Allocating Tasks in Extreme Teams

Paul Scerri*, Alessandro Farinelli+, Steven Okamoto* and Milind Tambe#

* Carnegie Mellon University
+ University of Rome
# University of Southern California

pscerri@cs.cmu.edu, Alessandro.Farinelli@dis.uniroma1.it,
sokamoto@cs.cmu.edu, tambe@usc.edu

ABSTRACT

Extreme teams, large-scale agent teams operating in dynamic environments, are on the horizon. Such environments are problematic for current task allocation algorithms due to the lack of locality in agent interactions. We propose a novel distributed task allocation algorithm for extreme teams, called LA-DCOP, that incorporates three key ideas. First, LA-DCOP's task allocation is based on a dynamically computed *minimum capability threshold* which uses approximate knowledge of overall task load. Second, LA-DCOP uses tokens to represent tasks and further minimize communication. Third, it creates *potential tokens* to deal with inter-task constraints of simultaneous execution. We show that LA-DCOP convincingly outperforms competing distributed task allocation algorithms while using orders of magnitude fewer messages, allowing a dramatic scale-up in extreme teams, upto a fully distributed, proxy-based team of 200 agents. Varying threshold are seen as a key to outperforming competing distributed algorithms in the domain of simulated disaster rescue.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms

Keywords

Task Allocation, Distributed Constraint Optimization

1. INTRODUCTION

Distributed task allocation is a fundamental research challenge in multiagent systems, with recent results reporting significant progress in task allocation in teams[13, 7, 22, 19, 9]. However, a significant large class of practical applications — that we call *extreme teams* — has emerged, imposing new requirements for task allocation in teams. Extreme teams include mobile sensor or UAV teams[6], robot teams for Mars colonies[5], disaster rescue scenarios, as well as large-scale future integrated manufacturing and service organizations (e.g., hospitals)[15]. Extreme teams require team members, each with limited resources, to act in real-time dynamic environments. More importantly, team members possess overlapping functionality, but differing capabilities to perform different tasks. For instance, in disaster rescue simulations, different fire fighters and paramedics comprise an extreme team; and while fire fighters and paramedics have overlapping functionality to rescue civilians, for a specific rescue task, one set of paramedics may have a higher due to their specific training and current context.

The problem of task allocation in teams is one of optimally assigning tasks in a team plan to agents to maximize overall team utility[12, 22]. Extreme teams emphasize four key constraints on task allocation: (i) domain dynamics may cause tasks to appear and disappear; (ii) agents may perform multiple tasks within resource limits; (iii) many agents have overlapping functionality to perform each task, but with differing levels of capability; and (iv) inter-task constraints (such as simultaneous execution requirements) may be present. This task allocation challenge in extreme teams will be referred to as E-GAP, as it subsumes the generalized assignment problem (GAP), which is NP-complete[21].

The first two constraints in E-GAP above (dynamics and multiple tasks) make approximations necessary, since it is extremely difficult to obtain optimal solutions in a timely fashion. The remaining two constraints emphasize *lack of locality* in agent interactions, e.g., due to overlapping agent functionality, in assigning a specific task, an agent must potentially consider all other agents (and not a small subset). However, in practical extreme team domains agents will frequently possess reasonable estimates of the overall team capabilities or the situation. For example, fire fighter team members may know the number of fire trucks to an order of magnitude, and have (only) a probability distribution on the locations of fires. This imperfect team knowledge is a key property of extreme teams, and provides a valuable way to restrict the search space to good (if suboptimal) solutions.

This paper builds on Distributed Constraint Optimization (DCOP)[11, 4] for task allocation, as DCOP offers the key advantages of distributedness, presence of fast/approximate algorithms and a rich representational language which can consider costs/utilities of tasks. Despite these advantages, previous DCOP approaches to task allocation suffer from three key weaknesses. First, DCOP algorithms are unable to use imperfect team knowledge to efficiently and effectively allocate tasks. Second, constraints exist between any team

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.
Copyright 2005 ACM 1-59593-094-9/05/0007 ...$5.00.

members with overlapping functionality, resulting in dense constraint graphs that dramatically increase communication within DCOP algorithms; even approximate DCOP algorithms. Third, DCOP algorithms handle interdependencies between tasks (such as requirements of simultaneous execution) very inefficiently, as these are, in effect, non-binary constraints.

We propose a novel DCOP algorithm called LA-DCOP (Low-communication Approximate DCOP) to meet the requirements of E-GAP. LA-DCOP uses a representation where agents are variables that can take on values from a common pool, i.e., the pool of tasks to be assigned. The mechanism for allocating tasks to agents encapsulates three novel ideas. First, LA-DCOP improves efficiency by not solving for an exact optimal reward; instead, it focuses on maximizing the team's expected total reward, given available probabilistic information, by computing a minimum capability *threshold* for each task. This threshold is dynamically computed and altered based on dynamic knowledge of the team and task environment. Second, *token*-based access to values reduces the communication overhead due to constraint graph denseness by allowing at most one agent to perform each task at any given time. Third, to deal with groups of interdependent tasks, we introduce the idea of allowing values to be represented by *potential tokens*. By accepting a potential token, an agent confirms that it will perform the task once the interdependencies have been worked out. In the meantime, the agent can perform other tasks.

