36

Task Assignment with Dynamic Token Generation

Alessandro Farinelli, Luca Iocchi, Daniele Nardi, and Fabio Patrizi

University of Rome La Sapienza,
Dipartimento di Informatica e Sistemistica
Via Salaria 113, Rome, Italy
lastname@dis.uniroma1.it

Summary. The problem of assigning tasks to a group of agents acting in a dynamic environment is a fundamental issue for a MAS and is relevant to several real world applications. Several techniques have been studied to address this problem, however when the system needs to scale up with size, communication quickly becomes an important issue to address; moreover, in several applications tasks to be assigned are dynamically evolving and perceived by agents during mission execution. In this paper we present a distributed task assignment approach that ensure very low communication overhead and is able to manage dynamic task creation. The basic idea of our approach is to use tokens to represent tasks to be executed, each team member creates, executes and propagates tokens based on its current knowledge of the situation. We test and evaluate our approach by means of experiments using the RoboCup Rescue simulator.

36.1 Introduction

The problem of assigning tasks to a group of agents or robots acting in a dynamic environment is a fundamental issue for Multi Agent Systems (MAS) and Multi Robot Systems (MRS) and is relevant to several real world applications. Many techniques have been studied to address this problem in different scenarios, providing solutions that in different ways approximate the optimal solution of the Generalized Assignment Problem (GAP), which consists in assigning a predefined set of tasks (or roles) to a set of agents maximizing an overall utility function that takes into account the capabilities of all the agents in the team.

While GAP requires the definition of a static set of tasks, that must thus be known in advance, in many application domains, tasks to be accomplished are not known a priori, but are discovered dynamically during the execution of the mission. Furthermore, when the system needs to scale up with size, communication quickly becomes an important issue to address.

The problem of dynamic task assignment has been studied and experimented by many researchers both in MRS (e.g. [3, 16, 10]) and in MAS (e.g. [6, 4, 7, 13]) communities. Several different aspects of the problem have been investigated and several approaches proposed. However, the growing complexity of missions in which

robots and agents are involved pushes toward the development of novel solutions for task assignment, which are able to address the more challenging issues posed by the applications. For example, auction based approaches to task assignment, have been proved to fail in the RoboCup Rescue domain, due to high communication requirements [8].

In this paper we present a distributed task assignment approach that is able to dynamically discover new tasks to be accomplished according to the situation perceived by the agents during the execution of their activities, and to ensure very low communication overhead. We focus on task assignment for teams operating in environments that need to meet (soft) real time constraints in their mission execution, where agents involved have similar functionalities but possibly varied capabilities. The reference scenario we are interested in has the following characteristics: i) the domain and the number of agents involved pose strict constraints on communications; ii) agents may perform one or more tasks, but within resource limits; iii) too many agents fulfilling the same task lead to conflicts that needs to be avoided; iv) tasks are discovered during mission execution.

The basic idea of our approach is derived from previous work based on token passing [12]. Tokens are used to represent tasks that must be executed by the agents, and each team member creates, executes and propagates these tokens based on its knowledge of the environment. The basic approach is based on the assumption that one token is associated to every task to be executed and that the token is maintained by the agent that is performing such a task, or passed to another agent if the agent that has the token is not in the condition of performing it.

In the case of dynamic discovery of the tasks to be performed and thus of dynamic token generation, the token passing approach must be appropriately extended in order to limit the number of tokens associated to the same task. Indeed, in our reference scenario optimal performance is obtained when there is a limited number of agents cooperating to execute the same task; when too many agents operate on a single task the overall performance decreases, since they ignore other tasks that evolve in a dynamic environment. The algorithm presented in this paper allows every agent to generate tokens dynamically whenever a task to be accomplished is perceived, while limiting the number of tokens associated to the same task and minimizing the bandwidth (i.e. communication messages among agents) required.

