Local Ant System for Allocating Robot Swarms to Time-constrained Tasks

Yara Khaluf*, Seppe Vanhee, Pieter Simoens

IDLab, Department of Information Technology at Ghent University - imec
Technologiepark 15, B-9052 Gent, Belgium

Abstract

We propose a novel application of the ant colony optimization algorithm to efficiently allocate a swarm of homogeneous robots to a set of tasks that need to be accomplished by specific deadlines. We exploit the local communication between robots to periodically evaluate the quality of the allocation solutions, and agents select independently among the high-quality alternatives. The evaluation is performed using pheromone trails to favor allocations which minimize the execution time of the tasks. Our approach is validated in both static and dynamic environments (i.e. the task availability changes over time) using different sets of physics-based simulations.

Keywords: Swarm robotics, Ant Colony Optimization, Task Allocation

1. Introduction

Swarm robotics is an approach to distributed robotics, in which large groups of robots execute tasks of which the complexity exceeds the capabilities of an individual robot. It holds promise as a useful approach for many real-world tasks through reducing costs, risks and execution times [1]. Many of the tasks for which swarm robotics can provide an efficient solution are associated with time constraints (i.e. deadlines). Examples of time-constrained tasks include rescuing

☆Local Ant System for Allocating Robot Swarms to Time-constrained Tasks
*Corresponding author
Email addresses: yara.khaluf@ugent.be (Yara Khaluf), Seppe.Vanhee@UGent.be (Seppe Vanhee), pieter.simoens@ugent.be (Pieter Simoens)

humans or objects in a disaster site before some level of harm is reached [2], collecting objects/products and transfer them to an aggregation point for further processing [3], or planting and harvesting of large fields using an automated agricultural robot team [4]. Thus, the consideration of time constraints when designing robot swarms represents a fundamental requirement for successful application of these systems in real-world scenarios.

In general, the design of robot swarms can follow either a microscopic-to-macroscopic paradigm or a macroscopic-to-microscopic one. The first approach starts from the individual (i.e. the microscopic) level by defining the behavior rules that are followed then by each robot. These rules are in general simple and often probabilistic, yet they result in a collective behavior observed at the swarm (i.e. the macroscopic) level. Applications that are associated with this design approach take inspiration from nature and aim either to create large-scale autonomous artificial systems or to validate functionalities that are observed in nature. Examples of these behaviors include; foraging [5, 6, 7, 8], flocking and motion coordination [9, 10], aggregation [11, 12]. The second design paradigm starts by defining a desired swarm behavior and attempts to derive the individual rules that lead to achieving this particular behavior. Due to its complexity, only a few works in the literature design robot swarms following this paradigm [13, 14, 15, 16]. In this paper, we apply a macroscopic-to-microscopic approach to design a swarm that collectively executes a set of time-constrained tasks, in which the task's deadlines represent the macroscopic conditions that we aim to satisfy.

We propose an autonomous task allocation algorithm for deriving the microscopic behavior rules in the robot swarm. These rules allow the robots to allocate themselves to the different active—i.e. of which the deadline is not exceeded yet—tasks considering their time constraints. Few works in the literature have tackled the problem of task allocation in swarm robotics for time-constrained tasks [17, 18]. Nevertheless, in these works, a priori knowledge of the task's execution time is assumed in addition to considering only static environments (i.e. the density of items doesn't change over time). Differently, in our study, we don't assume any a priori knowledge of the items' execution time. Moreover, we treat this time as the parameter we aim to optimize. Additionally, we address dynamic as well as static environments. In our study, we propose a novel use of the ant colony optimization (ACO) algorithm [19] to optimize the number of robots assigned to each task from a set of time-constrained tasks. To the best of our knowledge, there are no works which attempted to apply ACO for allocating robot swarms to time-constrained tasks.