We have extensively, empirically evaluated the LA-DCOP algorithm using a mixture of high and low fidelity simulation environments. Experiments on a simplified testbed illustrate four key points. First, the key features of the algorithm, including thresholds and potential tokens, significantly improve its performance. Second, when compared to other approximate DCOP algorithms, LA-DCOP finds better task allocations, while using up to six orders of magnitude fewer messages. Third, we illustrate that the algorithm performs well on two realistic domains, by embedding it in teamwork proxies. LA-DCOP has allowed a dramatic scale up in feasible task allocation for proxy teams, from 20 agents to 200 agents. We also illustrate effective task allocation in a large-scale disaster rescue application and illustrate LA-DCOP significantly outperforming its competitors.

2. PROBLEM STATEMENT

A static task allocation problem is an example of a GAP instance with a set $\Theta = \{\theta_1, \ldots, \theta_m\}$ of tasks to be performed and a set $E = \{e_1, \ldots, e_n\}$ of team members to perform them[21]. Each team member $e_i \in E$ has a capability to perform each task $\theta_j \in \Theta$, and a limited amount of resources with which to perform all of its tasks. Capability reflects the quality of the output or the speed of task performance or other factors affecting output, and is a measurement of the reward the team receives for the agent performing a task. Mathematically, the capability of e_i to perform θ_j is given by: $Cap(e_i, \theta_j) \rightarrow [0, 1]$; if $Cap(e_i, \theta_j) > 0$, we say that e_i is *functional* for θ_j. In extreme teams, $Cap(e_i, \theta_j) > 0$ for a significant proportion (or even all) e_i for each θ_j, to model overlapping functionality. We assume that each agent has a single type of resource with which to perform tasks, and denote the amount of resources available to e_i by $e_i.res$; e_i must spend $Res(e_i, \theta_j)$ to perform θ_j.

Following convention, we define an allocation matrix A, where a_{ij} is the value of the ith row and jth column given by

$$a_{ij} = \begin{cases} 1 & \text{if } e_i \text{ is performing } \theta_j \\ 0 & \text{otherwise} \end{cases}$$

The goal in GAP is to find A that maximizes team reward:

$$A = \arg\max_{A'} \sum_{e_i \in E} \sum_{\theta_j \in \Theta} Cap(e_i, \theta_j) \times a'_{ij}$$

such that all agents' resource limitations are respected:

$$\forall e_i \in E, \sum_{\theta_j \in \Theta} Res(e_i, \theta_j) \times a_{ij} \leq e_i.res$$

and at most one team member performs each task:

$$\forall \theta_j \in \Theta, \sum_{e_i \in E} a_{ij} \leq 1$$

While GAP captures many aspects of task allocation, its simplistic relationship between capability and reward does not capture interdependencies between tasks. Also, the solution A corresponds to a single static allocation, and thus is not suited for dynamic domains. We tackle both shortcomings next by extending GAP.

Extended GAP

Coordination constraints, $\bowtie$, are interdependencies between tasks. For example, in an AND constraint, the team only receives reward for each task if all the constrained tasks are simultaneously executed. An AND constrained set of tasks can be used to represent a task that requires multiple agents to successfully perform (such as extinguishing a large fire). More complex coordination constraints such as XOR or XOR-K may specify that exactly one or exactly K of the constrained tasks be simultaneously performed, or else the team suffers a penalty. We explicitly focus on AND constraints here, but the formalization can be extended to these other constraint types as well.

Let $\bowtie = \{\alpha_1, \ldots, \alpha_p\}$, where $\alpha_k = \{\theta_{k_1}, \ldots, \theta_{k_q}\}$ denotes the kth set of AND constrained tasks. The number of tasks in α_k that are being performed is then

$$x_k = \sum_{e_i \in E} \sum_{\theta_{k_j} \in \alpha_k} a_{ik_j}$$

Letting $v_{ij} = Cap(e_i, \theta_j) \times a_{ij}$, we then have that the value of e_i performing θ_j given $\bowtie$ is

$$Val(e_i, \theta_j, \bowtie) = \begin{cases} v_{ij} & \text{if } \forall \alpha_k \in \bowtie,\ \theta_j \notin \alpha_k \\ v_{ij} & \text{if } \exists \alpha_k \in \bowtie \text{ with } \theta_j \in \alpha_k \wedge x_k = |\alpha_k| \\ 0 & \text{otherwise} \end{cases}$$

where the first case is the reward for unconstrained tasks and the last two are for constrained tasks.

