Using Swarm-GAP for Distributed Task Allocation in Complex Scenarios

Paulo R. Ferreira Jr.[1,2], Felipe S. Boffo[1], and Ana L.C. Bazzan[1]

[1] Instituto de Informática, Universidade Federal do Rio Grande do Sul
Caixa Postal 15064, CEP 91501-970, Porto Alegre, RS, Brasil
{prferreiraj,fboffo,bazzan}@inf.ufrgs.br
[2] Instituto de Ciências Exatas e Tecnológicas, Centro Universitário Feevale
RS239, 2755, CEP 93352-000, Novo Hamburgo, RS, Brasil

Abstract. This paper addresses distributed task allocation in complex scenarios modeled using the distributed constraint optimization problem (DCOP) formalism. It is well known that DCOP, when used to model complex scenarios, generates problems with exponentially growing number of parameters. However, those scenarios are becoming ubiquitous in real-world applications. Therefore, approximate solutions are necessary. We propose and evaluate an algorithm for distributed task allocation. This algorithm, called Swarm-GAP, is based on theoretical models of division of labor in social insect colonies. It uses a probabilistic decision model. Swarm-GAP is experimented both in a scenario from RoboCup Rescue and an abstract simulation environment. We show that Swarm-GAP achieves similar results as other recent proposed algorithm with a reduction in communication and computation. Thus, our approach is highly scalable regarding both the number of agents and tasks.

1 Introduction

It is shown in [1] that optimal distributed coordination is associated with a high computational complexity, especially when agents lack full observability of the environment they operate. Even the less restrictive situations are proved to be NEXP-complete. However lack of full observability is a characteristic of many problems in the real world, where agents must reason with incomplete and uncertain information, in a timely fashion in order to cope with dynamic environments. Given these issues, Lesser [3] points out the fundamental principles for the construction of a multiagent system: agent flexibility. Flexibility would enable agents to react dynamically to the emerging state of the coordination effort.

When one exchanges centralized for distributed control, or trade total observability by local information, *self-organization* becomes key. To show good performance in realistic applications, multiagent systems must have a certain level of self-organization. Since the most natural way to organize work among agents is the decomposition of the objective in tasks, task allocation is an important part of the coordination problem. Research regarding multiagent systems

N. Jamali et al. (Eds.): CCMMS 2007, MMAS 2006, LSMAS 2006, LNAI 5043, pp. 107–121, 2008.

coordination through distributed task allocation has shown significant advances in the last few years. One successful direction, under the multiagent community perspective, has been the distributed constraint optimization problem (DCOP) framework.

Models of task allocation in complex environments, when modeled as a DCOP, yield hard problems, which cannot be treated with the traditional optimal/complete approaches to DCOP [5]. Thus complex DCOP scenarios introduce new challenges for the DCOP research. We see a complex scenario in distributed task allocation as the one in which, even small instances formalized as a DCOP generate large problems with exponentially growing number of parameters. Given that in the real-world we usually have large instances and these are dynamic, it is easy to conclude that new techniques are necessary.

We propose Swarm-GAP, an approximated algorithm for distributed task allocation based on theoretical models of division of labor in social insects colonies. This method is highly scalable regarding both the number of agents and tasks, and can solve the E-GAP (see next section) for dynamic task allocation in complex DCOP scenarios. Cooperative agents running our algorithm can coordinate their actions with low communication and computation.

We empirically evaluated the Swarm-GAP method on an abstract, domain-independent simulator, as well as in a scenario of the RoboCup Rescue simulator. Swarm-GAP is compared mainly to LA-DCOP [8], another approximated method for DCOP.

This paper is organized as follows: Section 2 discusses the use of the E-GAP model for task allocation in dynamic environments, and how it leads to a complex DCOP scenario. Section 3 presents our motivation to use swarm based heuristics and introduces the Swarm-GAP. The empirical evaluation of Swarm-GAP is shown in Section 4, together with a discussion on the results, while Section 5 presents our conclusions and future directions of this work.

2 Task Allocation Models and Complex DCOP Scenarios

In many real-world scenarios, a large number of agents must perform a large number of tasks. Besides, these tasks and their characteristics change over time and little information about the whole scenario, if any, is available to all agents. Each agent has different capabilities and limited resources to perform each task. The problem is how to find, in a distributed fashion, an appropriate tasks allocation that represents the best match among agents and tasks. This kind of scenario is becoming ubiquitous in manufacturing, robotics, computing, etc.

The Generalized Assignment Problem (GAP) deals with the assignment of tasks to agents, respecting agents resources, in order to maximize the total reward. GAP is known to be NP-complete [9]. It can be formalized as follows. Let us define $\mathcal{J}$ as the set of tasks to be allocated and $\mathcal{I}$ the set of agents. Each agent $i \in \mathcal{I}$ has a limited amount of resource r_i (a single type of resource is used). When a task $j \in \mathcal{J}$ is executed by agent i, task j consumes c_{ij} units of

i's resource. Each agent i also has a capability k_{ij} $(0 \leq k_{ij} \leq 1)$ to perform each task j.