The rest of the paper is organized as follows. In section 2, we give a brief background on the ant colony optimization algorithm and how solutions are constructed. In section 3, we formulate the particular allocation problem that we are focusing on and the scenario we use to study our problem. The ACO details and representation are explained in Section 4. In Section 5, we present the individual behavior rules that robots apply to allocate themselves autonomously to the different tasks. We validate the proposed algorithm in Section 6 using different sets of physics-based simulations. Conclusions are drawn in Section 7.

2. Background

In this section, we present a brief description of the Ant Colony Optimization (ACO) algorithm. Readers familiar with ACO may skip this section.

ACO is an optimization technique that draws inspiration from the foraging behavior of ants. It is a meta-heuristic approach that was proposed originally by Marco Dorigo to find the optimal path in a graph [20]. Later on, several variations were proposed to the original algorithm [21]. Optimization problems that are solved using ACO are combinatorial problems—optimal solution has to be identified from a finite set of solutions—that can be modeled using (directed or undirected) graphs, in which nodes represent discrete states whose combinations provide different solutions. In general, specific costs are assigned when traveling over the edges of the graph. An individual ant decides to travel from node i to node j with a probability:

$$P_{ij}^{t} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in G_i}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}, \tag{1}$$

where G_i is the set of neighbors of node i, and t is the current time step. η_{ij} is the heuristic information available for the individuals to help their decision (e.g., the distance between two nodes). Finally, τ_{ij} is the amount of pheromone assigned to the selected solution—i.e. switch from i to j—based how good this solution is according to the collected experiences. α and β are two parameters, which respectively determine the relative influence of the heuristic information and the pheromone on the selection of the next step. The two parameters define the weights of the exploration and exploitation behaviors that are applied by each individual at each time step. The parameter α represents the tendency of the individual to exploit the information that is collected by other individuals and that is coded using the pheromone trails. The parameter β reflects the tendency of the individual to explore the problem state space looking for new solutions (i.e. paths). Finding a proper balance between the exploration and the exploitation (how to tune α and β for an optimized performance) is one of the main challenges when applying any variant of ACO. Therefore, in most studies, one of the inevitable steps is to tune these two parameters [22, 23]. Accordingly, we investigate in our study the fundamental role of both parameters is achieving a desired allocation of the robots under the different assumptions in static and dynamic environments.

In ACO, a number of artificial ants iteratively build solutions of an optimization problem and use pheromones as a mean of communication to exchange information concerning the quality of the constructed solutions. In general, the pheromone update process continuously evaluates the solution by rewarding good choices and forgetting bad ones by pheromone deposition or evaporation respectively:

$$\tau_{ij}(t+1) \leftarrow (1-\epsilon)\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{2}$$

where $\tau_{ij}(t)$ is the amount of pheromone at time t, and ϵ is the parameter used to enable the algorithm to forget previous bad decisions through the evaporation of the pheromone. $\Delta\tau_{ij}(\mathrm{t})$ is the amount of pheromone to deposit on edge e_{ij} at time step t. Note that the pheromone update is a global process, i.e. the whole problem graph is updated by a central component.

ACO has been applied to a wide range of combinatorial optimization problems such as scheduling problems [24, 25], routing problems [26, 27], assignment problems [28, 29], etc. It has showed a high efficiency in finding near-optimal solutions within reasonable execution times.

3. Problem Description and Scenario

We consider a homogeneous swarm of N robots, which is used to execute a set of M time-constrained tasks in parallel. Each of the M tasks consists of a large number of identical work items (sub-tasks). Each task m_i is thus characterized by its minimum number of items to execute (S_i) and its deadline D_i. The items of a task can be executed in parallel and in arbitrary order. This abstracted description of robotic tasks can be translated to a large range of real-world application domains, such as agriculture [30], construction [31], transportation [32], and others. For example, in agriculture tasks robots need to pick-up a particular number of products or plant a number of seeds within a given time duration. The seeds and plants are the task items. In more general scenarios, the robotic task can be a combination of several missions to be carried out simultaneously at different sites, e.g., picking-up and planting at two fields in parallel. In this study, we tackle the general case of having M tasks presented at different locations. Both the time to seek the locations at which the robots will find the plants or plant the seeds, and the times the robots spend in avoiding obstacles, are stochastic due to the dynamic nature of the navigation activity. Therefore, the execution time of the single item becomes stochastic as well.