To introduce the dynamics of extreme teams into GAP, we index Θ, E, Cap, Res, $\bowtie$ and Val by time. The most important consequence of this is that we no longer seek a single allocation A; rather we need a sequence of allocations, $A^{\rightarrow}$, one for each discrete time step. A delay cost function, $DC^t(\theta_j^t)$, captures the cost of not performing θ_j^t at time t. Thus, the objective of the E-GAP problem is to maximize:

$$f(A^{\rightarrow}) = \sum_t \sum_{e_i^t \in E^t} \sum_{\theta_j^t \in \Theta^t} (Val^t(e_i^t, \theta_j^t, \bowtie^t) \times a_{ij}^t) - \sum_t \sum_{\theta_j^t \in \Theta^t} (1 - \sum_{e_i^t \in E^t} a_{ij}^t) \times DC^t(\theta_j^t)$$

such that

$$\forall t, \forall e_i^t \in E^t, \sum_{\theta_j^t \in \Theta^t} Res^t(e_i^t, \theta_j^t) \times a_{ij}^t \leq e_i^t.res$$

and

$$\forall t, \forall \theta_j^t \in \Theta^t, \sum_{e_i^t \in E^t} a_{ij}^t \leq 1$$

Thus, extreme teams must allocate tasks rapidly to accrue rewards, or else incur delay costs at each time step.

3. LA-DCOP

LA-DCOP is a DCOP algorithm that attempts to solve E-GAP in an approximate fashion, since high delay costs and dynamic changes in costs precludes an optimal response. In the DCOP framework, each agent is provided with a variable to which it must assign values[4, 23, 11] which correspond to tasks the agent will perform. Since agents can execute multiple tasks at once, variables can take on multiple values simultaneously, as in graph multicoloring. LA-DCOP exploits key properties of extreme teams that arise due to their large scale and similar agent functionality. The task allocation algorithms run by each agent is shown in Algorithms 1 and 2

A central requirement of E-GAP is that at most one team member performs each task, or, in DCOP terms, the same value is not assigned to two distinct variables. Thus, there is a "not-equal" constraint between every agent with functionality for the same task, which results in dense graphs due to the overlapping functionality of extreme team members. Dense graphs are problematic for DCOP algorithms[11, 4] because of the large amount of communication required to remove conflicts. To avoid this communication, we create a *token* for each value. The holder of a token has the exclusive right to assign the corresponding value to its variable, and must either do so or pass the token to a teammate. In this way, conflicts cannot occur and so communication is reduced.

Given the token-based access to values, the decision for the agent becomes whether to assign to its variable values represented by tokens it currently has or to pass the tokens on. First, a team member must decide whether it is in the best interests of the team for it to assign the value represented by a token to its variable (Alg 1, line 8). Algorithms like DSA and DBA[23] attempt hill-climbing at each step by enabling an agent to change its value to enable maximum gain to the team, given knowledge of neighboring agents. However, communication of neighboring agents' values is expensive (section 5 provides detailed experimental results). Instead, LA-DCOP uses a *threshold* on the minimum capability an agent must have in order to assign the value. This threshold is attached to the token. If the agent computes that its own capability is less than the minimum threshold, it passes it randomly to a teammate. (To avoid agents passing tokens back and forth, each token maintains the list of agents it has visited; if all agents have been visited, the token can revisit agents, but only after a small delay.) In this way, LA-DCOP performs a search for a local maximum similar to DBA and DSA, but without additional communication beyond passing the token; the threshold guides the tokens towards agents with higher capabilities to perform them.

The burden of finding a good allocation thus rests with computing good thresholds. Computing thresholds that maximize expected utility is a key part of this algorithm and is described in Section 4. The threshold is calculated once (Alg 1, line 7), 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, agents will be able to respond by changing the threshold for newly-created tokens. This allows the team great 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 agent must check whether the value can be assigned while respecting its local resource constraints (Alg. 1, line 15). If the value cannot be assigned within the resource constraints of the team member, it must choose a value(s) to reject and pass on to other teammates in the form of a token(s) (Alg. 1, lines 20 and 22). The agent chooses the set of values that maximize the sum of its capabilities for those values, while respecting its resource constraints (performed in the MAXCAP function, Alg. 1, line 16), and so acts in a locally optimal manner.

Algorithm 1: VarMonitor

```
(1)  V ← ∅, PV ← ∅
(2)  while true
(3)    msg ← getMsg()
(4)    if msg is token
(5)      token ← msg
(6)      if token.threshold = NULL
(7)        token.threshold ← CALCTHRESHOLD(token)
(8)      if token.threshold < Cap(token.value)
(9)        if token.potential
(10)         PV ← PV ∪ token.value
(11)         SENDMSG(token.owner, "retained")
(12)       else
(13)         V ← V ∪ token.value

(15)       if Σ_{v∈V} Resources(v) ≥ agent.resources
(16)         out ← V − MAXCAP(V)
(17)         foreach v ∈ out
(18)           if v.potential
(19)             SENDMSG(pv.owner, "release")
(20)             PASSON(new token(v, potential))
(21)           else
(22)             PASSON(new token(v))
(23)           V ← V − out

(25)     else
(26)       PASSON(token) /* threshold < Cap */
(27)   else if msg is "lock v a*"
(28)     if v ∈ PV
(29)       PV ← PV − v
(30)       V ← V ∪ v
(31)     else
(32)       ∀a ∈ a* SENDMSG(a, "released")
(33)   else if msg is "release v"
(34)     PV ← PV − v
```

AND Constrained Tasks

In addition to dynamics, E-GAP presented the difficulty of coordination constraints between tasks. When there are AND constraints between tasks there is the potential for *deadlocks* or, at best, severe inefficiencies. To avoid such problems we introduce the idea of *potential values*. A second algorithm, shown in Algorithm 2, runs alongside Algorithm 1 and works as follows. The tokens for all tasks in an AND constrained set are given to one team member. For each of the tokens the team member sends out a small number of *potential tokens* (Alg. 2, line 3). The potential tokens work in exactly the same way as "normal" tokens except that when a team member accepts a potential token it agrees to accept the task represented by the token (Alg. 1, line 10), *only* if a potential token for each of the other real tokens is accepted by another agent and may perform other tasks in the meantime. Thus, LA-DCOP allows agents to continue working on individual tasks while subteams are formed for constrained tasks. This parallelism is not available to other DCOP algorithms and is a major advantage of LA-DCOP.