We test and evaluate our approach by means of experiments on a simulated scenario, that models a team of fire-fighters engaged in fighting fires in a city. To this end, we use the RoboCup Rescue simulator, that models the evolution of fires in the buildings of a city, city traffic, fire-fighters actions of extinguishing fires and communication among them. In this scenario, the location of the fires are not known a priori and the fire-fighter agents find them during their activities; in addition fires may unpredictably spread over adjacent buildings if not extinguished in time. Moreover, communication constraints are very strict, since messages are both limited and costly (in terms of simulation time steps).

The results that are reported in this paper show that the proposed extension of the token passing approach provides good performance in this scenario, while maintaining a very low communication bandwidth and thus significantly increasing the scala-

bility of the system. Therefore, the proposed approach is specifically well-suited for large scale teams operating in dynamic environment, as compared to other dynamic task assignment methods that require a wider communication bandwidth.

36.2 Problem Definition

The definition of the problem considered in this paper is derived from the GAP problem [14], which consists in assigning a set of tasks (or roles) $R = \{r_1 \dots r_m\}$ to a set of agents (or entities) $E = \{e_1 \dots e_n\}$ with different capabilities for each task $Cap(e_i, r_j) \in [0, 1]$ (i.e. a reward for the team when agent e_i performs task r_j), different resources needed by the agents for performing each task $Resources(e_i, r_j)$, and the resources available for an agent $e_i.resources$. An allocation matrix A is used for establishing task assignment: $a_{i,j} = 1$ if and only if the agent e_i is assigned to task r_j. The goal for the GAP problem is to find such an allocation matrix, that maximizes the overall capability function:

$$f(A) = \sum_i \sum_j Cap(e_i, r_j) \times a_{i,j}$$

subject to:

$$\forall i \sum_j Resources(e_i, r_j) \times a_{i,j} \leq e_i.resources$$

$$\forall j \sum_i a_{i,j} \leq 1$$

For example, in the rescue scenario that we have considered in our experiments, tasks are fires to be extinguished and agents are fire fighter brigades. The capability of a fire fighter to extinguish a fire, maybe dependent on several parameters, however a good approximation could be to consider the capability as a function of distance from the fire; clearly, if the nearest fire fighter is allocated to each fire the team gain a reward in terms of total traveled distance and time to extinguish all the fires. Resources are represented by the amount of water needed to put out fires.

The above formulation is well defined for a static environment, where agents and tasks are fixed and capabilities and resources do not depend on time. However, in several applications it is useful or even necessary to solve a similar problem where the defined parameters changes with time.

For example, in the above mentioned rescue scenario, all the defined parameters clearly depends on time, (e.g. fire fighters capabilities are strongly dependent on the environment evolution). Indeed several methods for dynamic task assignment implicitly take into consideration such an aspect, providing solutions that consider the dynamics of the world and derive a task allocation that approximate solutions of the GAP problem at each time steps (see for example [3, 16, 10, 8]).

The method described in this paper follows the line described above, and aims at solving the GAP problem when the set of tasks R is not known a priori when the mission starts, but it is discovered and dynamically updated during tasks execution.

To describe our method we will use the following notation. We denote that the set R depends on time with $R(t) = \{r_1 \dots r_{m(t)}\}$, where $m(t)$ is the number of tasks considered at time t, and we express the capabilities and the resources depending on time with $Cap(e_i, r_j, t)$, $Resources(e_i, r_j, t)$, and $e_i.resources(t)$. The dynamic allocation matrix is denoted by A_t, in which $a_{i,j,t} = 1$ if and only if the agent e_i is assigned to task r_j at time t. Consequently, the problem definition is to find a dynamic allocation matrix that maximizes the following function

$$f(A_t) = \sum_t \sum_i \sum_{j=1}^{m(t)} Cap(e_i, r_j, t) \times a_{i,j,t}$$

subject to:

$$\forall t \forall i \sum_{j=1}^{m(t)} Resources(e_i, r_j, t) \times a_{i,j,t} \leq e_i.resources(t)$$

$$\forall t \forall j \in \{0, \dots, m(t)\} \sum_i a_{i,j,t} \leq 1$$

36.3 Token Generation for Tasks Allocation

The main idea of the token passing approach is to regulate access to tasks execution through the use of tokens, i.e. only the agent currently holding the token can execute the task. Following this approach the communication needed to guarantee that each task is performed by one agent at time is dramatically reduced (see [2]).