We model the execution time of a single item to include three time intervals:

1. The seeking time, which is the time the robot spent in searching for an item since it entered the task site (as explained below) or since it finished the execution of the previous work item.
2. The processing time of the item. This time is task-specific but constant for all items.

3. The avoidance time, which is the time spent in avoiding collisions with other robots.

Both seeking and avoidance times are stochastic. Their stochasticity depends on the item density, the spatial distribution of the items and the intensity of the physical interferences at the task site (more robots will lead to more interferences). Consequently, the execution time of a single item that sums up all three times (mentioned above) is stochastic, and so is the total time required to execute a particular number (S_i) of items of task m_i.

Let the average seeking time of an item of task m_i be denoted by $\hat{\theta}_i$. Similarly, the average avoidance time is denoted by $\hat{\omega}_i$. The key goal of our proposed allocation algorithm is to find an allocation $A(t)$ at time step t—$A(t) = \{N_1(t), N_2(t), \ldots, N_{K(t)}(t)\}$, where $K(t)$ is the set of tasks whose deadlines are not exceeded yet—that assigns $N_i(t)$ of robots to task m_i at time step t so that task execution time is minimized. The minimization of the task's execution time is achieved by minimizing the average seeking time $\hat{\theta}_i$ and/or the average avoidance time $\hat{\omega}_i$ at the site of task m_i. Minimizing the execution time of a single item will impose a maximization of the number of executed items, and consequently increasing the probability of executing the minimum number of items (S_i) required, before the deadline (D_i) is exceeded. In the most general case, the density or the spatial distribution of items changes over time, e.g. because executed items are not replaced. Such environments are referred to as dynamic environments, in which the average seeking ($\hat{\theta}_j$) and avoidance ($\hat{\omega}_j$) time changes over time. In this paper, we consider dynamic but also static environments, in which $\hat{\theta}_j$ and $\hat{\omega}_j$ stay constant over time.

Formally, our setting can be represented as the following multi-objective optimization problem:

$$\underset{A(t)=\{N_1(t),N_2(t),\ldots,N_{K(t)}(t)\}}{\text{minimize}} \quad \begin{cases} \hat{\theta}_i(t) = f(N_i(t)) & \forall m_i \in M \\ \hat{\omega}_i(t) = f(N_i(t)) & \forall m_i \in M \end{cases} \tag{3}$$

The functions $\hat{\theta}_i(t)$ and $\hat{\omega}_i(t)$ are not known a priori. Therefore, the optimization problem is solved online and distributively by the individual robots.

Our proposed approach exploits the well-known ant colony optimization [33] to solve the above optimization problem. Because ACO is executed by individual robots, we refer to our allocation algorithm as Local Ant System (LAS).

Fig.1 depicts the concrete scenario that we use in our study to illustrate the above-defined problem. We have $M = 4$ task locations. On each location, a random number of work items of that task is deployed. The task locations are connected via a common space referred to as the nest. Robots start at the nest, select a task, move to the task site, search and execute zero, one or more work items. Robots re-visit the nest during specific periods, referred to as the synchronization periods. These periods are used to exchange information among the robots. Robots communicate only in the nest. After each synchronization period, robots re-select their next task, using the LAS algorithm based on the information they have collected while interacting with their neighbors at the nest. The time periods during which robots are at the task sites are referred to as execution phases. The execution phase may differ from one robot to another, when the robots select random times to spend at the task site before coming back to the nest.

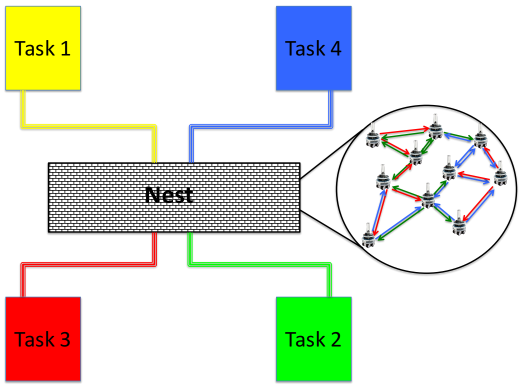

Figure 1: A simplified scenario for the studied task allocation problem.