The allocation matrix A, where a_{ij} is the value of the i-th row and j-th column, is given by Equation 1.

$$a_{ij} = \begin{cases} 1 \text{ if } j \text{ is allocated by } i \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

An optimum solution to the problem is given by matrix A^*, which maximizes the system reward as stated by Equation 2, subject to the agents resource limitations and the constraint of having only one agent allocated to each task.

$$A^* = \text{argmax}_{A'} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} k_{ij} * a'_{ij} \tag{2}$$

such that

$$\forall i \in \mathcal{I}, \ \sum_{j \in \mathcal{J}} c_{ij} * a_{ij} \leq r_i \ and \ \forall j \in \mathcal{J}, \ \sum_{i \in \mathcal{I}} a_{ij} \leq 1$$

The GAP was extended by [8] to capture dynamic domains and interdependence among tasks. This extension, called Extended-GAP (E-GAP), improves the model in two ways:

Allocation constraints among tasks. Tasks in E-GAP can be interrelated by an AND constraint. All interrelated tasks by this constraint must be allocated at the same time to be considered by the reward computation. Following [8], let us define $\bowtie \ = \{\alpha_1, ..., \alpha_p\}$, where $\alpha_k = \{j_{k_1}, ..., j_{k_q}\}$ denotes the k-th set of an AND constrained tasks. Thus, the partial reward w_{ij} for allocating task j to agent i is given by Equation 3.

$$w_{ij} = \begin{cases} k_{ij} * a_{ij} \text{ if } \forall \alpha_k \in \ \bowtie, \ j \notin \alpha_k \\ k_{ij} * a_{ij} \text{ if } \exists \alpha_k \in \ \bowtie \text{ with } j \in \alpha_k \wedge \\ \qquad\qquad \forall j_{k_u} \in \alpha_k, \ a_{xj_{k_u}} \neq 0 \\ 0 \qquad\quad \text{otherwise} \end{cases} \tag{3}$$

Reward dynamically computed over time. The total reward W is computed in E-GAP as the sum of the agents partial rewards (Eq. 3) in the last t time steps. In this case, the sequence of allocations over time is considered against the single allocation used in the GAP. Additionally, a delay cost d_j is used in order to punish the agents when task j was not allocated by time t. The objective of the E-GAP is to maximize the total reward W given by Equation 4.

$$W = \sum_t \sum_{i^t \in \mathcal{I}^t} \sum_{j^t \in \mathcal{J}^t} w_{ij}^t * a_{ij}^t - \sum_t \sum_{j^t \in \mathcal{J}^t} (1 - a_{ij}^t) * d_j^t \tag{4}$$

such that

$$\forall t \forall i^t \in \mathcal{I}^t, \quad \sum_{j^t \in \mathcal{J}^t} c_{ij}^t * a_{ij}^t \leq r_i^t \tag{5}$$

and

$$\forall t \forall j^t \in \mathcal{J}^t, \quad \sum_{i^t \in \mathcal{I}^t} a_{ij}^t \leq 1 \tag{6}$$

Several task allocation situations in large scale and dynamic scenarios can be modeled as an E-GAP [8]. Thus, the question now is how to find the best solution to E-GAP. The choice to model the E-GAP as a DCOP is mainly motivated by the recent advances in DCOP algorithms.

DCOP consists of n variables $V = \{x_1, x_2, ..., x_n\}$ that can assume values from a discrete domain $D_1, D_2, ..., D_n$ respectively. Each variable is assigned to one agent which has the control over its value. The goal of the agents is to choose values for the variables to optimize a global objective function. This function is described as the sum over a set of valued constraints related to pairs of variables. Thus, for a pair of variables x_k, x_l, there is a cost function defined as $f_{kl} : D_k \times D_l \rightarrow N$.

In DCOP, an E-GAP can be formalized as follows:

- Each variable $x_i \in V$ represents each agent i;
- Let us define a global domain D, whose elements are the set of all possible subsets of $\mathcal{J}$. The domain D_i of x_i is the set of elements from D, such that $\forall d \in D_i, \ \sum_{j \in d} c_{ij} \leq r_i$. This means that, to include d in D_i, the agent i must have enough resources to perform the entire task subset (each agent can allocate more than one task in E-GAP).
- The constraint cost function f_{kl}, related to the variables x_k and x_l, is given by the inverse of the sum of the reward obtained by each agent (Eq. 3). Besides, f_{kl} must prevent that more than one agent allocate the same task. Equation 7 defines f_{kl}.

$$f_{kl} = \begin{cases} -(\sum_{j \in D_k} w_{kj} + \sum_{j \in D_l} w_{lj}) \text{ if } a_{kj} \neq a_{lj} \\ \infty \qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \tag{7}$$

- There is one constraint to each pair of variables in V. We compute the cost as the inverse of reward because DCOP searches for minimizing the cost and E-GAP for maximizing the reward.

