Token Approach for Role Allocation in Extreme Teams: analysis and experimental evaluation

Paul Scerri*, Alessandro Farinelli+, Stephen Okamoto# and Milind Tambe#
* Carnegie Mellon University
+ University of Rome "La Sapienza"
# University of Southern California
pscerri@cs.cmu.edu, Alessandro.Farinelli@dis.uniroma1.it, sokamoto@usc.edu, tambe@usc.edu

Abstract

Open Computational systems comprise physical entities coordinating their activities in dynamic environments. Many exciting applications require a large number of such entities to achieve team coordination in complex missions execution. To meet the fundamental challenge of role allocation in such extreme teams, we propose an algorithm called LA-DCOP, that overcomes the limitations of previous algorithms by incorporating three key ideas. First, we represent the role allocation problem as a Distributed Constraint Optimization Problem and use tokens *representing roles to minimize constraint violations. Second, we use probabilistic information about the team to guide the search quickly towards* good *solutions Third, we designed the algorithm to manage constrained roles. We show that LA-DCOP not only meets our requirements in extreme teams, but also compares favorably against previous role allocation algorithms. LA-DCOP has allowed an order of magnitude scale-up in extreme teams, with role allocation in a fully distributed proxy-based teams with up to 200 members.*

1 Introduction

Open Computational Systems (OCS), comprise physical entities that have to achieve a common goal, interacting among them through a networked infrastructure[9]. Team coordination is a key issue for OCS and the rapid advances obtained in the design of team coordination infrastructures enabled in recent years a wide range of exciting applications[11]. Typical domains for OCS involve distributed health care, manufacturing control, mobile robots and sensor networks[9, 5]. A key requirement in recent applications for such domains is to effectively coordinate *extreme teams*, which are large teams that need (soft) real-time response given dynamic tasks, and where many *resource limited* agents have similar functionality, but possibly varied capability. For instance, when responding to a disaster, fire fighters and paramedics comprise an extreme team as they must respond rapidly to dynamic tasks; and fire fighters can all extinguish fires although their capability to extinguish a particular fire quickly will depend on their initial distance from that fire. Other examples of extreme teams include mobile sensor or UAV teams, robot teams for Mars colonies, as well as large-scale future integrated manufacturing and service organizations.

This paper focuses on the critical challenge of role allocation in extreme teams. In general in teamwork, role allocation is the problem of assigning roles to agents so as to maximize overall team utility[7, 13, 14]. Extreme teams emphasize key additional requirements in role allocation: (i) rapid role allocation as domain dynamics may cause tasks to disappear; (ii) agents may perform one or more roles, but within resource limits; (iii) many agents can fulfill the same role; (iv) inter-role constraints may be present. This role allocation challenge in extreme teams will be referred to as E-GAP, as it subsumes the generalized assignment problem (GAP), which is NP-complete[12].

This paper focuses on Distributed Constraint Optimization (DCOP)[6, 1] for role allocation, as DCOP offers the key advantages of distributedness and a rich representational language which can consider costs/utilities of tasks. Despite these advantages, DCOP approaches to role allocation suffer from three weaknesses. First, complete DCOP algorithms[6] have exponential run-time complexity and, thus, fail to meet the response requirements of extreme teams. One reason for this is that the purely local view of the team that each agent has, forces the search to explore many potential solutions that are clearly sub-optimal. However, teams of agents will often have reasonably accurate estimates of both the situation and the state of the team which can be used to accurately estimate likely solution characteristics. For example, when a team of fire fighters responds to a disaster, it is reasonable to assume that they know the number of fires and number of available fire trucks to within

an order of magnitude, even though they may have very little specific knowledge of individual fires or trucks. While relying on such estimates prevents guarantees of optimality, they can dramatically reduce the search space. Second, similar agent functionality within extreme teams results in dense constraint graphs increasing communication within a DCOP algorithm. Third, DCOP algorithms do not address the additional complications of constraints *between* roles.