4. Local Ant system for the task allocation problem (LAS)

Our task allocation problem can be represented using a connected graph $G = \{V, E\}$, see Figure 2. The nodes V represent the set of tasks and its edges E represent a robot decision to switch from one task to another. Our graph is fully connected since robots can switch from any task to any other task. We modify the traditional ACO algorithm by applying a local pheromone update (i.e performed by the robot) instead of having a centralized and global procedure, as we will explain in the following.

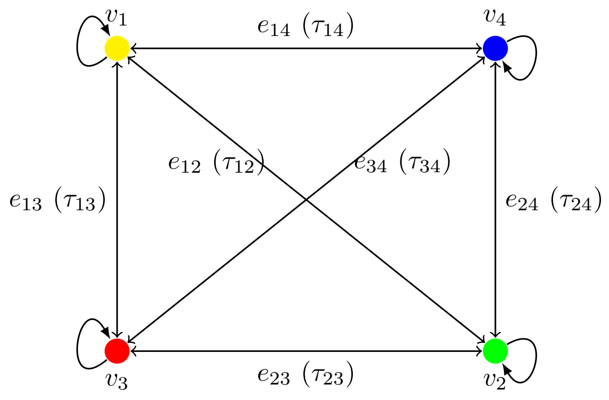

Figure 2: A graphic representation of the task allocation problem.

4.1. Solution Construction

Initially, a robot selects randomly one of the given tasks, and the allocation solution is constructed periodically over several steps, which are executed each time the robot visits the nest. At each construction step (i.e. during the synchronization period), the robot applies the probabilistic choice rule in Eq. (4) to select the task to work on during its next execution phase. The probability of switching from task m_i to task m_j is then computed as follows:

$$P_{ij}^t = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in K(t)} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta} \tag{4}$$

where $K(t)$ is the set of active tasks at time step t, η_{ij} is a heuristic value, and τ_{ij} is the pheromone value. In the following, we present both the heuristic information used to capture the information available for helping the allocation decision and the pheromone information used by the LAS allocation algorithm.

4.2. Heuristic Information

As mentioned above, each task is characterized by the minimum number of items to execute (S_i) and its deadline. We use these two parameters to define the task's priority γ_i as given in Eq. (5):

$$\gamma_i = \begin{cases} \dfrac{S_i}{D_i} & \text{for } t \leq D_i \\ 0 & \text{for } t > D_i \end{cases} \tag{5}$$

The task priority is a quantitative measure that reflects the urgency of the task. It is higher when the number of items that need to be executed is higher and/or when the deadlines are shorter.

We exploit the priority of the task as the piece of information available for the robots to help their allocation decision in light of the task's urgency. Thus, the task's relative priority—i.e. its priority relative to the priority of other tasks—is used as the heuristic information. η_j is the heuristic information associated with the switch from any active task to the active task m_j, and it is given by:

$$\eta_j = \frac{\gamma_j}{\sum\limits_{k \epsilon K(t)} \gamma_k} \tag{6}$$

where the set $K(t)$ is the set of active tasks at time step t. Note that the heuristic information of task m_j does not depend on the currently chosen task m_i, as the switching cost between any two tasks is assumed to be equal.

Robots keep track of the tasks deadlines, and as soon as a deadline is expired the priority of that task is set to zero and consequently the heuristic information as well. This prevents the task from being chosen when its deadline is exceeded. With this definition of η_j, the parameter α in Eq. (4) determines the importance of assigning robots based on the relative priority of the tasks.