As we can see, an E-GAP formalized as a DCOP yields a large number of constraints, since there must be one for each pair of variables (a complete graph). The total number of required constraints can be computed as $\frac{n(n-1)}{2}$, where n is the number of agents (represented as variables). The size of variables' domain in the worst case, where agents have enough resources to allocate all tasks simultaneously, is $|\mathcal{P}(\mathcal{J})| = 2^{|\mathcal{J}|}$. The number of constraints grows exponentially

according to the number of agents, while the size of variables' domains grows exponentially according to the number of tasks.

An important question about all DCOP algorithms is whether they are fast enough to be applied in complex scenarios. This translates to whether the number and size of exchanged messages turns the approach feasible and efficient. In distributed approaches, communication among agents usually imposes demands that can cause network overload. Complex problems usually mean that the planning (for allocation) and action should be treated as quickly as possible. Most of the proposed approaches yields good results in simple scenarios, but there is a lack of analysis regarding complex ones.

Most complete algorithms for DCOP were tested in small scenarios like the MaxSAT 3-coloring problem. It is an interesting one as this problem can be considered a benchmark, but it is not enough to deal with more complicated problems. The largest and hardest scenario reported in the literature where the DCOP algorithms were applied are related to distributed meeting schedule (DMS) [4,7].

In [4], the authors analyze the performance of the Adopt algorithm with an instance of DMS problem with 47 variables, an 8-element domain and 123 constraints. It was shown that using Adopt, agents exchange about 750.000 messages to compute the solution. In [7] the DPOP was experimented with a DMS instance of 136 variables, an 8-element domain and 161 constraints. In this case, 132 messages were exchanged by the agents. According to the authors, the number of messages grows linearly according to the number of constraints. However, the size of messages grows exponentially and the time necessary to compute and send this messages are critic to the performance.

Complex scenarios, as we define, result in problems dramatically more hard then the ones cited above. Let us suppose an E-GAP scenario with 100 agents and 100 tasks. This is a small scenario if thinking in large scale (thousands of tasks and agents). The number of variables is equal to the number of agents. The total number of constraints can be computed, as we mentioned before, as $\frac{n(n-1)}{2}$ where n is the number of agents: 4950 for 100 agents. Assuming that each agent has, on average, enough resources to perform only 3 tasks simultaneously, the size of the variables domain is equal to the number of possible tasks' subsets (each with 1, 2 or 3 tasks), namely 166,750 elements. These figures are much higher than the ones related to the DMS problem. The amount of messages or their size as well as the computational effort in DCOP algorithms grow exponentially with those numbers. Thus, to deal with complex scenarios, it is necessary to minimize the communication (including the messages size) among the agents as much as possible. Besides, in this kind of problem, it is better to get an approximated solution as fast as possible than to find the optimal one in an unfeasible time.

Most DCOP algorithms have approximated it is possible to define an upper bound for the number or size of messages. However, normally the mechanisms used by those algorithms become very inefficient as the scale of a complex scenario grows. Since it is not possible to use the optimal algorithms, nor their

variants, we must look for other heuristic approaches, based on different mechanisms.

In [8], authors present an approximated algorithm called Low-communication Approximation DCOP (LA-DCOP) to solve instances of E-GAP. LA-DCOP outperforms DSA, another approximated algorithm able to deal with E-GAP, both regarding solutions' quality and number/size of messages. LA-DCOP uses a token based protocol to improve communication performance. Agents perceive a task in the environment, and either create a token to represent it or they receive a token from another agent. An agent decides whether to allocate a task based on a threshold and tries to maximize the use of its resources. After an agent decides whether or not to allocate a task, it sends the token to another randomly chosen agent.

3 Swarm-GAP

3.1 Motivation: Division of Labor in Swarms

Nature often rely on self-organization. There are several examples of self-organized biological systems. We focus here on the social insect colonies – also called swarms. A social insect colony with hundreds of thousand of members operates without any explicit coordination. An individual worker cannot assess the needs of the colony; it just has a fairly simple local information, and no one is in charge of coordination. From individual workers aggregation, the colony behavior emerges without any type of explicit coordination or planning. The key feature of this emergent behavior is the plasticity in division of labor inside the colony. Colonies respond to changing conditions by adjusting the ratios of individual workers engaged in the various tasks. Observations regarding this behavior are the basis of the theoretical model described in [11]. In this model, interactions among members of the colony and the individual perception of local needs result in a dynamic distribution of tasks. This model describes task distribution among individuals using the stimulus produced by tasks that need to be performed and an individual response threshold related to each task. The intensity of this stimulus can be associated with a pheromone concentration, a number of encounters among individuals performing the task, or any other quantitative cue sensed by individuals. An individual that perceives a task stimulus higher than its associated threshold, has a higher probability to perform this task.