To address these limitations in addressing E-GAP, we propose a novel DCOP algorithm called LA-DCOP (Low communication Approximate DCOP). LA-DCOP uses a representation where agents are variables that can take on values from a common pool, i.e., the pool of roles to be assigned. The mechanism for allocating values to variables encapsulates three novel ideas. First, LA-DCOP improves efficiency by not focusing on an exact optimal reward; instead by exploiting the likely characteristics of optimal allocations, given the available probabilistic information, it focuses on maximizing the team's expected total reward. In particular, the agents *compute* a minimum threshold on the expected capability of the agent that would maximize expected team performance. If the agent's capability to perform a role is less than the threshold capability, it does not consider taking on the role, channeling the role towards more capable agents. Second, to reduce the significant communication overheads due to constraint graph denseness, *tokens* are used to regulate access to values. Only the agent currently holding the token for a particular value can consider assigning that value to its variable. The use of tokens removes the possibility of several agents taking on the same role, thus dramatically reducing the need to communicate about and repair conflicts. Third, to deal with groups of tightly constrained roles, we introduce the idea of allowing values to be represented by *potential tokens*. When groups of roles must be simultaneously performed, instead of committing to a role by assigning the value represented by a token, a team member accepts a potential token. This indicates that it will accept the role only when all simultaneous roles can be assigned. While team members are being found to fill the other simultaneous roles, a team member with a potential token can perform other roles. Only when team members have been found for all roles will the holders of the potential tokens actually take on the roles. This technique frees team members up for other roles when not all roles in a constrained set can be filled.

Using a mixture of high and low fidelity simulation environment, we have extensively empirically evaluated the LA-DCOP algorithm. Experiments on a simplified testbed illustrate three key points. First, the key features of the algorithm, including thresholds and potential tokens do significantly improve its performance. Second, when compared to other DCOP algorithms on simplified role allocation problems,[1] LA-DCOP performed very well, finding better allocations than other approximate algorithms, while using up to four orders of magnitude fewer messages. Third, we illustrate that the algorithm performs well on two realistic domains, by embedding it in teamwork proxies. Prior research on teamwork proxies had demonstrated teams limited to 20 agents, partially due to limitations on the role allocation algorithm. In our work we dramatically escalate the team size to 200 proxies and illustrate the effective role allocation performance over these 200 agents.

[1] Other DCOP algorithms cannot be easily adapted to E-GAP

2 Problem Statement

A GAP problem is defined by team members for performing roles and roles to be assigned[12]. Each team member, $e_i \in E$, is defined by their capability to perform roles, $R = \{r_1, \ldots, r_n\}$, and their available resources. The capability of a team member, e_i, to perform a role, r_i, is quantitatively given by: $Cap(e_i, r_i) \rightarrow [0, 1]$. Capability reflects the quality of the output or the speed of task performance or other factors affecting output. Each role requires some resources of the team member in order to be performed. We write the resource requirements of a team member e_k for a role r_j as $Resources(e_k, r_j)$ and the available resources of an agent, e, as $e.resources$.

Following convention, we define a matrix A, where $a_{i,j}$ is value of the ith row and jth column and

$$a_{i,j} = 1 \text{ if } e_i \text{ is performing } r_j \text{ otherwise } a_{i,j} = 0$$

Thus, the matrix A defines the allocation of roles to team members. The goal in GAP is to maximize:

$$f(A) = \sum_{e \in E} \sum_{r \in R} Cap(e, r) \times a_{e,r}$$

such that

$$\forall e \in E \sum_{r \in R} Resources(e, r) \times a_{e,r} \leq e.resources$$

and

$$\forall r \in R \sum_{e \in E} a_{e,r} \leq 1$$

Intuitively this says that we attempt to maximize the capabilities of the agents assigned to roles, subject to the resource constraints of team members, ensuring that at most one team member is assigned to each role but potentially more than one role per team member.

Extended GAP

Coordination constraints, $\bowtie$, tie groups of roles together. Although one can imagine a wide range of constraint types, here we specifically discuss only one type: AND constraints. When a set of roles are AND constrained the team only receives value if all the constrained roles are simultaneously being executed. We write AND constrained roles as $AND^i = \{r_1, ..., r_k\}$. Thus, $\bowtie = \{AND^1, \ldots, AND^n\}$. When $r_i \in AND^j$, the value of r_i to the team depends on whether other roles in AND^j are allocated. That is,

$$Val(e_i, r_i, AND^j) = \begin{cases} Cap(e_i, r_i) & if \sum_{e \in E} \sum_{r \in AND^j} a_{e,r} = |AND^j| \\ 0 & \text{otherwise} \end{cases}$$

Notice that if the role is unconstrained, $|AND^j| = 1$, and this degenerates to $Cap(e, r) \times a_{e,r}$, as above.