4.3. Pheromone Update

Different from traditional ACO-based schemes, in our proposed solution the pheromone update is performed individually by each robot. Consequently, each robot develops its individual solution for the given macroscopic optimization problem in Eq. (3). For doing so, each robot maintains a pheromone table that contains the pheromone amounts assigned to the different switching options (including staying at the currently selected task). The robot updates its pheromone table during each visit to the nest (i.e. each synchronization period).

In the nest, before to update the pheromone table, the robot exchanges information with its neighbours concerning their collected experiences during their latest execution phases. These experiences involve both the seeking time θ_i and avoidance time ω_i that each robot experience in its previous execution phase. By averaging all the communicated values, the robot calculates new estimates $\hat{\theta}_i$ and $\hat{\omega}_i$ for each task m_i. The robot then deposits a pheromone update (positive or negative) on each edge e_{ij}, calculated as follows:

$$\triangle\tau_{ij} = \frac{\hat{\theta}_j \times \hat{\omega}_i}{\hat{\omega}_j \times \hat{\theta}_i} \tag{7}$$

Eq. (7) translates the macroscopic goal of minimizing the task's execution time—by minimizing both of the avoidance and the seeking times—to the individual behavior. For the avoidance time ω_i, it is straightforward that increasing the density of robots at a specific task site beyond a particular limit increases physical interferences ([5]) among robots and consequently, the time robots spend in avoiding each other. Hence, by depositing a smaller amount of pheromones for tasks with higher avoidance times, the number of robots at that task site will decrease accordingly.

The decision on increasing or decreasing the number of robots allocated to a particular task based on the estimated item's seeking time $\hat{\theta}_i$ is more complicated if defining the seeking time to include the item the robot searches for the item and avoiding other robots. This is due to the fact that a large seeking time can result from either (i) having to spend a long time in avoiding other robots or

(ii) a low item density at the task site. In the first case, the number of robots assigned to the task needs to be reduced, whereas in the second case the number of robots needs to be increased in order to find the items faster. In order to avoid such conundrum, we define the seeking time as the time spent by the robot to find an item after subtracting the time this robot spends in avoiding obstacles (e.g., other robots). Thus a large average of the seeking time indicates only the need to increase the number of robots allocated to this particular task and hence a higher amount of pheromone is deposited to encourage the switching to this task.

According to the above-described macroscopic-microscopic behaviors, pheromone trails are added to reinforce a switch between two tasks, when this switch allows to gain in terms of the average avoidance time or/and the average item's seeking time.

5. Implementation of the local ant system (LAS)

In this section, we describe how we have implemented the algorithm described in the previous section into a behavioral model of the robots. Our implementation meets the requirement of simplicity that is imposed by the limited capabilities associated with swarm robotics.

Robots can be in one of the following two states: (i) Uncommitted: an uncommitted robot is a robot that doesn't decide yet for its next task. Each time a robot visits the nest to exchange information with the other robots, it becomes uncommitted. (ii) Committed: a committed robot is a robot that has decided for its next task. This decision is taken by each robot individually using the LAS algorithm executed locally by the robot.

Initially, robots start scattered at the nest site. A robot selects its next task randomly with a probability biased by the task's priority. Afterwards, robots sample the time period for their visit to the selected task site. These times are sampled from a normal distribution with the mean $\mu_{execute}$ and standard deviation $\sigma_{execute}$, and referred to as the *execution phase*. Robots use their cameras to identify the led associated with their selected tasks and start moving in that direction. The robot detects the task site using a combination of the ground sensors reading (all tasks have gray grounds) and their cameras (the specific led of the selected task). As soon as the robot arrives at the task site, the execution phase starts to count-down while the robot is searching for items. During the execution phase, the robots wonder following a random walk with collision avoidance. When an item is found, the robot spends a constant time executing the item before it resumes its random walk. As mentioned above, in static environments, the items are regenerated at new locations over the task site. Whereas in dynamic environments, the executed items are removed from the task site, and hence the density of the items decreases over time.

Robots continue executing the items of their selected task until their execution phase is over. Afterwards, robots start to move towards the nest for executing a new synchronization period, in which they exchange the experiences collected during their last task's visit.

