A Self-Organized Algorithm 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. We see a complex scenario in distributed task allocation as the one in which small instances formalized as a DCOP generate large problems with exponentially growing parameters. Such scenarios are becoming more and more ubiquitous in real-world applications. We propose and evaluate a novel self-organized algorithm for distributed task allocation based on theoretical models of division of labor in social insect colonies, called Swarm-GAP. Our algorithm uses a probabilistic decision model, based on the social insects tendency of performing certain tasks. Swarm-GAP was experimented in an abstract centralized simulation environment. We show that Swarm-GAP achieves similar results as other recent proposed algorithm with a dramatic reduction in communication and computation. Thus, our approach is highly scalable regarding both the number of agents and tasks.

1 Introduction

Coordination, a central issue in multiagent systems, is a process in which agents engage to ensure that a community of individual agents acts in a coherent manner. Theoretical results [2] show the high computational complexity of optimal distributed coordination, specially when agents lack full observability of the environment they operate. Even the less restrictive situations are proved to be NEXP-complete.

In complex environments of real world applications, agents must reason with incomplete and uncertain information, in a timely fashion in order to cope with dynamic environments. Generally, in such environments information is incomplete due to partial observability and communication constraints, or due to the dynamic nature of the environment itself. Given the elevate complexity of computing optimal solutions under these conditions, Lesser [4] points out the fundamental principles for the construction of a multiagent system: agent flexibility with respect to the availability, completeness and accuracy of its information

and the availability and capabilities of external resources, which enables agents to react dynamically to the emerging state of the coordination effort.

When we exchange centralized for distributed control or trade total observability by local information, *self-organization* becomes the key. To show good performance in realistic applications, multiagent systems must present a certain level of self-organization [11]. Among the needs behind this requirement, the most important are: fault-tolerance, self-configuration, ability to manage large collections of agents and resources towards an implicit defined collective goal, and adaptation for better performance.

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. The research regarding multiagent systems 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 the complex environments discussed above, when modeled as a DCOP, results in very hard problems,which cannot be treated with the traditional optimal/complete DCOP approaches [6], leading to a restrict applicability. These *complex DCOP scenarios* introduce new challenges for the DCOP research. We see a complex scenario in distributed task allocation as the one in which small instances formalized as a DCOP generate large problems with exponentially growing parameters. In the real-world we usually have large scale and dynamic complex scenarios.

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

A social insects colony is an example of a self-organizing biological system, where plenty of evidences of ecological success exist, despite the apparent lack of explicit coordination. In those colonies, hundreds of thousands insects adapt to the changes in the environment and to the needs of the colony using the plasticity in division of labor. There are well experimented theoretical models that capture this plasticity.

We empirically evaluated the Swarm-GAP method on an abstract, domain-independent simulator. Our swarm algorithm is compared mainly to LA-DCOP [8], an approximated method for DCOP that seems to outperform prominent contestants.

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, with a discussion on the achieved 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 which represents the best match among agents and tasks. This kind of scenario is becoming more and more ubiquitous in manufacturing, robotics, computing, etc.

The Generalized Assignment Problem (GAP) is a general allocation problem which examines the assignment of tasks to agents, respecting the agents resources, and maximizing a total reward. It is known to be NP-complete [9]. The GAP 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} $(0 < k_{ij} \leq 1)$ 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.

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

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 could be used in order to punish the agents if task j was not allocated by time t. The objective of the E-GAP is to maximize this 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

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

and

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

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 bring E-GAP to the DCOP framework, which has been used to formalize and solve distributed task allocation problems, 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 $f_{kl} : D_k \times D_l \rightarrow N$ [12]. We can view a DCOP as a graph where variables are their edges and constraints their vertices.

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}.

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

– There is one constraint to each pair of variables in V in order to make possible to DCOP algorithms to maximize the problem's total reward. 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, which means 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. We define a *complex scenario*, in terms of distributed task allocation using the DCOP framework, as any scenario with the characteristics discussed above.

An important question about all DCOP algorithms is whether they are fast enough to be applied in complex scenarios. An important issue here is whether the number and size of exchanged messages turns the approach feasible and efficient. In distributed approaches, the communication among agents usually imposes demands that can cause network overload. Is the total time consumed acceptable in these situations? 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.

When proposing DCOP algorithms the authors usually experiment them in small scenarios like the MaxSAT 3-coloring problem. It is quite interesting to validate their approaches as this problem can be considered a benchmark, but it is not enough to measure its abilities 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) [5, 3, 7].