If a task can benefit from the simultaneous execution of several agents, we can decide to create several tokens referring to the same task. However, when tokens are generated and perceived by agents during mission execution conflicts on tasks may arise. In this paper we will deal with two kinds of conflicts: the first one is due to the fact that the same task can be perceived by several agents during the missions, and if no explicit procedure is used the allocation process has no control on the maximum number of agents operating on such a task; this can lead to a consistent waste of resources and result in poor performance. The second type of conflict arises when an agent accomplishes a task and other tokens referring to the same task are still active, causing agents to waste precious time in trying to accomplishing terminated tasks.

We explicitly address these problems by proposing an extension to the algorithm presented in [2]. In the following, a *Task* refers to the physical object or event that the agent perceives and that implies an activity to be executed (e.g. a fire to be extinguished), therefore given a perceived object o we define the related task $T(o)$; a *Token* comprises the physical object related to the task and an identification number, that identifies different tokens for the same task, therefore given a task $T(o)$ we

may have a number s of tokens $TK(o,1)...TK(o,s)$. The main idea of the proposed algorithm is that when an agent perceives a task, it records this information in a local structure and announces the presence of the task to all its team mates. Only the agent that first perceives a new task (e.g. a fire) creates one ore more tokens for it; conflicts that might arise due to simultaneous perception are addressed and solved as explained later. Whenever, an agent accomplishes a task it announces to the entire team the task termination, and each of the team members removes the tokens referring to the accomplished task from their local structures.

Using this approach conflicting tokens can still be created for two main reasons: i) **Contemporary task discovery**: two agents e_1 and e_2 perceive a new task t, creating a set of tokens $Tk(t,1)...Tk(t,s)$ exactly at the same time, such that both agents will have different tokens referring to the same task. ii) **Messages asynchrony** Assume we have three agents e_1, e_2, e_3; if e_1 immediately after the creation of a new set of tokens $Tk(t,1)...Tk(t,s)$ decide to send one of them, say $Tk(t,j)$, to agent e_3, this token will not be found in the local structure of e_1 when the announce messages of e_2 arrives and therefore will not be deleted; for e_3 we can have two situations: a) the message referring to token $Tk(t,j)$ arrives before the announce message of e_2 b) the announce message of e_2 arrives before the message referring to token $Tk(t,j)$. In both these situations the token $Tk(t,j)$ will not be deleted, and the conflict will not be solved. Both these problems have been addressed and solved in our approach as explained later in this section.

In the algorithm the following data structures are used: i) Known Tasks Set (KTS) is a set containing at each time step all the tasks that has been perceived by all the agents; ii) Token Set (TkS) is the set of tokens each agent currently holds; iii) Temporary Token Set (TmpTkS) is a set containing the tokens created by the agent in the current time step; iv) Accomplished Tasks Set (ATS) is a set containing at each time step all the tasks that have been accomplished by all the agents each of this data structure is local to one agent. v) A message has three fields: $type \in \{announce, accomplishedTask, token\}$, *task* that contains information about the perceived task (e.g. fire position), valid when *type* is *announce* or $accomplishedTask$; finally the *token* field is valid only when the message is of type *token* and contains information about the token (e.g. task position, Id number, visited agents etc.); whenever an agent detects a new task through its perception it adds the new task to the KTS, creates s tokens referring to the task and adds them in the TmpTkS, then it *Announce* the new task to all its team members (Algorithm 1 OnPercReceived). Each team member when accomplishes a task sends an accomplished message to all its team mates and update its ATS (Algorithm 1 OnTaskAccomplishment). Each team member when receiving a message updates its local structures as explained in Algorithm 1, OnMsgReceived. Whenever a task is perceived, a new token is generated only if that task is not present in the KTS. After tokens have been processed (Algorithm 1, TokenManagement) the $TmpTkS$ is copied in the TkS . Assuming that messages cannot get lost, Algorithm 1 guarantees that when an agent a perceives a task T, that has already been discovered before (i.e. that is present in the KTS), it will not create new tokens for it, correctly assuming that someone else already has the token(s) for T.