During the synchronization period, robots use their range-and-bearing system[1] to exchange messages from the following format: (i) Byte-0 to exchange the robot ID, (ii) Byte-1 is set to 0 when exchanging avoidance time, to 1 when exchanging item seeking time, and to 2 when exchanging the task ID. (iii) Bytes-(2-9) for exchanging the actual data. The receiver robot will check the message for the sender id and the time-stamp to avoid accounting for the same message more than once.

At the end of the synchronization period, the robot exploits the information exchanged during this period to evaluate the average of the seeking and the avoidance times of each task. These two averages are used in Eq. (7) to update the pheromone amount assigned to each active task, and consequently, the allocation probability for each task Eq. (4). Algo. 1 illustrates the behavior of the individual robot during the experiment time over both the execution phase,

[1]The range-and-bearing system of simulated footbots allows the exchange of 10-byte messages

and the synchronization period.

Algorithm 1: The robot's (microscopic) behavior

```
1  while $K(t)$ is not empty do
2      select next task using Eq. (4);
3      sample a new execution_phase to spent at the selected task;
4      while execution_phase $> 0$ do
5          seek items at the task site;
6          measure and average the avoidance time;
7          measure and average the item seeking time;
8      end
9      return to the nest;
10     while synchronization_period $> 0$ do
11         exchange with the neighborhood and estimated:
12         (a) the average avoidance time;
13         (b) the average seeking time;
14     end
15     update pheromone trails using Eq. (7);
16 end
```

6. Results and Discussion

6.1. Arena setup

To verify our approach, we conducted a set of physics-based simulations implemented using the ARGoS simulator[2]. Fig. 3 depicts the 26×26 m^2 arena, which consists of one nest and four task sites. The nest is the central part of the arena (10×10 m^2) where robots meet to communicate and exchange

[2]ARGoS is a discrete-time physics-based simulator. It can simulate various robots in large numbers and at different levels of details.

their individual experiences. The communication range of the robot is set to 1 m. Each task site measures 8×8 m^2 and can be reached from the nest by an entrance with a width of 2 m. Robots distinguish between the different task sites through colored LEDs (red, yellow, blue, and green), which are placed at the entrance of each site and the colors are used to label the tasks. To help the robots returning from a task site to the nest, orange LEDs are placed at the nest side of each task entrance. The items of each task are simulated as black spots, which are scattered over the task site. The execution time of an item is constant per task and can differ from one task to another representing the level of difficulty.