Assuming the existence of $\mathcal{J}$ tasks to be performed, each task j has a s_j stimulus associated. $\mathcal{I}$ different individuals can perform them, each individual i having a response threshold θ_{ij} associated to a task j, according the task's type. Individual i engages in the task j with probability:

$$T_{\theta_{ij}}(s_j) = \frac{s_j^2}{s_j^2 + \theta_{ij}^2} \tag{8}$$

Each insect in the colony can potentially perform all types of tasks. However, it is possible for individuals to specialize in some type of tasks based on morphological aspects (polymorphism). Polymorphism plays a key rule to determine the division of labor in ant colonies. It is possible to capture the physical variety in the theoretical model by differentiating individual thresholds. The threshold θ_{ij} of the individual (i) for the task i decreases proportionally to the individual capability to perform these tasks $capability_i(j)$. Thus, individuals with large capability for a set of tasks have higher tendency to perform tasks of this set.

$$\theta_{ij} = 1 - capability_i(j) \tag{9}$$

where $capability_i(j)$ is the capability of individual i regarding to task j.

Social insects behavior seems to fit the requirements of complex problems since they are the result of millions of years of survival-of-the-fittest evolution.

3.2 Swarm-GAP Algorithm

The aim of Swarm-GAP is to allow agents to decide individually which task to execute in a simple and efficient way, minimizing computational and communication efforts. As in [8], we assume that the communication does not fail. In the future we intend to relax this assumption and perform tests with unreliable communication channels. Agents in Swarm-GAP decide which task to execute based on the same mechanism used by social insects as shown in Equation 8. Each task has the same associated stimulus, there is no priority on task allocation. The stimulus s is the same for every task j and its value was empirically determined to maximize the system reward, through direct experimentation. Swarm-GAP uses polymorphism to setup the agents thresholds according to agents capabilities. Equation 9 sets the agents threshold θ_{ij} as 1 minus the capability $capacity_i(j)$ of agent i to perform task j. This is so because threshold and capability are inversely proportional values.

Algorithm 1 details Swarm-GAP. Agents running Swarm-GAP communicate using a token based protocol, and react to two events: task perception and message arriving. When an agent perceives some tasks, it creates a token composed by these tasks (line 5), or it receives the token from another agent (line 10). Once this is done, the agent has the right to determine which tasks to allocate (lines 14 to 21) according to their tendency given by Equation 8. This decision also depends on whether the agent has the resource which is required to perform the task. The quantity of resource one agent has is decreased by the amount required by the task (line 19). Afterwards, the agent is marked as visited (line 23). This prevent deadlocks, since it avoids passing the token to agents that already received the token. At the end of this process, if the token still has available tasks, a token message is sent to an agent randomly selected among those agents which have not received the token in the current allocation (lines 24 to 29). The size of the token message is proportional to the number of tasks.

Algorithm 1. Swarm-GAP(*agentId*)

```
1: loop
2:    ev ← waitEvent()
3:    if ev = task perception then
4:       J ← set of new tasks
5:       token ← newToken()
6:       for all j ∈ J do
7:          token.addTask(j, -1)
8:    else
9:       token ← receiveToken()
10:
11:   r ← avaiableResources()
12:   τ ← token.avaiableTasks()
13:   for all t ∈ τ do
14:      θ_t ← 1− capability(t)
15:      if roulette() < T_{θ_t(s)} and r ≥ c_t then
16:         token.aloc(j, agentId)
17:         r ← r − c_t
18:
19:   token.visited(agentId)
20:   τ ← token.avaiableTasks()
21:   if |τ| > 0 then
22:      ι ← token.avaiableAgents()
23:      i ← rand(ι)
24:      sendToken(i)
```

4 Experiments and Results

Empirical evaluations of Swarm-GAP were conducted in an abstract, domain-independent simulator, and in the RoboCup Rescue Simulator [2,10].

The abstract simulator allows experimentation with a large number of agents and tasks. In general, we have 2000 tasks in each experiment, with 5 different classes randomly assigned to the tasks, and a variation in the number of agents from 500 to 4000 (that means, the latter is twice the number of tasks). Tasks cost are assigned uniformly from $\{0.25, 0.50, 0.75\}$. Each agent has 60% probability of having a non-zero capability for each class. Capabilities are uniformly assigned, with values ranging from 0 to 1. 60% of the tasks are AND inter-related in groups of 5. When new tasks arise at each simulation cycle they are randomly perceived by agents. The total number of tasks is kept constant, which means we modify the tasks characteristics (10% of the tasks has a probability of 10% to have their classes and demanded resources changed). At each simulation cycle, agents are constrained in the number of messages they can send to a maximum of 20.

Rewards are computed over 1000 simulation cycles, where in each cycle one allocation is performed. All data is averaged over 20 runs. Swarm-GAP is compared with three methods: LA-DCOP, a distributed greedy algorithm, and a centralized greedy algorithm. In the distributed greedy strategy, agents allocate

the local perceived tasks for which they have highest capability. The central greedy strategy, used to benchmark other methods, allocates the appearing tasks to the most capable available agent, i.e. it does this in a centralized way. This simulation has exactly the same setup used by [8] to experiment LA-DCOP and, according the authors, different values for these parameters do not affect the results.