To introduce the dynamics of extreme teams into GAP we make R, E, Cap and $Resources$ functions of time. The most important consequence of this is that we no longer need a single allocation A, but a sequence of allocations, $A^{\rightarrow}$, one for each discrete time step. A delay cost function, $DC(r_i, t)$, captures the cost of not performing r_i at time t. Thus, the objective of the E-GAP problem is to maximize:

$$f(A^{\rightarrow}) = \sum_t \sum_{e \in E} \sum_{r \in R} (Val(e, r, \bowtie, t) \times a_{e,r,t}) - \sum_t \sum_{r \in R} (1 - \sum_{e \in E} a_{e,r,t}) \times DC(r, t)$$

such that

$$\forall e \in E \sum_{r \in R} Resources(e, r) \times a_{e,r,t} \leq e.resources$$

and

$$\forall r \in R \sum_{e \in E} a_{e,r,t} \leq 1$$

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

3 LA-DCOP

Given the response requirements for agents in extreme teams, they must solve E-GAP in an approximate fashion. We propose LA-DCOP, a DCOP algorithm for addressing E-GAP in a distributed fashion, which exploits key properties of extreme teams that arise due to large-scale and similarity of agent functionality (e.g., using probability distributions), and simultaneously, guards against special role allocation challenges of extreme teams (e.g., inability of strong decomposition into smaller subproblems.) In DCOP, each agent is provided with one or more variables and must assign values to variables[1, 15, 6]. LA-DCOP maps team members to variables and roles to values, as shown in Algorithm 1. Thus, a variable taking on a value corresponds to a team member taking on a role. Since team members can take on multiple roles, the variable can take on multiple values, as in graph multi-coloring, simultaneously.

In E-GAP, a central constraint is that each role should be assigned to only one team member, corresponding to each value being assigned by only one variable. In DCOP, this requires having a complete graph of *not equals* constraints between variables (or at least a dense graph, if not strictly E-GAP) – the complete graph arises because agents in extreme teams have similar functionality. Dense graphs are problematic for DCOP algorithms[6, 1], so a novel technique is required. For each value, we create a *token*. Only the team member currently holding a token representing a value can assign that value to its variable. If the team member does not assign the value to its variable, it passes the token to a team mate who then has the opportunity to assign the value represented by the token. Essentially, tokens deliberately reduce DCOP parallelism in a controlled manner. Thus, the agents do not need to communicate to resolve conflicts.

Given the token-based access to values, the decision for the agent becomes whether to assign values represented by tokens it currently has to its variable or to pass the tokens on. First 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 team mates in the form of a token(s) (Alg. 1, lines 20 and 22). The agent keeps values that maximize the use of its capabilities (performed in the MAXCAP function, Alg. 1, line 16). Notice that changing values corresponds to changing roles and may not be without cost. Also notice that the agent is "greedy" in that it performs the roles it is best at.

ALGORITHM 1: VARMONITOR(Cap, $Resources$)

```
(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 = COMPUTETHRESHOLD(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(Values)
(17)          foreach v ∈ out
(18)            if v.potential
(19)              SENDMSG(pv.owner, "released")
(20)              PASSON(new token(pv, potential))
(21)            else
(22)              PASSON(new token(v))
(23)              Values ← Values − out

(25)      else
(26)        PASSON(token) /* Cap < threshold */
(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, "release")
(33)    else if msg is "release v"
(34)      PV ← PV − v
```

Secondly, 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). The key question is whether passing the token on will lead to a more capable team member taking on the role. Using probabilistic models of the members of the team and the roles that need to be assigned, the team member can choose the mini-

mum capability the agent should have in order to assign the value. Notice that it is the similar functionality of the agents in extreme teams, and their large numbers that allows us to apply probabilistic models. Intuitively, the agent estimates the likely capability of an agent performing this role in a good allocation. We refer to this minimum capability as the *threshold*. The threshold is calculated once (Alg 1, line 7), and attached to the token as it moves around the team. Thus, the agents must simply circulate tokens until each token is held by an agent with capability above threshold for the role and within resource constraints. (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.)

AND Constrained Roles

When there are AND constraints between roles there is the potential for *deadlocks* or, at best, severe inefficiencies. Consider a plan that requires two roles, r_j and r_k, to be simultaneously performed. When a team member, a, accepts role r_j, it may reject other roles that it could potentially perform. If there is no team member currently available to perform role r_k, a must simply wait.

```
ALGORITHM 2: ANDMONITOR(V)
(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")
```

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 roles 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 role represented by the token (Alg 1, line 10), only if a potential token for each of the other real tokens is accepted and may perform other roles in the meantime.