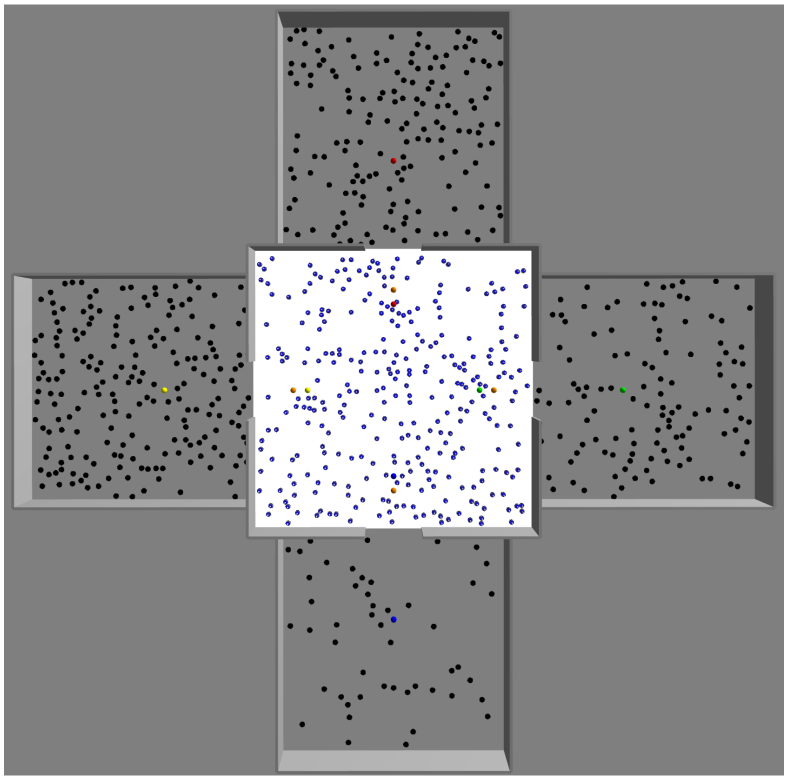

Figure 3: A birds-eye view of the arena. The part with the white floor is the nest used by the robots to communicate and update their pheromone trails as well as their allocation probabilities. The other four parts represent the task sites (in the snapshot with different item densities).

6.2. Benchmark

For task allocation in robot swarms, one of the widely-applied models is the "Response Threshold Model" (RTM) [34, 35, 36, 37, 38]. This model is inspired from the principle of division of labor that is observed in social insects and other animal groups. It is used by the individuals to allocate themselves to the different tasks and to specialize on these tasks [39]. In the RTM model, each individual measures one or more stimuli in its environment and when the level exceeds a particular threshold (in most studies this threshold is constant over time) the individual probability to switch its task increases. For example in a set of sequential tasks, in which an agent is allocated to task B but needs to wait until some task A is executed, the waiting time is defined as the stimulus. When the agent waits longer than a specified threshold, its probability to switch to task A increases. In robot swarms, RTM is mostly used to optimize the number of robots allocated to each task, accounting for physical interferences. We refer to the optimal number as the number of robots that maximizes the swarm performance under spatial interferences.
We have selected this optimal number of robots to use as our benchmark when evaluating the performance of the LAS algorithm. Please note that the optimal number of robots is generally computed in static environments. Nevertheless, since we consider both static and dynamic environments—i.e. the density of the task changes over time—, we developed an approximated function to characterize how the optimal number of robots changes over time and used this as our benchmark in the dynamic environments.

- Static environments: in static environments, we have determined the optimal number of robots for the four tasks, accounting for spatial interferences. To do so, we have run exploratory simulations, in which we measured the total number of executed items over 1000 time steps, in the arena of Fig. 3, with only items in one of the task sites. We ran simulations with the number of robots in the range of $\{20, 520\}$ with steps of 10. Initially, the number of executed items increased with the number of robots allocated to the task. However, due to spatial interferences, this number started to decrease again when the number of robots exceeds a specific limit—i.e. the optimal number of robots. Based on our simulations, the optimal number of robots is $\{260, 230, 240, 210\}$, for the yellow, the green, the red, and the blue tasks, respectively. Please note, that the summation of the optimal number of robots for the different tasks is 940, which is considerably above the swarm size ($N = 300$). Note that our experimental setup is purposely designed so that the swarm is not large enough to assign the optimal number of robots to each task.

- Dynamic environments: Due to the change in the number of items over time , the optimal number of robots is not constant. Therefore, in order to compare the number of robots allocated by LAS to each task with the optimal number, under the current item density, we derive a function to compute the optimal number of robots in terms of the number of available items. For this purpose, we perform experiments with different item densities $[50 - 600]$ with a jump of 50 items per configuration. Afterwards, we extrapolate the optimal number of robots as a function of the items number:

$$N_{opt} = -0.0004755x^2 + 0.711x + 29.45, \tag{8}$$

  where x is the current number of items available at the task site.

In the following, we study the influence of the exploration (α) and exploitation (β) parameters on the performance of LAS and compare the obtained performance to the optimal number of robots computed in a set of independent simulations.

6.3. Influence of the exploration exponent (α)

We first investigate the influence of the exploration exponent α on the performance of the algorithm LAS. We recall the reader that the value of α determines how important it is to assign the robots to the different tasks based on the heuristic information (i.e. the relative priority of the task). Since the priority of a specific task is computed as in Eq. (5) using its deadline and the minimum number of items to execute of that task, to quantify the influence of α, we consider static environments where tasks have different priorities but their relative priorities (i.e. how urgent this task in comparison to the other active tasks) do not change over time. The task and the experiment parameters are shown in Tab. 1 and Tab. 2, respectively. The task sizes and deadlines are selected such that we cover a range of differences between task priorities.

