacturing Challenges. Visionary manufacturing challenges for 2020. National Research Council.
36. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web service capabilities. In *Proceedings of the First International Semantic Web Conference*, 2002.
37. Chong K. T. Parlos, A. G. and A. F. Atiya. Application of the Recurrent Multilayer Perceptron in Modeling Complex Process Dynamics . In *IEEE Transactions on Neural Networks*, pages 255–266, 1994.
38. Leonid I. Perlovsky. *Neural Networks and Intellect: Using Model-Based Concepts*. Oxford University Press, 2001.
39. J. Polvichai and P. Khosla. An evolutionary behavior programming system with dynamic networks for mobile robots in dynamic environments. In *Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and System*, volume 1, pages 978–983, 2002.
40. J. Polvichai and P. Khosla. Applying dynamic networks and staged evolution for soccer robots. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and System*, volume 3, pages 3016–3021, 2003.
41. J. Pujol and R. Sanguesa. Emergence of coordination in scale-free networks. In *Web Intelligence and Agent Systems 131-138*, 2003.
42. 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.
43. 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.
44. 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.
45. C. Reynolds. Authoring autonomous characters. Invited Talk, Distinguished Lecture Series, Georgia Institute of Technology, Fall 1995.
46. C. Rich and C. Sidner. COLLAGEN: When agents collaborate with people. In *Proceedings of the International Conference on Autonomous Agents (Agents'97)"*, 1997.
47. 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.
48. Pedro Sander, Denis Peleshchuk, and Barabara Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of AAMAS'02*, 2002.
49. 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.

50. 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.
51. P. Scerri, K. Sycara, and M Tambe. Adjustable autonomy in the context of coordination. In *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, 2004. Invited Paper.
52. P. Scerri, Yang. Xu, E. Liao, J. Lai, and K. Sycara. Scaling teamwork to very large teams. In *Proceedings of AAMAS'04*, 2004.
53. Paul Scerri, Alessandro Farinelli, Steven Okamoto, and Milind Tambe. Allocating tasks in extreme teams. In *AAMAS'05*, 2005.
54. N. Schurr, J. Marecki, J.P. Lewis, M. Tambe, and P.Scerri. The DEFACTO system: Training tool for incident commanders. In *IAAI'05*, 2005.
55. Munindar Singh. Developing formal specifications to coordinate hetrogeneous agents. In *Proceedings of third international conference on multiagent systems*, pages 261–268, 1998.
56. Milind Tambe. Agent architectures for flexible, practical teamwork. *National Conference on AI (AAAI97)*, pages 22–28, 1997.
57. G. Tidhar, A.S. Rao, and E.A. Sonenberg. Guided team selection. In *Proceedings of the Second International Conference on Multi-Agent Systems*, 1996.
58. T. Wagner, J. Phelps, V. Guralnik, and Ryan VanRiper. COORDINATORS: Coordination managers for first responders. In *AAMAS'04*, 2004.
59. Duncan Watts and Steven Strogatz. Collective dynamics of small world networks. *Nature*, 393:440–442, 1998.
60. 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.
61. Michael Wooldridge and Nicholas Jennings. *Distributed Software agents and applications*, chapter Towards a theory of cooperative problem solving, pages 40–53. Springer-Verlag, 1994.
62. D.S.Yeung Xiaoqin Zeng. Sensitivity analysis of multilayer perceptron to input and weight perturbations. *IEEE Transactions on Neural Networks*, 12(6):1358–1366, 2001.
63. 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.
64. Y. Xu, P. Scerri, B. Yu, S. Okamoto, M. Lewis, and K. Sycara. An integrated token-based algorithm for scalable coordination. In *AAMAS'05*, 2005.
65. W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proceedings of AAAI'02*, 2002.