Algorithm 1: Procedures for on line token generation

ONPERCRECEIVED($task$)
(1) **if** ($task \notin KTS$)
(2) $KTS = KTS \cup task$
(3) $TmpTkS = TmpTkS \cup T(task, 1) \cup ... \cup T(task, s)$
(4) SEND(Msg(Announce,$task$))

ONMSGRECEIVED(Msg)
(1) **if** $Msg.type == AccomplishedTask$
(2) $ATS = ATS \cup Msg.task$
(3) **if** $Msg.type == Announce$
(4) **if** ($Msg.task \notin KTS$)
(5) $KTS = KTS \cup \{Msg.task\}$
(6) **else**
(7) **if** $Msg.senderId \geq MyId$
(8) $TmpTkS = TmpTkS \setminus \{T | \forall j T(Msg.task, j)\}$
(9) **if** $CurrentTask == Msg.task$
(10) STOPCURRENTTASK()
(11) **if** $Msg.type == Token$
(12) $TkS = TkS \cup Msg.Token$

ONTASKACCOMPLISHMENT($task$)
(1) $ATS = ATS \cup task$
(2) SEND(Msg(AccomplishedTask,$task$))

TOKENMANAGEMENT()
(1) $TkS = TkS \setminus ATS$
(2) $TokenSet$ = CHOOSETOKENSET(TkS)
(3) $SendTokenSet = TkS \setminus TokenSet$
(4) SEND(Msg(Token,$SendTokenSet$))
(5) $TkS = TkS \cup TmpTkSet$
(6) STARTTASK(CHOOSETASK(TokenSet))

Notice that OnPercReceived, OnMsgReceived and OnTaskAccomplishment are asynchronous procedures, triggered by particular events; theoretically all the possible interleaving of their execution could occur, however, if we assume that each procedure is atomic (which is a reasonable assumption since no synchronization among agents is involved), we can guarantee that there will never be two tokens referring to the same task in the system for a time longer than the time required for the *Announce* messages to reach all the team members. In fact, as explained above conflicting tokens may be created in case of **Contemporary task discovery** or due to **Messages asynchrony**.

The problem of **Contemporary task discovery** is considered and solved by procedure OnMsgReceived: when agents receive the announce messages the one with a lower static priority, represented in the procedure by the lower Id number, will delete the token for task t from $TmpTkS$, solving the conflict; if t is already being

executed by the agent with lower static priority, it will stop its execution yielding to the higher priority agent the possibility to execute the task. The problem arising due to **Messages asynchrony** is avoided thanks to the distinction between temporary tokens (stored in $TmpTkS$) and normal tokens (stored in TkS). In fact, assuming that the time needed for an announce message to reach all the agents is less than one simulation step (i.e. assuming that messages are synchronized with agents execution) the use of a Temporary Token Set guarantees that the conflicts will be detected and avoided. Otherwise, a higher communication overhead is needed in order to recover from such conflicts.

Setting a static fixed priority among agents can obviously result in non optimal behavior of the team, for example assuming that $Cap(e_1, r_j, t_i) > Cap(e_2, r_j, t_i)$ following the static priority, we yield to the less capable agent the access to the task r_j. However, while theoretically the difference among capabilities can be unbounded, generally, when tasks are discovered using perception capabilities agents perceive tasks when they are close to the object location, (e.g. if two fire fighters perceive the same fire their distance from the fire is comparable) and therefore the loss of performance due to the use of a fixed priority is limited.