The other scenario used is the RoboCup Rescue [2]. The goal of the RoboCup Rescue Simulation League is to provide a testbed for agent rescue teams acting in large urban disasters. RoboCup Rescue is a complex multiagent domain characterized by: agents' heterogeneity, restricted access to information, long-term planning, and real-time. In the RoboCup Rescue, each agent has perception limited to its surroundings. Agents are in contact by radio, but they are limited on the number of messages they can exchange. As a cooperative multiagent system, efficiency can be achieved with the correct coordination. Approaches based on communication are not appropriate, because of the high communication constraints. According to [6], the challenge imposed by the environment can be decomposed into a task allocation problem. The allocation sub problem that we use in this paper concerns to the efficient assignment of fire brigade agents to the task of extinguishing fire spots. Swarm-GAP is also compared with LA-DCOP in this case and both methods are benchmarked by a greedy algorithm. This algorithm considers a heuristic where brigade agents decide always to tackle their shortest distant fire spots.

4.1 Abstract Simulator

In the first experiment, the aim is to find the best stimulus value (Equation 8) that maximizes the reward when the number of agents changes regarding the number of tasks. In this case there are no tasks with AND constraints.

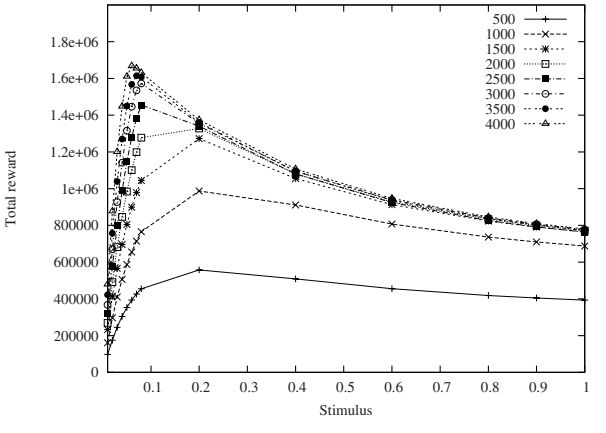

Fig. 1. Total reward *versus* task stimulus, for different number of agents

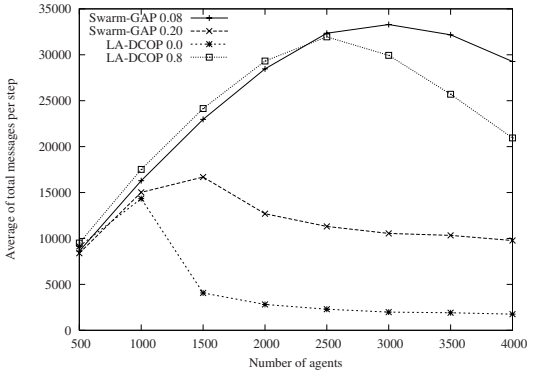

Fig. 2. Average number of messages per simulation cicle *versus* the number of agents

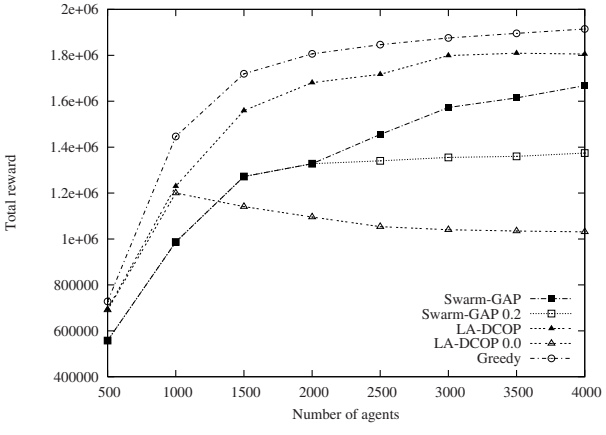

Fig. 3. Total reward for different number of agents

Figure 1 shows the rewards achieved for different values of stimulus and different number of agents. We use different quantities of agents to experiment different proportions related to the number of tasks (500 means 25% of the 2000 tasks, 1000 means 50%, and so on.)

In the second experiment, we measure the average number of messages per simulation cycle, according to the number of agents, for different setup parameters of Swarm-GAP (stimulus 0.2 and 0.08) and LA-DCOP (threshold 0.0 and 0.8). As we can see in Figure 2, when LA-DCOP works with threshold equal to zero, the number of messages exchanged is the smallest. Only on this case the average number of messages exchanged by Swarm-GAP is significantly greater than that of LA-DCOP. However, as we can see further on Figure 3, in this specific case, the total reward is significantly lower for LA-DCOP in comparison with Swarm-GAP. The total reward in E-GAP is computed as the sum of the agents capabilities to each task they allocated.