When the team member holding the real tokens is informed that at least one potential token for each real token has been accepted by a team member it *locks* the group. Locking is done by selecting the holder of one potential token for each real token and sending them the real token (Alg. 2, line 15). A list of agents accepting the other real tokens is also sent. Note that this mechanism guards against deadlocks: if an agent a sends a "Release" message first and then receives a "Lock" message, a is now responsible for sending messages to other receivers of the "Lock" message to also release (Alg. 1, line 32). Holders of potential tokens that are not replaced with real tokens are also released (Alg. 2, line 19).

Algorithm 2: ANDMonitor

```
(1)  foreach v ∈ V
(2)    for 1 to No. Potential Values
(3)      PASSON(new token(v,potential))

(5)  /* Wait to accept potential tokens */
(6)  while Π_{v∈V} |Retained[v]| = 0
(7)    msg ← getMsg()
(8)    if msg is "retained v"
(9)      Retained[v] ← Retained[v] ∪ msg.sender
(10)   else if msg is "release v"
(11)     Retained[v] ← Retained[v] − msg.sender

(13) /* Send real tokens */
(14) foreach v ∈ V
(15)   a* = ∀a ∈ Retained[v] Cap(a*, v) > Cap(a, v)
(16)   foreach a ∈ a*
(17)     SENDMSG(a, { "lock v", a* })
(18)   foreach a ∈ Retained[v] − a*
(19)     SENDMSG(a, "release v")

(21) /* Watch out for agent releasing after lock */
(22) while true
(23)   msg ← getMsg()
(24)   if msg is "release v"
(25)     Retained[v] ← Retained[v] − msg.sender
(26)     goto 6
```

Observe that a similar approach would be sufficient for other constraints such as $XOR - K$. Instead of waiting for all agents to respond, a lock could be issued as soon as potential tokens for the first K tasks are accepted, and any agents not part of the locked group could be released. The flexibility to deal with multiple types of constraints demonstrates the generality of the potential token approach.

4. CALCULATING THRESHOLDS

In this section, we present a model which allows calculation of the *maximum expected utility (MEU) threshold* for one simple class of problems. This type of calculation can be done by a team member to determine the best threshold for a newly-created token, as described in the previous section. Our calculation is based on the *expected utility* (EU) to the team of using that threshold. Specifically, we calculate an expectation of which tasks will be executed and the capability of the agents that will be executing those tasks when the algorithm settles into a steady state. Abstractly, we can write the EU of using a particular threshold, T, as:

$$\begin{aligned} EU(T) &= E(\#\ tasks\ executed|T) \times \\ &\quad E(capability\ of\ capable\ agent|T) \\ &= E(\#\ capable\ agents|T) \times \\ &\quad E(\#\ tasks\ per\ capable\ agent|T) \times \\ &\quad E(capability\ of\ capable\ agent|T) \end{aligned}$$

where a capable agent has at least one capability above the threshold. Notice that since we are using expectations for each value, the result is an expectation of the utility to the team, not a precise calculation of the utility it will receive. While the above equation is the most general, calculating the values of the the terms for specific teams is non-trivial. Below we look at class of models that covers a wide range of extreme team domains.

We assume that *classes* of tasks require the same capability and that there are M tasks, N agents and K classes of tasks ($\frac{M}{K}$ tasks of each class). Each agent has a capability for a class of tasks chosen from a uniform, random distribution over $[0, 1]$. An agent's capability to perform one type of task is independent of its ability to perform any other type.

We also assume that each agent has one normalized unit of resources (i.e., $\forall e,\ e.res = 1$). Tasks within a class require different amounts of resources. Specifically, we discretize the resource requirements of tasks to $0 < r_1 < r_2 < \ldots < r_q \leq 1$ and say that a proportion p_i of the tasks requires an amount r_i of resources. To execute all tasks requiring a specific amount of resource, r_i, requires number of agents $N_{r_i} = p_i \times r_i \times M$ (which is an approximation of $\lfloor \frac{1}{r_i} \rfloor \times N_{r_i} = p_i \times M$).

Due to the independence and uniformity of the capability distributions, we can write $E(\#capable\ agents|T) = (1-T^K) \times N = N_T$. Due to the independence between capability distributions, if all tasks cannot be performed, a good approximation of the highest utility is received when the tasks requiring the most resources are not performed. Thus we can write a calculation based on assigning tasks requiring least resources first, as shown in Eqn. 1. Given the uniform capability distribution, the capability of an agent performing a task will be $\frac{1+T}{2}$. Hence, substituting for N_T and N_{r_i}, we get the equation for EU given T shown in Eqn. 2. Since this is a continuous, piecewise function, if we take the maximum of each of the pieces, we see that the maximum of these is the maximum of the overall function. We can readily determine the maximizing value of T on each of the pieces via linear time numeric methods, and so find the maximizing value for T. In the next section, we show that the MEU threshold determined via this approach yields a reward that is very close to the experimentally determined maximum.