In [5], the authors analyze the Adopt performance 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 [3] the OptAPO was experimented with a similar instance of DMS with 23 variables, an 8-element domain, and 16 constraints. The authors show that OptAPO was not able to find an optimal solution in a feasible time. 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 of DPOP, 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 as shown in [7].

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. The domain size could decrease if agents have no capability to perform some of the tasks, but we must emphasize that we are considering a small number of tasks. These figures are much more 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 are able to deal with approximated solutions. It is possible to define an upper bound for the number of messages in Adopt, as well as for the size of the messages in DPOP. However, this upper bound configurations introduced in the algorithms are not enough to deal with the complex scenarios. The mechanisms used by the algorithms become very inefficient as the scale of a complex scenario grows. Since it is not possible to use the optimal algorithms, nor its approximated variants, we must look for other heuristic approaches, based on different mechanisms.

In [8] the authors present an approximated algorithm to solve instances of E-GAP called Low-communication Approximation DCOP (LA-DCOP). LA-DCOP is an approximated DCOP algorithm developed to deal with the E-GAP special characteristics. LA-DCOP outperforms DSA, another approximated algorithm able to deal with E-GAP, both regarding solutions' quality and number/size of messages. DSA uses a hill-climbing strategy to allow the agents allocate tasks in order to maximize the reward based on neighbors' information. LA-DCOP uses a token based protocol to improve communication performance. Agents perceive a task in the environment, and 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.

The agents' threshold represents the agent capability to allocate each task and can be either a fixed value or dynamically computed. Additional tokens, called *potential tokens*, are used by the agents to make commitments regarding the allocation of AND constrained tasks.

3 Self-organization and Social Insects

A complex system is any system composed by elements interacting in many different ways, whose aggregate behavior is more than the summation of the individual elements' contribution. Many natural and artificial systems can be considered complex, and they are investigated by interdisciplinary researches in several areas of knowledge. Complex systems become organized by the emergence of patterns, achieved by their own internal process. In other words, they are self-organized. Self-organization refers to the process which new patterns or structures arise in the system from the interaction of their numerous elements, without interventions of external directing influences.

Nature often rely on self-organization. Biological systems lacking self-organization can be organized in many different ways, using well-informed leaders for instance. However, these alternatives demands high communication and cognitive abilities of their elements. Biological systems must be efficient in the physiological and behavioral needs to solve complex problems, just as to survive in our severe competitive environment. There are several examples of self-organized biological systems. We focus here on the social insect colonies – also called swarms.

3.1 Division of Labor in 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. The biological observations regarding the social insect individuals and colonies behavior led researchers to propose theoretical models inspired on this plasticity.

In [10] the authors present a model where interactions among members of the colony and the individual perception of local needs result in a dynamic distribution of tasks. This model describes the colony task distribution 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 (e.g. after walking around randomly) 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. The individual i engages in the task j performance 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 (morphological polyethism or simply 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.

The 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. These model have been applied in specific problems of distributed task allocation in the past [1], with relative success.

3.2 Swarm-GAP

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. The tendency of agent $i \in (I)$ to execute task $j \in (J)$ is given by its internal threshold θ_{ij}, the task stimulus s and the coefficient of execution x_j of task j, which is related to other tasks by an AND constraint, as shows Equation 10. The constant ω is used as a discount rate to decrease the weight of the coefficient of execution on the computation of the tendency.

$$T_{\theta_{ij}}(s) = \frac{s_j^2}{s_j^2 + \theta_{ij}^2} + \omega x_j \tag{10}$$

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. The execution coefficient x_j associated to the task j is computed using the rate between the number of allocated tasks $|n_j|$, and the total number of tasks $|N_j|$ which are related to task j by means of an AND relationship.

$$x_j = \begin{cases} \frac{1+|n_j|}{|N_j|} \text{ if } |n_j| \neq |N_j| \text{ and } |n_j| > 1 \\ 0 \qquad \text{otherwise} \end{cases} \tag{11}$$

Swarm-GAP uses the polymorphism to setup the agents thresholds according to the 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 (because threshold and capability are inversely proportional values).