Once a token has been created and added to the TkS the token-based access to values requires that each agent decides whether to execute the tasks represented by tokens it currently has or to pass the tokens on. The token management procedure of Algorithm 1 describes how tokens are processed: each agent erases from its TkS the accomplished task set ATS, then it chooses a set of tokens it can execute (ChooseTokenSet(TkS)). Each agent follows a greedy policy in this decision process, i.e. it tries to maximize its utility given the tokens it currently can access and its resource constraints. However, each agent in its decision consider whether it is in the best interest of the team for it to execute the tasks represented by its tokens. The key question is whether passing the token on will lead to a more capable team member taking on the token. Using probabilistic models of the members of the team and the tasks that need to be assigned, the team member can choose the minimum capability the agent should have in order to take on a token. Each agent sends the remaining tokens to its team mates, following a round robin policy and copies the $TmpTkS$ in the TkS. Finally, each agent chooses the best task (e.g. for fire fighters could be the nearest fire) among the $TokenSet$ it currently has (ChooseTask(TokenSet)) and starts the task execution.

36.4 Experiments and Results

We tested our task assignment approach in the RoboCup Rescue environment [5]. RoboCup Rescue provides an ideal simulation environment to test allocation strategies for team comprised of rescue agents. We focus on a real city map of Foligno in Italy [9], so as to test the performance of our approach in a realistic disaster rescue environment and where agents must navigate narrow streets and passages. Here, a team of fire brigades must fight fires in real-time, while facing the uncertainty of fire spreading and the dynamism that arises due to several factors: (i) agent has a limited

view of the world, and do not know in advance fires initial positions (ignition points); (ii) the way fires spread can not be precisely predicted; (iii) agents can be blocked in narrow passages.

To show that the algorithm presented in section 36.3 does actually avoid conflicts of both types, we implemented three different kinds of allocation strategies. The first strategy, referred to as Token Passing (TP), is a plain implementation of the token based approach algorithm, no announce procedure is used, but agents record in a Known Fire List the fires they perceive to avoid that different agents create two tokens for the same fire. This strategy does not enforce any constraint on the maximum number of agents simultaneously fighting the same fire. The second strategy, referred to as TP with Announce (TPA-n), makes use of the announce procedure to enforce that no more than n agents are simultaneously fighting the same fire, however this strategy does not address the second kind of conflict type, therefore situations in which agents can try to fight already extinguished fires may arise. The third strategy, referred to as TPA-n with AccomplishedTask (TPAA-n), makes use of the *announce* and *AccomplishedTask* messages, avoiding both types of conflicts.

In all the strategies the processing token procedure is the same and the capability to execute a task is computed considering the distance between the fire fighting agent and the fire, and whether the agent is blocked in a narrow passage. If an agent is blocked, it sends out the task it is currently executing and choose a different task from its set. The set of tasks to be executed is computed choosing the nearest fire f and keeping up to K fires whose distance from f is lower than a fixed Threshold T. The Threshold T and the number of tokens each agent can retain is statically defined, and is computed considering global information, such as the number of agents involved in the simulation and their distribution on the map. For a detailed discussion on how this static values can be computed we refer to [11].

We tested each strategy in different operative conditions, changing the extinguish power the fire fighting agents have. We start each simulation from the same initial configuration, comprised of 10 fire fighting agents and 18 ignition points distributed as shown in Figure 36.1;

In this experiments we assume that messages cannot be lost, and that their delay is not higher than a simulation step (i.e. agent execution is synchronized with message passing), moreover we set the number of tokens to be created for each task to be a fixed number (three in the performed experiments); while it is possible to dynamically change this number during mission execution depending on the environment situation, in these experiments we focus on studying how conflicts influence performance of fire fighting agents, leaving the problem of how many tokens would be needed for each task and how to deal with possible lost messages and unpredictable delays to later investigation.