5. EXPERIMENTS AND RESULTS

We have tested LA-DCOP extensively in three environments. The first is an abstract simulator that allows us to run many experiments with very large numbers of agents[14]. We simulate five different classes of task. Each of the agents had randomly assigned capabilities uniformly drawn from zero to one for each of the different classes of task. For each time step that the agent has the task and has the resources to execute it, the team receives a reward equal to the agent's capability. The team aims to maximize the sum of total reward over the length of the simulation. Message passing is simulated as perfect (lossless) communication in a fully connected network. During each simulation step, each token was allowed to move from one agent to another only once. 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 in the Figures below represents the average from 20 runs. Notice that the experiments below are based on this specific setup of the simulation. However, a large number of additional experiments with other configurations were performed, e.g., more different classes of task or different distributions of capabilities, and while there were some differences, the results below are representative of the results achieved.

The first set of experiments tested LA-DCOP against three competitors. The first is DSA, which is shown to outperform other approximate DCOP algorithms in a range of settings [11, 4]; we choose optimal parameters for DSA [23]. As a baseline we also compare against a centralized algorithm that uses a "greedy" assignment[3]. Results are shown for LA-DCOP using two different thresholds, T=0.0, i.e., keep a token if functional and have available resources, and T=0.5 which was determined to give good performance for these configurations. Figure 1(a) shows the relative performance of each algorithm as the number of agents is

$$E(\#\ tasks\ executed|T) = \begin{cases} \frac{1}{r_1} N_T & \text{if } N_T < N_{r_1} \\ \frac{1}{r_u} N_T + \sum_{i=1}^{u-1} (1 - \frac{r_i}{r_u}) P_i M & \text{if for any } u \in \{2, q\}, \sum_{i=1}^{u-1} N_{r_i} < N_T < \sum_{i=1}^{u} N_{r_i} \\ M & \text{otherwise} \end{cases}$$

Eqn 1: Calculation of the number of tasks executed.

$$EU(T) = \begin{cases} \frac{1}{r_1}(1 - T^K) N \frac{1+T}{2} & \text{if } T > (1 - \frac{r_1 p_1 M}{N})^{\frac{1}{K}} \\ (\frac{1}{r_u}(1 - T^K) N + \sum_{i=1}^{u-1} (1 - \frac{r_i}{r_u}) p_i M) \frac{1+T}{2} & \text{if for any } u \in \{2, q\}, (1 - \frac{\sum_{i=1}^{u} r_i p_i M}{N})^{\frac{1}{K}} < T < (1 - \frac{\sum_{i=1}^{u-1} r_i p_i M}{N})^{\frac{1}{K}} \\ M \frac{1+T}{2} & \text{if } (1 - \frac{\sum_{i=1}^{q} r_i p_i M}{N})^{\frac{1}{K}} > T \end{cases}$$

Eqn 2: Calculation of the expected utility of a particular threshold, T.

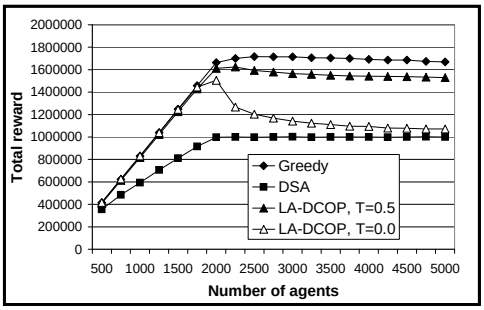
increased. The experiment used 2000 tasks over 1000 time steps. The y-axis shows the total reward, while the x-axis shows the number of agents. Not surprisingly, the centralized algorithm performs best. LA-DCOP performs significantly better with a threshold of 0.5 than with no threshold. LA-DCOP with either threshold statistically outperforms DSA (with probability greater than 99.9%, as determined by a t-test).

The real key to the comparison, however, is the amount of communication used, as shown in Figure 1(b). Notice that the y-axis is a logarithmic scale; thus LA-DCOP uses approximately four orders of magnitude fewer messages than the greedy algorithm and six orders of magnitude fewer messages than DSA. LA-DCOP performs better than DSA despite using only a tiny fraction of the number of messages and only marginally worse than a centralized approach, despite using far less communication. We can also see a tradeoff in the volume of messages that LA-DCOP uses compared to its reward; with no thresholds, LA-DCOP uses fewer messages than with a threshold of 0.5, at the cost of reduced reward.

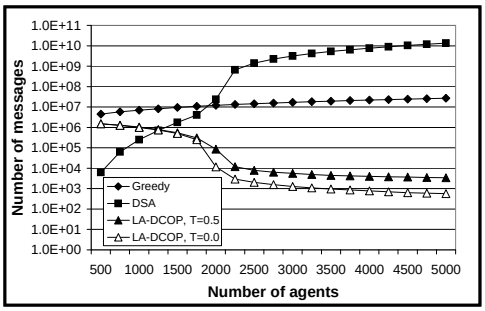
To validate the calculation of MEU threshold, Figure 2a shows the reward found experimentally versus the expected reward as calculated via the theory when the ratio of tasks to agents (the *load*) is 1. The data points have a correlation coefficient of 0.9679. The close match of the theory and experimental results illustrates that we can rely on mathematical analysis to approximate MEU thresholds.