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)
```

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 10. 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.

4 Experiments and Results

Empirical evaluations was conducted in an abstract, domain-independent simulator written in C++. This simulator allows experimentation with a large number of agents and tasks. In each experiment we have 2000 tasks, 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 a 60% probability of having a non-zero capability for each class. In this case, they are uniformly assigned, with values ranging from 0 to 1. When new tasks arise at each cycle (the total number of tasks is maintained constant), they are randomly distributed to agents.

At each step, agents are constrained in the number of messages they can send to a maximum of 20. Rewards are computed over 1000 rounds, where in each round one allocation is performed. All data is averaged over 20 runs. Swarm-GAP is compared with two methods: LA-DCOP, mentioned earlier on this paper, and a centralized greedy algorithm. The greedy strategy, used to benchmark other methods, allocates the appearing tasks to the most capable available agent, but does this in a centralized way. This simulation has exactly the same setup used by [8] to experiment LA-DCOP.

In the first experiment the aim is to find the best stimulus value (Equation 10) 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. 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.)

As we can see in Figure 2, when the number of agents is equal to or greater than the number of tasks, the stimulus that maximizes the reward change be-

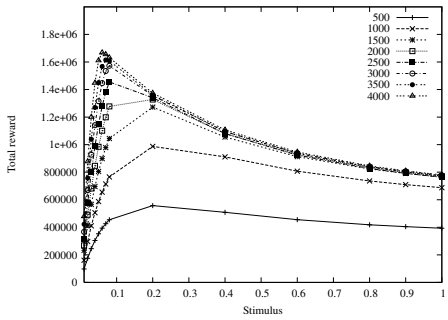

Fig. 1. Comparison of the total reward with the task stimulus, for different number of agents.

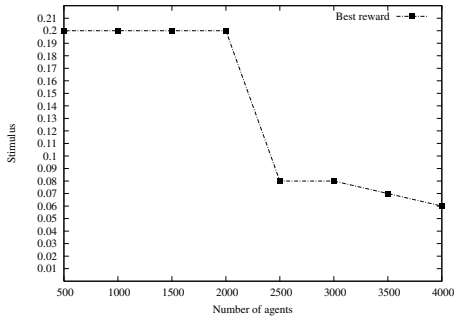

Fig. 2. Stimulus for the best total reward, for different number of agents.

tween 0.06 and 0.08 (i.e. for 2000, 2500, etc.) When the number of agents decreases, the stimulus related to the best reward increases to 0.2. This experiment shows that, to maximize the rewards, the stimulus must change proportionally to the rate on the number agents and tasks. The same happens with the thresholds in LA-DCOP.

In the second experiment, we measure the average number of messages per simulation cicle, 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 3 , when LA-DCOP works with threshold equal to zero, the number of messages changed 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 4, on 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. Swarm-GAP with 0.2 and LA-DCOP with 0.8 achieve similar rewards (Figure 4), Swarm-GAP is less than 10%

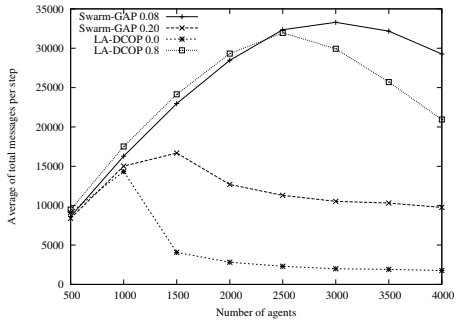

Fig. 3. Average number of messages per simulation cicle versus the number of agents.

worst than LA-DCOP, but the number of exchanged messages by LA-DCOP is dramatically greater than Swarm-GAP, about 100% on average.

In the third experiment, we evaluate Swarm-GAP comparing its results with those achieved by the greedy centralized algorithm and LA-DCOP. Figure 4 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 with also the best threshold for each number of agents. As expected, the greedy approach outperforms Swarm-GAP and LA-DCOP. Swarm-GAP performs well achieving rewards that are only 20% lower (on average) than the greedy ones and 15% than LA-DCOP. The greedy strategy, used to benchmark other methods, allocates the appearing tasks to the most capable available agent, but does this in a centralized way.

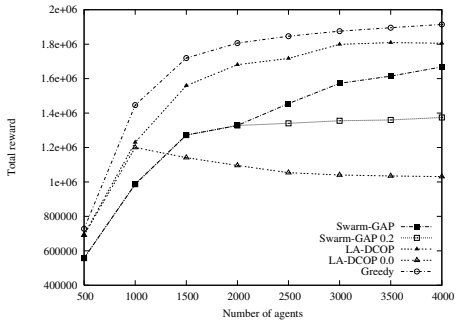

Fig. 4. Comparison of the total reward for different number of agents.

To illustrate the advantages of Swarm-GAP related to LA-DCOP regarding the number of exchanged messages, Figure 5 shows the average reward divided by

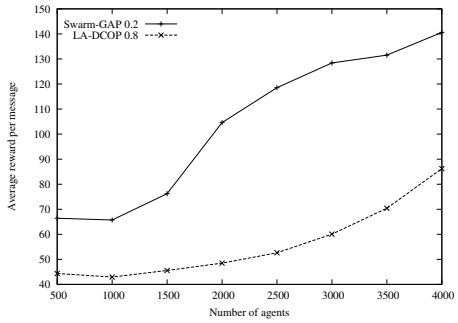

Fig. 5. Comparison of the average reward divided by the total number of exchanged messages for different number of agents.

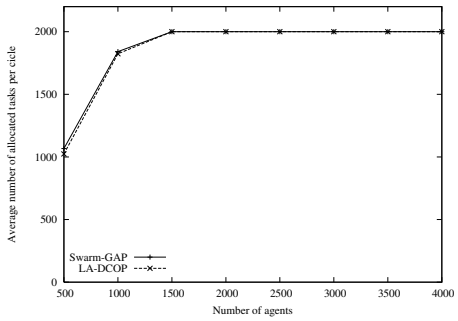

Fig. 6. Comparison of the average number of allocated tasks per cicle, for different number of agents.