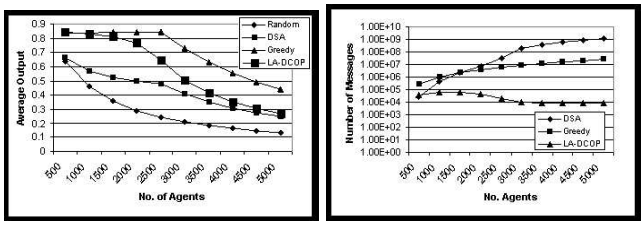
When the team member holding the real tokens is informed that at least one potential token for each real token has been accepted by a team member it *locks* the group by selecting the holder of one potential token for each real token and sending them the real token (Alg 2, line 15). A list of agents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

4 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[8]. In the simulator, agents are randomly given capabilities for each type of role with some percentage being given zero capability. Given many agents with overlapping capabilities for role types, dense constraint graphs result, where a constraint ensures two agents do not take the same role. For each time step that the agent has the role, the team receives ongoing reward based on the agent's capability. Message passing is simulated as taking one time step and messages always get through. New roles appear spontaneously and the corresponding tokens are distributed randomly. The new roles appear at the same rate that old roles disappear, hence keeping the total number of roles constant. Each data point represents the average from 20 runs.

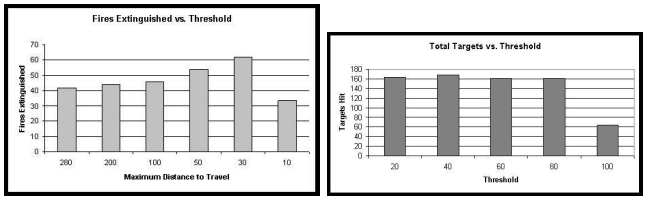
Our first experiments tests LA-DCOP against three competitors. The first is DSA, which is shown to outperform other approximate DCOP algorithms in a range of settings [6, 1]; we choose optimal parameters for DSA [15]. DSA does not easily allow multiple roles to be assigned to a single agent so our comparison focuses on the case where each agent can take only one role. As a baseline we also compare against a centralized algorithm that uses a "greedy" assignment (similar to the one presented in [14]) and against a random assignment. Figure 1(a) shows the relative performance of each algorithm. The experiment used 2000 roles over 1000 time steps. The y-axis shows the total reward per agent, while the x-axis shows the number of agents. Not surprisingly, the centralized algorithm performs best and the random algorithm performs worst. LA-DCOP is statistically better than DSA. However, the key 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 three orders of magnitude fewer messages than the greedy algorithm and four orders of magnitude less messages than DSA. Thus, LA-DCOP 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.

A key feature of extreme teams domains is that the roles to be assigned change rapidly and unpredictably. In Figure 2a, we show that LA-DCOP performs well even when



(a) **(b)**

Figure 1. (a) comparing the average output per agent versus the number of agents. (b) the number of messages sent versus the number of agents

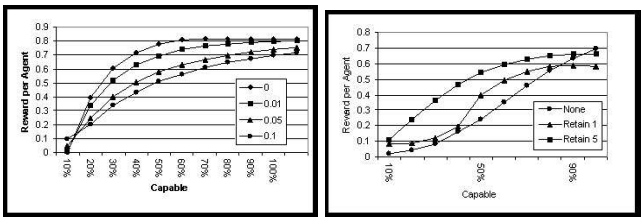


(a) **(b)**

Figure 3. (a) shows the number of fires extinguished by 200 fire trucks versus threshold (b) shows the number of targets hit by UAVs versus threshold.

the 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 roles are replaced with roles requiring a different capability. At middling capability (50%), with a 1% dynamics, LA-DCOP loses 10% of reward/agent on average, but complete DCOP algorithms today cannot even handle dynamics.

Finally, Figure 2b shows the utility of the use of potential tokens when groups of roles are AND constrained. In the figure, 60% of all roles (900 roles) are AND constrained into groups of five roles. Unless an agent with non-zero capability is assigned to each role 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 have non-zero capability because this is the case when most deadlocks and idleness occur.



(a) **(b)**

Figure 2. (a) shows the effects of different proportions of roles changing each step. The y-axis shows the output, x-axis shows the percentage of agents with capability > 0. (b) shows the effect of retainers, with the lines representing no retainers, one retained role per agent and five retained roles per agent.

In our second set of experiments, we used 200 LA-DCOP enhanced versions of Machinetta proxies[11], distributed over a network, executing plans in two simple simulation environments. To the best of our knowledge, this is larger than any published report on complex multiagent teams, certainly an order of magnitude jump over the last published reports of teamwork based on proxies [11]. 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 4(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 roles and a decrease in performance. In the second domain, 200 simulated unmanned aerial vehicles (UAVs) explored a battle space, destroying targets of interest. While in this domain LA-DCOP effectively allocates roles 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 [3]. RoboCup Rescue provides an ideal, realistic testing ground for LA-DCOP in allocating roles to an extreme team comprised of fire engines.