In the third experiment, we evaluate Swarm-GAP comparing its results with those achieved by the greedy centralized algorithm and LA-DCOP. Figure 3 shows the total reward, for different number of agents, achieved by Swarm-GAP, with the best stimulus for each number of agents and LA-DCOP, also with the best threshold for each number of agents. As expected, the greedy approach outperforms both Swarm-GAP and LA-DCOP. Swarm-GAP performs well achieving rewards that are only 20% lower (on average) than those achieved by the greedy, and 15% lower than those of LA-DCOP.

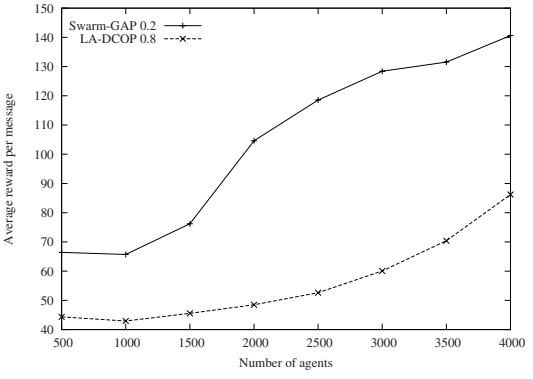

Fig. 4. Average reward divided by total number of exchanged messages, for different number of agents

To illustrate the advantages of Swarm-GAP related to LA-DCOP regarding the number of exchanged messages, Figure 4 shows the average reward divided by the total number of exchanged messages for different number of agents achieved by Swarm-GAP with stimulus 0.2 and LA-DCOP with threshold 0.8. We use this setup because in this case, both algorithms achieve to similar rewards. As we can see, Swarm-GAP exchanges a significant lower number of messages.

Furthermore, as Swarm-GAP uses a probabilistic decision process, the computation necessary for agents to make their decision is significantly lower than in LA-DCOP. As mentioned in Section 2, an agent running LA-DCOP chooses to allocate tasks which maximize the sum of its capabilities, while respecting its resource constraints. This is a maximization problem that can be reduced to a Binary Knapsack Problem (BKP), which is proved to be NP-complete. The computational complexity of LA-DCOP depends of the complexity of its function to deal with BKP. Each agent solves several instances of BKPs during a complete allocation. Agents running Swarm-GAP chooses to allocate tasks according to a probability computed by Equation 8, constrained by its available resources. This is a simple one-shot decision process.

Regarding the average number of allocated tasks per simulation step, both algorithms allocate almost the same number of tasks.

The last experiment with the abstract scenario measures the impact that AND-constrained tasks have over the reward. In this experiment, 60% of the

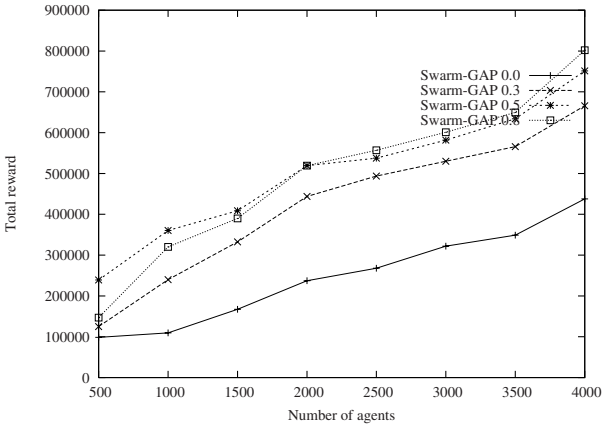

Fig. 5. Total reward for different number of agents in the presence of AND constrained tasks

tasks are AND constrained in groups of 5. Figure 5 shows the expected decrease in the performance of Swarm-GAP when we consider the AND constraints and several values for ω (Equation 8). Rewards improve when $\omega \neq 0$, improving the average performance in 25%.

4.2 Experiments on the RoboCup Rescue Simulator

In the following experiments, we evaluate Swarm-GAP comparing its results with those achieved by LA-DCOP. Both methods are benchmarked by a greedy approach based on a shortest distance heuristic.

We run our experiments in two partial maps of the Japanese city of Kobe (commonly used in RoboCup Rescue simulations), using 30 fire brigade agents in the first ($Kobe$) and twice this number in the second and largest map ($Kobe\_4$). For each map, we have tested two different scenarios for fire ignition points: 30 fires ($Kobe_1$), 42 fires ($Kobe_2$), 43 fires ($Kobe\_4_1$) and 59 fires ($Kobe\_4_2$). These points were uniformelly distributed. To measure performance, we use a metric which is the building area left (e.g. after an earthquake followed by a fire and the intervention of the fire brigades). Table 1 and 2 list results obtained in the above described scenarios. On the algorithm name column we also show the differents threshold values for LA-DCOP and stimulus value for Swarm-GAP that were experimented. All data is averaged over 20 runs of the simulator. Figure 6 depicts only the best performance for each method, for different scenarios.