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. This setup of the algorithms leads 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 to agents take 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 allocate tasks according a probability computed by equation 10, constrained by its available resources. This is a simple one-shot decision process.

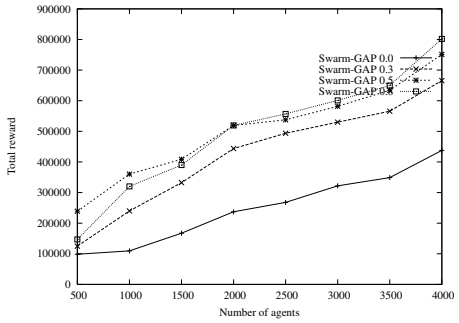

Fig. 7. Comparing the total reward for different number of agents, in the presence of AND constrained tasks.

Figure 6 depicts the average number of allocated tasks per step of simulation according to the number of agents achieved by Swarm-GAP and LA-DCOP. Both algorithms allocate almost the same number of tasks. The overall performance in terms of reward achieved by Swarm-GAP is similar with the one achieved by LA-DCOP.

The last experiment with the abstract scenario measures the impact that AND-constrained tasks have over the reward. In this experiment, 60% of the tasks are AND constrained in groups of 5. Figure 7 shows the expected decrease in the performance of Swarm-GAP when we consider the AND constraints and several values for ω (Equation 10). Rewards improve when $\omega \neq 0$, improving the average performance in 25%.

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. The presented algorithm solves complex DCOPs in an approximated and distributed fashion. Swarm-GAP intends to be a simple and effective algorithm.

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 the Swarm-GAP performs well, achieving rewards, on average, only 20% worse than the ones achieved by a greedy centralized approach and 15% than the ones achieved by other recent proposed algorithm. However, by the nature of its mechanisms, Swarm-GAP uses significant less communication and computation than the other algorithm mentioned above. The execution coefficient improves the average reward in 25% in scenarios with several interrelated tasks.

In the future, 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 failures easily with simple mechanisms.

References

1. V. A. Cicirello and S. F. Smith. Wasp-like agents for distributed factory coordination. *Autonomous Agents and Multi-Agent Systems*, 8(3):237–266, May 2004.
2. C. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.
3. P. R. F. Jr. and A. L. C. Bazzan. Distributed meeting schedule through cooperative mediation: analysing optapo's performance in a complex scenario. In A. Meisels, editor, *Proceedings of the Sixth International Workshop on Distributed Constraint Reasoning (DCR2005) - Nineteenth International Conference on Artificial Intelligence (IJCAI2005)*, pages 101–113, July 2005.
4. V. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):133–142, January 1999.
5. R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 310–317, New York, USA, July 2004. Los Alamitos, IEEE Computer Society.
6. P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proc. of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 161–168, Melbourne, Australia, 2003. New York, ACM Press.
7. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proc. of the Nineteenth International Conference on Artificial Intelligence*, pages 266–271, Edinburgh, Scotland, Aug 2005.
8. P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *Proc. of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 727–734, Utrecht, The Netherlands, 2005. New York, ACM Press.
9. D. B. Shmoys and V. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(3):461–474, 1993.
10. G. Theraulaz, E. Bonabeau, and J. Deneubourg. Response threshold reinforcement and division of labour in insect societies. In *Royal Society of London Series B – Biological Sciences*, volume 265, pages 327–332, 2 1998.
11. A. Visser, G. Pavlin, S. van Gosliga, and M. Maris. Self-organization of multi-agent systems. In *Proc. of the International Workshop on Military Applications of Agent Technology in ICT and Robotics*, The Hague, The Netherlands, 2004.
12. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.