Figure 2b shows the reward obtained using different thresholds over experiments with loads of 0.2, 0.5, and 2.0, averaged over 20 runs each. Such load variance models expected dynamic events in extreme team domains, e.g., the spread of fires causing an explosion in disaster rescue. As load is increased, the threshold that yields maximal reward decreases. However, no single fixed threshold is able to maximize reward under all three loads. The first bar, labeled DC, shows the reward obtained using the MEU threshold for each load (as calculated by Equation 2), as is done in LA-DCOP. The figure clearly shows that the LA-DCOP approach of dynamically computing thresholds outperforms fixed, static thresholds under varying load.

Even when load does not change dynamically in an extreme team domain, tasks will often turn over at a rapid rate. In Figure 3a, we show that LA-DCOP performs well even when this change is very rapid. The four lines represent different rates of change, with 0.01 meaning that every time step (i.e., the time it takes to send one message) 1% of all tasks are replaced with tasks potentially requiring a different capability. The x-axis measures the probability that an agent is functional in type of task. When this value is 50%, with 1% dynamics, LA-DCOP loses 10% of reward/agent on average,



(a)



(b)

Figure 1: (a) comparing the reward versus the number of agents. (b) the number of messages sent versus the number of agents

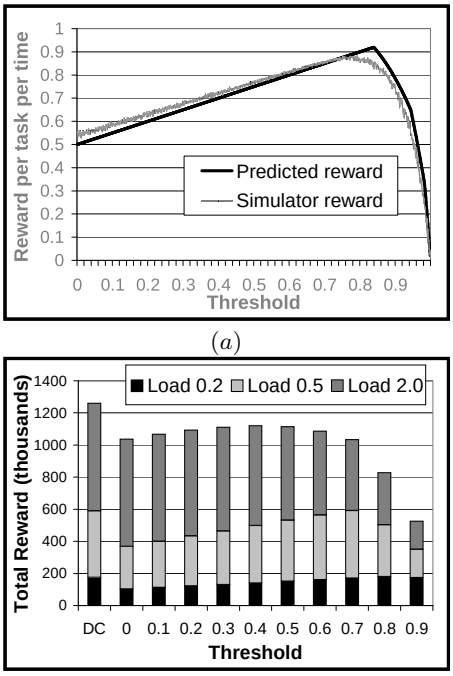


Figure 2: (a) comparison between theoretical and experimental reward versus threshold. (b) effect of thresholds on total reward for different loads of tasks/agents.

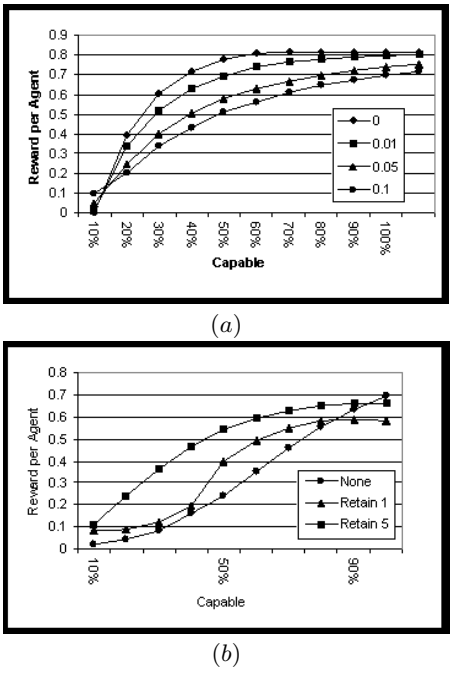


Figure 3: (a) the effects of different proportions of tasks changing each step. The y-axis shows the output, x-axis shows the percentage of agents with capability > 0. (b) the effect of retainers, with the lines representing no retainers, one retained task per agent and five retained tasks per agent.

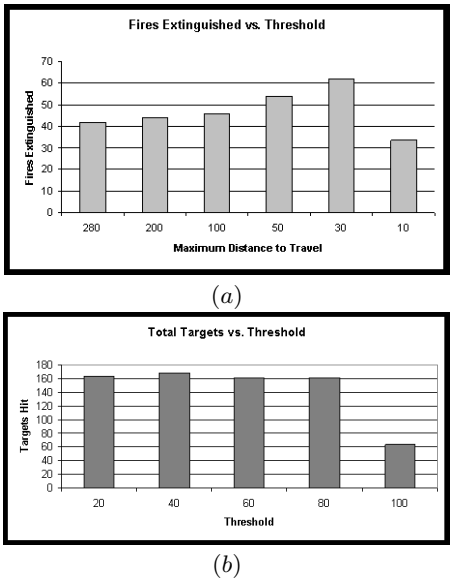


Figure 4: (a) the number of fires extinguished by 200 fire trucks versus threshold (b) the number of targets hit by UAVs versus threshold.

but when more agents are likely to be functional, the loss due to even high dynamics is within 10% reward/agent.