To compute agents capabilities in the RoboCup Rescue we use the Euclidian distance from the fire brigade to the fire spot. Equation 10 show how this measure is normalized by agents.

$$capability(t) = \frac{\max\{distance(t_i), \forall t_i \in tasks\} - distance(t)}{\max\{distance(t_i), \forall t_i \in tasks\}} \qquad (10)$$

Table 1. Results from the *Kobe* map, showing the average building area left

	$Kobe_1$ (30 fires)	$Kobe_2$ (42 fires)
Swarm-GAP 0.1	$1.35 \times 10^7 \pm 0.08 \times 10^7$	$0.66 \times 10^7 \pm 0.89 \times 10^7$
Swarm-GAP 0.2	$1.25 \times 10^7 \pm 0.14 \times 10^7$	$0.65 \times 10^7 \pm 0.12 \times 10^7$
Swarm-GAP 0.3	$1.26 \times 10^7 \pm 0.14 \times 10^7$	$0.58 \times 10^7 \pm 0.12 \times 10^7$
Swarm-GAP 0.4	$1.13 \times 10^7 \pm 0.29 \times 10^7$	$0.57 \times 10^7 \pm 0.90 \times 10^7$
Swarm-GAP 0.5	$1.25 \times 10^7 \pm 0.18 \times 10^7$	$0.60 \times 10^7 \pm 0.11 \times 10^7$
Swarm-GAP 0.6	$1.12 \times 10^7 \pm 0.42 \times 10^7$	$0.59 \times 10^7 \pm 0.12 \times 10^7$
LA-DCOP 0.1	$1.04 \times 10^7 \pm 0.22 \times 10^7$	$0.45 \times 10^7 \pm 0.11 \times 10^7$
LA-DCOP 0.2	$1.09 \times 10^7 \pm 0.19 \times 10^7$	$0.50 \times 10^7 \pm 0.72 \times 10^7$
LA-DCOP 0.3	$1.08 \times 10^7 \pm 0.23 \times 10^7$	$0.51 \times 10^7 \pm 0.10 \times 10^7$
LA-DCOP 0.4	$0.97 \times 10^7 \pm 0.21 \times 10^7$	$0.51 \times 10^7 \pm 0.83 \times 10^7$
LA-DCOP 0.5	$0.77 \times 10^7 \pm 0.14 \times 10^7$	$0.48 \times 10^7 \pm 0.11 \times 10^7$
LA-DCOP 0.6	$0.70 \times 10^7 \pm 0.15 \times 10^7$	$0.46 \times 10^7 \pm 0.12 \times 10^7$
Shortest distance	$1.31 \times 10^7 \pm 0.10 \times 10^7$	$0.97 \times 10^7 \pm 14 \times 10^7$

Table 2. Results from the *Kobe_4* map, showing the average building area left

	$Kobe\_4_1$ (43 fires)	$Kobe\_4_2$ (59 fires)
Swarm-GAP 0.1	$4.80 \times 10^7 \pm 0.09 \times 10^7$	$2.04 \times 10^7 \pm 0.09 \times 10^7$
Swarm-GAP 0.2	$4.78 \times 10^7 \pm 0.04 \times 10^7$	$1.96 \times 10^7 \pm 0.11 \times 10^7$
Swarm-GAP 0.3	$4.77 \times 10^7 \pm 0.14 \times 10^7$	$1.89 \times 10^7 \pm 0.18 \times 10^7$
Swarm-GAP 0.4	$4.76 \times 10^7 \pm 0.10 \times 10^7$	$1.78 \times 10^7 \pm 0.12 \times 10^7$
Swarm-GAP 0.5	$4.75 \times 10^7 \pm 0.10 \times 10^7$	$1.80 \times 10^7 \pm 0.08 \times 10^7$
Swarm-GAP 0.6	$4.76 \times 10^7 \pm 0.08 \times 10^7$	$1.78 \times 10^7 \pm 0.11 \times 10^7$
LA-DCOP 0.1	$4.64 \times 10^7 \pm 0.14 \times 10^7$	$1.65 \times 10^7 \pm 0.09 \times 10^7$
LA-DCOP 0.2	$4.57 \times 10^7 \pm 0.18 \times 10^7$	$1.62 \times 10^7 \pm 0.08 \times 10^7$
LA-DCOP 0.3	$4.48 \times 10^7 \pm 0.24 \times 10^7$	$1.61 \times 10^7 \pm 0.09 \times 10^7$
LA-DCOP 0.4	$4.52 \times 10^7 \pm 0.17 \times 10^7$	$1.64 \times 10^7 \pm 0.10 \times 10^7$
LA-DCOP 0.5	$4.40 \times 10^7 \pm 0.23 \times 10^7$	$1.61 \times 10^7 \pm 0.10 \times 10^7$
LA-DCOP 0.6	$4.51 \times 10^7 \pm 0.03 \times 10^7$	$1.61 \times 10^7 \pm 0.13 \times 10^7$
Shortest distance	$4.78 \times 10^7 \pm 0.05 \times 10^7$	$2.39 \times 10^7 \pm 0.15 \times 10^7$

Swarm-GAP, in the first scenario of each map ($Kobe_1$ and $Kobe\_4_1$), outperforms other methods by a slight difference. These are a very simple situations with few fire spots and all methods perform almost equally well.