We extracted from the performed experiments the *extinguish time*, as the time needed to put out all the fires, the number of *point to point messages* exchanged among agents per time step, the number of *broadcast messages* sent by agents per time step, the total *traveled distance* per agent and finally the total number of *conflicts*, as the number of times during the entire simulation that more than three agents have the same fire as target.

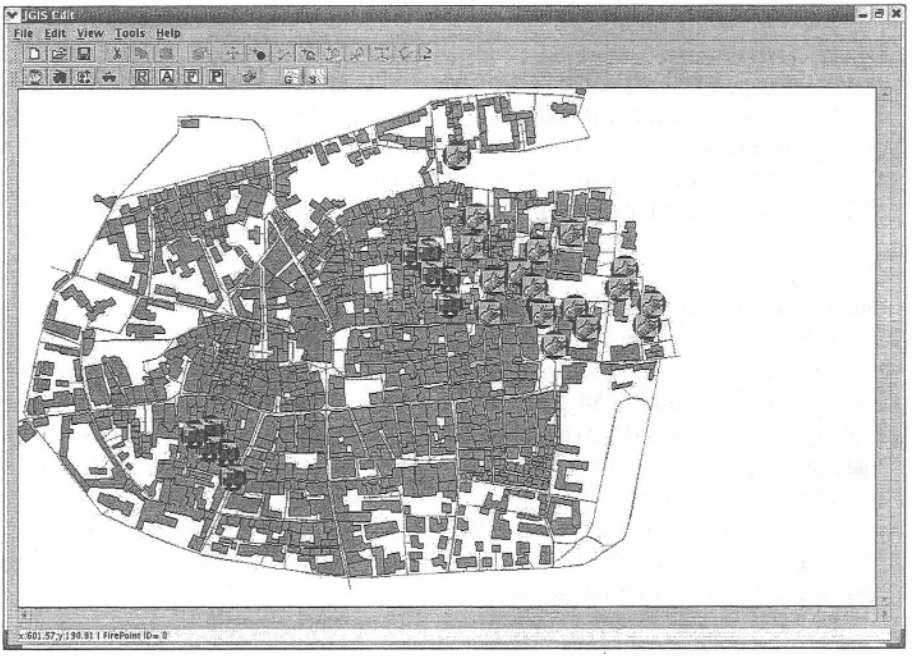

Fig. 36.1. Foligno Map used in the experiments

Table 36.1. Results obtained averaging 10 simulations

	TP	TPA-3	TPAA-3
Ext. Time	67 [0.7]	59.63 [16.6]	50.62 [2.5]
Ptp Msg per time step	1.4 [0.04]	1.8 [0.56]	1.7 [0.053]
Bcast Msg per time step	0 [0]	0.68 [0.63]	1.63 [0.13]
Trav. Dist. per agent	2495 [198]	3201 [527]	2221 [195]
Conflicts	26.62 [1.92]	0 [0]	0 [0]

In Table 36.1 we report results obtained from the simulations performed. Each reported value is the average obtained from ten repetitions of the simulation with the same operative conditions, along with the computed standard deviation (reported between brackets). From the table it is possible to see that the TPAA-3 strategy consistently outperform the TP strategy with a higher but still acceptable amount of messages. Moreover, the traveled distance for each agent is smaller on average, showing that better results are reached with a smaller waste of resources. The performance of TPA-3 strategy are on average in the middle with respect to TP and TPAA-3, however this strategy is characterized by a very high variance specially regarding the extinguish time and traveled distance. The high variance is due to the fact that the

strategy does not avoid the second type of conflicts, possibly generating consistent resource wasting.

In the performed experiments we have used values for extinguish power ranging from 6000 (water unit per minute) and up to model situations where it is useful that the agents allocation is balanced among the different tasks. Indeed, we found that the similar relationships among strategies hold increasing the extinguish power from 6000 (results reported in table 36.1) to 8000 and 10000.