Finally, Figure 3b shows the utility of potential tokens when groups of tasks are AND constrained. In the figure, 60% of all tasks (900 tasks) are AND constrained into groups of five tasks. Unless a functional agent is assigned to each task in the group, the team receives no reward. It is clear that potential tokens help since the lowest output is received without the potential tokens (labeled "None"). Moreover, allowing agents to have up to five potential tokens (labeled "Retain 5") leads to better performance than allowing them to have only one potential token (labeled "Retain 1"). The effect is most pronounced when about 40% of agents are functional because this is the case when most deadlocks and idleness occur otherwise.

In our second set of experiments, we used 200 LA-DCOP enhanced versions of Machinetta proxies[19], distributed over a network, executing plans in two simple simulation environments. The proxies execute sophisticated teamwork algorithms as well as LA-DCOP and thus provide a realistic test of LA-DCOP. The first environment is a version of a disaster response domain where fire trucks must fight fires. Capability in this case is the distance of the truck from the fire, since this affects the time until the fire is extinguished. Hence, in this case, the threshold corresponds to the maximum distance the truck will travel to a fire. Figure 5(a) shows the number of fires extinguished by the team versus threshold. Increasing thresholds initially improves the number of fires extinguished, but too high a threshold results in a lack of trucks accepting tasks and a decrease in performance. In the second domain, 200 simulated unmanned aerial vehicles (UAVs) explore a battle space, destroying targets of interest. While in this domain LA-DCOP effectively allocates tasks across a large team, thresholds are of no benefit. The key point of these experiments is to show that LA-DCOP can work effectively, in a fully distributed environment with realistic domains and large teams.

RoboCup Rescue Experiments

We also tested our approach in the RoboCup Rescue environment [8]. RoboCup Rescue provides an ideal, realistic testing ground

for LA-DCOP in allocating roles to an extreme team comprised of fire engines. Our experimental setting features 10 fire fighters and 18 ignition points. We considered different distributions of agents and fires, testing our approach in situations where fires are clustered in one, three, and four regions of the map (Clusters-1, Clusters-3, and Clusters-4, respectively). Figure 5 shows the RoboCup Rescue simulator with three clusters of fires.

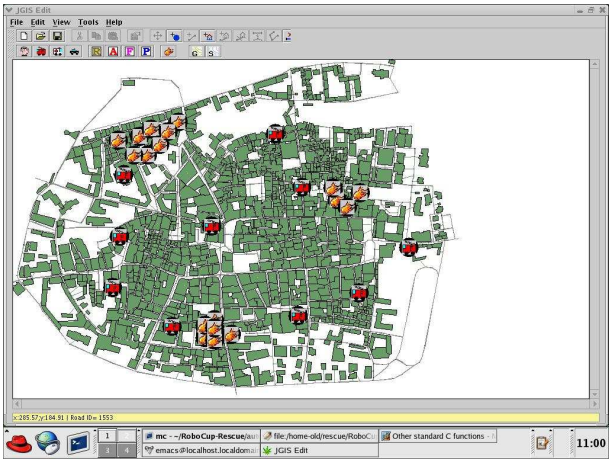


Figure 5: RoboCup Rescue environment with three clusters of fires

In previous work, researchers have documented the failure of auction based algorithms for role allocation in RoboCup Rescue[12], due to the high communication requirements. To test whether LA-DCOP can allocate roles within the communication and time limitations of RoboCup Rescue, we compared against a shortest distance based strategy, which exploits domain characteristics and is similar to that used by top-performing RoboCup Rescue teams. Agents' capabilities are computed considering whether the agent is blocked or not and its current distance from the fire. Because the number and strength of fires varies with time, we also compared against LA-DCOP run with fixed thresholds.

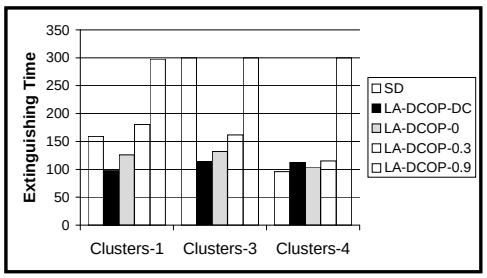


Figure 6: LA-DCOP outperforms SD

Figure 6 compares the different strategies, averaged over 20 runs. LA-DCOP with dynamically computed thresholds (LA-DCOP-DC) is seen to outperform (i.e., extinguish fires faster than) competitors for the Clusters-1 and Clusters-3. Indeed, in Clusters-3, LA-DCOP-DC extinguishes fires in 100 time units, while SD is unable to extinguish the fires within even 300 units (our cutoff). In Cluster4, fires spread throughout the city, creating a scenario that is very difficult for LA-DCOP-DC. The key to note is that even in this difficult cluster4 scenario, LA-DCOP-DC is performing similarly to SD.