In the scenarios $Kobe_2$ and $Kobe\_4_2$, Swarm-GAP outperforms LA-DCOP, but not the shortest distance heuristic. When there are more fires and they get bigger, there is a demand for more agents concentrated in clusters. Here, the stochastic decision of Swarm-GAP agents keep them from performing a more concentrated effort, and make them waste more simulations cycles to go from one spot to another. When we increase task stimulus, Swarm-GAP performance get worse. In LA-DCOP, the agents simply reject tasks for which their capabilities are lower than the associated threshold. In LA-DCOP there is always the possibility that agents do not allocate any task during a time step, or keep from accepting distant tasks for a long time. Swarm-GAP agents can also allocate

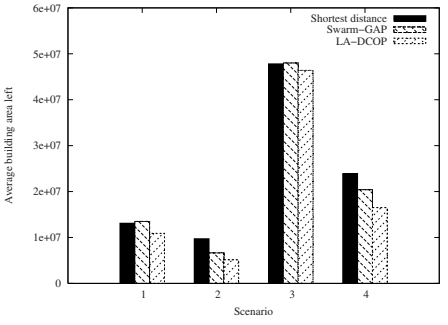

Fig. 6. Best performance of each method, for the scenarios $Kobe_1(1)$, $Kobe_2(2)$, $Kobe\_4_1(3)$, and $Kobe\_4_2(4)$

zero tasks during a time step or neglect far tasks, but when the number of tasks is high (like in scenario of $Kobe\_4_2$ map), this occurs with lower probability. In fact, in an overall average, LA-DCOP number of non-allocated tasks was almost 80% higher than Swarm-GAP.

Our results points that Swarm-GAP, by doing more flexible allocation choices, enable agents to better divide the tasks, sometimes outperforming LA-DCOP or performing equally well in comparison to the shortest distance in some situations.

5 Conclusions and Further Work

The approach introduced here – Swarm-GAP – deals with task allocation in complex scenarios modeled as DCOPs based on the theoretical models of division of labor in swarms. This algorithm solves complex DCOPs in an approximated and distributed fashion.

The experimental results show that the probabilistic decision, based on the tendency and polymorphism models, allows the agents to make reasonable coordinated actions. In the abstract simulation, Swarm-GAP performs well in comparison with LA-DCOP. However, by the nature of its mechanisms, Swarm-GAP uses less communication and computation than LA-DCOP.

Next, we intend to introduce failures in the simulation regarding the communication channel and agents task perception. This failures contribute to a realistic analysis of all algorithms. The idea is to evaluate our intuition that swarm like algorithms are able to deal with communication failures.

Acknowledgments

This research is partially supported by the Air Force Office of Scientific Research (AFOSR) (grant number FA9550-06-1-0517).

References

1. Goldman, C., Zilberstein, S.: Decentralized control of cooperative systems: Categorization and complexity analysis. Journal of Artificial Intelligence Research 22, 143–174 (2004)
2. Kitano, H.: Robocup rescue: A grand challenge for multi-agent systems. In: Proc. of the 4th International Conference on MultiAgent Systems, Boston, USA, pp. 5–12. IEEE Computer Society, Los Alamitos (2000)
3. Lesser, V.: Cooperative multiagent systems: A personal view of the state of the art. IEEE Transactions on Knowledge and Data Engineering 11(1), 133–142 (1999)
4. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, July 2004, vol. 1, pp. 310–317. IEEE Computer Society, Los Alamitos (2004)
5. Modi, P.J., Shen, W.-M., Tambe, M., Yokoo, M.: An asynchronous complete method for distributed constraint optimization. In: Proc. of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 161–168. ACM Press, New York (2003)
6. Nair, R., Ito, T., Tambe, M., Marsella, S.: Task allocation in the rescue simulation domain: A short note. In: Birk, A., Coradeschi, S., Tadokoro, S. (eds.) RoboCup 2001. LNCS (LNAI), vol. 2377, pp. 751–754. Springer, Heidelberg (2002)
7. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, August 2005, pp. 266–271. Professional Book Center (2005)
8. Scerri, P., Farinelli, A., Okamoto, S., Tambe, M.: Allocating tasks in extreme teams. In: Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 727–734. ACM Press, New York (2005)
9. Shmoys, D.B., Tardos, V.: An approximation algorithm for the generalized assignment problem. Mathematical Programming 62(3), 461–474 (1993)
10. Skinner, C., Barley, M.: Robocup rescue simulation competition: Status report. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 632–639. Springer, Heidelberg (2006)
11. Theraulaz, G., Bonabeau, E., Deneubourg, J.: Response threshold reinforcement and division of labour in insect societies. In: Royal Society of London Series B - Biological Sciences, vol. 265, pp. 327–332 (1998)