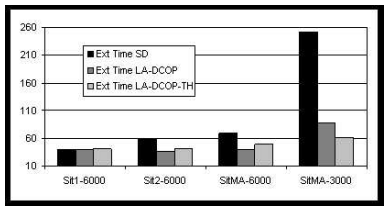


Figure 4. LA-DCOP fights fires twice as fast as SD

In previous work, researchers have documented the failure of auction based algorithms for role allocation in RoboCup Rescue[7], due to the high communication requirements. To test whether LA-DCOP can allocate roles within the communication and time limitation of RoboCup Rescue we compared against a shortest distance based strategy, which exploits domain characteristics, similar to top-performing RoboCup Rescue teams. We tested the LA-DCOP with and without the use of thresholds. Agents capabilities are computed considering whether the agent is blocked or not and its current distance from the fire.

We compared the strategies on three different initial situations: i) situation 1 has 10 agents and 18 ignition points uniformly distributed. ii) situation 2 has 10 agents and 18 ignition points but fire brigades are concentrated in two regions of the map and fires are concentrated in a third region. iii) situation 3 has 15 agents and 27 ignition points, fires are concentrated in two regions of the map while agents are positioned in other three regions. Notice that situation 2 and situation 3 are quite realistic.

The results are shown in Figure 4. The x-axis shows the different situations; in the fourth case we have lowered the extinguishing power of each truck. The y-axis shows the extinguish time for LA-DCOP without using threshold, using threshold LA-DCOP-TH (threshold is set to 300 meters), and shortest distance strategy SD. The graph shows that for the first situation when fires and fire brigades are nicely spread all over the city map, the SD allocation have similar performance with respect to LA-DCOP and LA-DCOP-TH. However, In the other two scenarios, where fires and fire fighters are not uniformly spread, the LA-DCOP and LA-DCOP-TH are at least twice as fast in extinguishing fires; and when extinguishing power is diminished, LA-DCOP is 10 times faster.

5 Summary and Related Work

Coordination for extreme teams in OCS domains poses novel and challenging issues. In particular, the system should be able to adapt to the continuous changing environmental conditions and to work with a dynamic network infrastructure while ensuring global coherence of the mission to be achieved. In this paper, we have described a novel and flexible approach to role allocation in extreme teams that can successfully operate in OCS domains. Our DCOP based approach was shown to substantially out-perform other approximate DCOP algorithms, reducing the number of messages by up to four orders of magnitude, while handling additional challenges of extreme team. We showed the effectiveness of LA-DCOP by testing it in three distinct domains with teams an order of magnitude bigger than previously published. An important key to the algorithm was the use of probabilistic models about the state of the team.

Role allocation is an extensively studied area with work ranging from high complexity, forward looking optimal models[7] to symbolic matching that ignores cost[13] to centralized auctions[2]. However, none of these approaches deal with full range of issues seen in E-GAP.

A number of approaches have been developed specifically for dynamic teams: SPAM allocates resources to tasks using a heuristic DCOP algorithm[4]. When applied to distributed sensor nets, SPAM has been shown to be effective in sparse constraint graphs. It thus compliments our work, which focuses on more dense graphs. Other role allocation techniques have been developed for swarm-like groups[10], but these techniques use only local sensing and are not applicable to widely distributed teams.

As future works, an important step would be to test how our approach behaves when the team connection topology changes dynamically during mission execution and when messages can get lost due to communication errors.

References

[1] S. Fitzpatrick and L. 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.

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

[3] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, , and H. Matsubara. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.

[4] R. Mailler, V. Lesser, and B. Horling. Cooperative negotiation for soft real-time distributed resource allocation. In *Proceedings of AAMAS'03*, 2003.

[5] M. Mamei, F. Zambonelli, and L. Leonardi. Developing adaptive and context-aware applications in dynamic network. In *Proceedings of WET ICE 2003*, pages 401–406, June 2003.

[6] P. J. Modi, W. Shen, and M. Tambe. Distributed constraint optimization and its application. Technical Report ISI-TR-509, University of Southern California/Information Sciences Institute, 2002.

[7] 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.

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

[9] A. Ricci and A. Omicini. Supporting coordination in open computational systems with tucson. In *Proceedings of WET ICE 2003*, pages 365– 370, June 2003.

[10] P. Sander, D. Peleshchuk, and B. Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of AAMAS'02*, 2002.

[11] P. Scerri, D. V. Pynadath, L. Johnson, R. P., N. Schurr, M. Si, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *In Proceedings of AAMAS*, 2003.

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

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

[14] B. B. Werger and M. J. Mataric. Broadcast of local eligibility for multi-target observation. In *Proc. of 5th Int. Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2000.

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