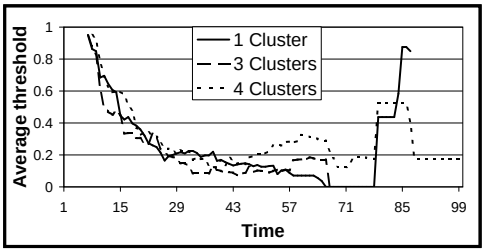


Figure 7: average threshold versus time

Figure 7 shows how the thresholds in LA-DCOP-DC change with time. The thresholds averaged over all tokens are plotted for each of the three scenarios. Average thresholds begin high, then fall as load increases. Since thresholds stay constant once assigned to tokens, this means that as new tasks arise, the MEU thresholds calculated for them are lower than at previous times. This ability under LA-DCOP to compute thresholds based on current conditions gives it valuable flexibility in dealing with the dynamic domains in which extreme teams must operate.

6. SUMMARY AND RELATED WORK

In this paper, we have described a novel approach to task allocation in extreme teams. Our DCOP based approach substantially outperforms other approximate DCOP algorithms, both in total reward and in communication, where we demonstrated a dramatic six orders of magnitude reduction in messages. It allows a scaling up in team size by an order of magnitude, while coping with additional challenges of extreme team domains that other algorithms cannot address. In particular, the ability to use team knowledge to dynamically compute MEU thresholds allows LA-DCOP to find good allocations even in dynamic domains. The strengths and limitations of LA-DCOP will be thoroughly tested in coming months as in plays a key role in some major projects. The DEFACTO project is aimed at developing high fidelity simulation environments for training of rescue response teams (see Figure 8) and LA-DCOP must perform task assignment for a large, dynamic rescue response team[20]. The CAMRA project is focused on coordination of unmanned aerial vehicles and LA-DCOP will feature in a major flight test in late 2005[18].

Task allocation is an extensively studied area with work ranging from high complexity, forward looking optimal models[12], to symbolic matching that ignores cost[22, 16], to centralized auctions[7], to swarm techniques[17, 1, 2], to distributed constraint optimization[23, 11, 10]. Among these, the forward looking optimal models and centralized auctions are not only highly centralized, but their computationally expensive considerations of optimality lead to difficulties in their application in highly dynamic extreme team domains. The symbolic matching models ignore costs completely, which is highly detrimental. Swarm techniques use local sensing to modulate flexibility, but LA-DCOP permits additional global knowledge to factor into thresholds. Finally, we have discussed the DCOP models, particularly incomplete DCOP algorithms, in detail throughout the paper and presented comparison of our work to these algorithms. Complete DCOP algorithms like ADOPT[11] and OptAPO[10] are appropriate in key domains where optimality is critical, but the significant amount of communication engendered would be highly problematic in densely connected constraint

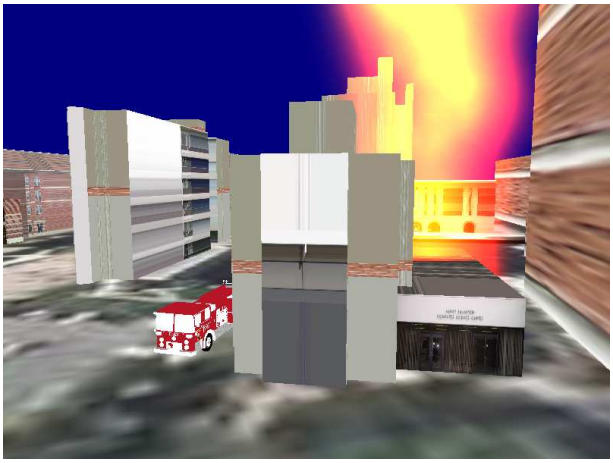


Figure 8: A view of a disaster response scenario in the DEFACTO project.

graphs in extreme teams, and they are unable to handle dynamics of extreme team domains.

7. 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] M. Campos, E. Bonabeau, G. Therauluz, and J.-L. Deneubourg. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 2001.

[3] 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.

[4] 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.

[5] 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.

[6] H. Hexmoor and X. Zhang. Socially intelligent air combat simulator. In *PRICAI-02*, 2002.

[7] L. Hunsberger and B. Grosz. A combinatorial auction for collaborative planning, 2000.

[8] Hiraoki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, , and Hitoshi Matsubara. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.

[9] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking dcop to the real world : Efficient complete solutions for distributed event scheduling. In *AAMAS'04*, 2004.

[10] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS'04*, 2004.

[11] 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.

[12] R. Nair, T. Ito, M. Tambe, and S. Marsella. Task allocation in robocup rescue simulation domain. In *Proceedings of the International Symposium on RoboCup*, 2002.

[13] 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.

[14] Steven Okamoto. Dcop in la: Relaxed. Master's thesis, University of Southern California, 2003.

[15] Committee on Visionary Manufacturing Challenges. Visionary manufacturing challenges for 2020. National Research Council.

[16] 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.

[17] Pedro Sander, Denis Peleshchuk, and Barabara Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of AAMAS'02*, 2002.

[18] 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.

[19] 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.

[20] N. Schurr, J. Marecki, J.P. Lewis, M. Tambe, and P.Scerri. The DEFACTO system: Training tool for incident commanders. In *IAAI'05*, 2005.

[21] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

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

[23] W. Zhang and L. Wittenburg. Distributed breakout revisited. In *Proceedings of AAAI'02*, 2002.