36.5 Conclusions and Future works

Task allocation is a very widely studied area and several approaches have been presented in literature addressing different issues and techniques ranging from forward looking optimal model [8], to market or auction based techniques [16, 4], to symbolic matching [15] and Distributed Constrained Optimization Problem based algorithms [6]. However, the growing complexity of application for MAS and MRS requires novel solutions for task assignment, which are able to address specific features posed by the domain, such as dynamic tasks evolution, strict constraints on communication and soft real time constrained to be met.

Token based approach have been proved to be well suited for task allocation in such scenario [13, 1], however the specific problems of dynamic token generation and conflicts resolution have not been considered yet. In this paper we take a step in this direction proposing an extension to the token approach able to address this issue while keeping a reasonably low communication overhead. Moreover, we present first experimental results obtained for our approach, showing that it is actually applicable in a rescue scenario and is able to resolve conflicts improving the performances of the rescue teams.

Several other issues need to be further addressed, in particular we intend to test our algorithm with different types of rescue teams, such as ambulances or police force. The ambulance case is particularly interesting because it is important to enforce the constraint that only one agent can take care of a civilian, since no further benefit can be given to the team by having more than one ambulance trying to pick up a civilian, therefore we plan to further test our approach with ambulances. When dealing with different forces type constrained tasks comes into play, for example an ambulance agent could need to have a blocked road freed to pick up a civilian by a police agent, and an evaluation of our approach in such situation is particularly interesting. Finally, in our working scenario we assumed that no messages can be lost, this is quite a strong assumption, that can be easily violated in real world applications, therefore an interesting extension of our method will be devoted to explicitly deal with such situations.

Acknowledgment

This effort was partially funded by the U.S. Air Force European Office Of Scientific Research under grant number 033065 and by project "Simulation and Robotic

Systems for intervention in emergency scenarios" within program COFIN03 of the Italian MIUR, grant number 2003097252

References

1. A. Farinelli, P. Scerri, and M. Tambe. Allocating and reallocating roles in very large scale teams. In *First Int. Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster*, Padua, Italy, July 2003.
2. A. Farinelli, P. Scerri, and M. Tambe. Building large-scale robot systems: Distributed role assignment in dynamic, uncertain domains. In *Representation and approaches for time-critical decentralized resources/role/task allocation (AAMAS WorkShop)*, 2003.
3. B. Gerkey and J. M. Matarić. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA'03)*, Taipei, Taiwan, Sep 14 - 19 2003.
4. L. Hunsberger and B. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 151–158, 2000.
5. 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.
6. Roger Mailler, Victor Lesser, and Bryan Horling. Cooperative negotiation for soft real-time distributed resource allocation. In *Proceedings of AAMAS'03*, 2003.
7. P. J. Modi, H. Jung, W. Shen, M. Tambe, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Proc of Constraint Programming*, 2001.
8. 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.
9. D. Nardi, A. Biagetti, G. Colombo, L. Iocchi, and R. Zaccaria. Real-time planning and monitoring for search and rescue operations in large-scale disasters. Technical report, University "La Sapienza" Rome, 2002. `http://www.dis.uniroma1.it/~rescue/`.
10. L. E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.
11. P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating roles in extreme team. In *AAMAS 2004 (Poster)*, New York, USA, 2004.
12. P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Token approach for role allocation in extreme teams: analysis and experimental evaluation. In *13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004).*, Modena, Italy, 2004.
13. P. Scerri, D. V. Pynadath, L. Johnson, Rosenbloom P., N. Schurr, M Si, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *In Proceedings of AAMAS*, 2003.
14. D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.
15. G. Tidhar, A. S. Rao, and E. A. Sonenberg. Guided team selection. In *Proceedings of the Second International Conference on Multi-Agent Systems*, 1996.
16. R. Zlot, A Stenz, M. B. Dias, and S. Thayer. Multi robot exploration controlled by a market economy. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA'02)*, pages 3016–3023, Washington DC, May 2002.