Task	**Deadline**	**# of items**	**Size**	**Execution time**	**Priority**
Yellow	12000	200	2000	8	0,16
Green	9000	120	3000	5	0,33
Red	6000	160	3000	10	0,5
Blue	3000	50	2000	5	0,66

Table 1: Task parameters including from left to right: task deadline, initial number of items available at the task site, the minimum number of items need to be executed of the task, the execution time of each item, and the task's priority computed using Eq. (6).

Parameter	**Value**
Swarm size	300
Mean of foraging time $\mu_{execute}$	300
Standard deviation of foraging time $\sigma_{execute}$	10
Nest time	200
The exploitation exponent β	1.0

Table 2: The experiment parameters.

We apply LAS with $\alpha = 2.5$ and $\beta = 1.0$. Setting a higher value of α compared to β assigns a higher importance to the relative task's priority as a decision motive when applying the probabilistic decision in Eq. (4).

In Fig. 4, we can notice that robots are assigned during the first 3000 seconds (until the deadline of the blue task with the highest priority) to the blue task hitting its optimal number of robots (the horizontal blue dotted line). The number of robots allocated to the other tasks is significantly far from their optimal numbers. When the deadline of the blue task is exceeded, the red task becomes the task with the highest relative priority, hence the allocation algorithm (LAS) pushes the number of robots allocated to this task as near as possible to its optimal number (up to its deadline 6000 seconds).

Reaching the optimal number (230 for the red task) was not possible due to the requirements of robots at the other task sites (the green and the yellow) reflected in their pheromone trails ($\beta = 1.0$). Hence, a specific number of robots needs to be assigned to these tasks as a result of the exploitation parameter. As soon as the deadline of the red task expires, the green task is assigned the highest number of robots (around its optimal number 240), and the yellow task (i.e. with the lowest priority) is assigned all robots when it becomes the only active task.

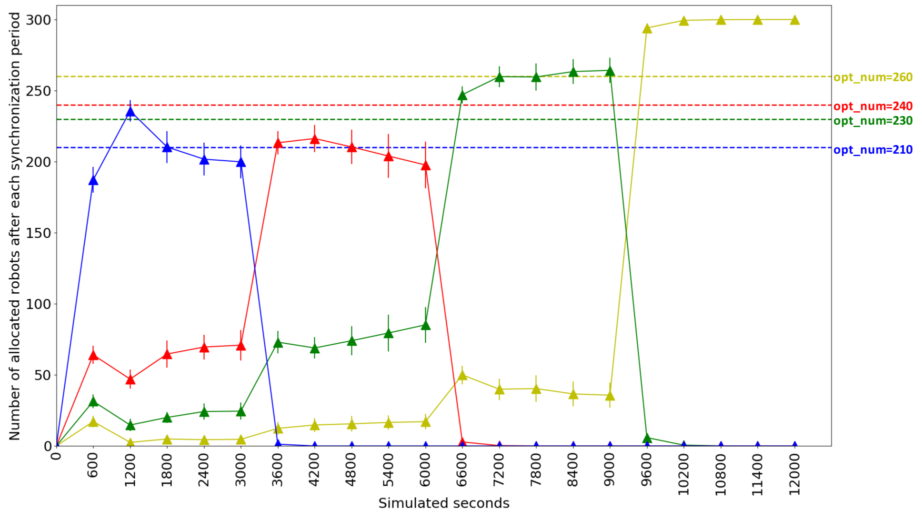

Figure 4: The number of allocated robots to the different tasks for $\alpha = 2.5$ and $\beta = 1.0$ in comparison to the optimal number computed under spatial interferences. Experiments performed in a static environment (i.e. no change in the number of items available at the site of each task).

In Fig.5, we compare the performance of the allocation algorithm (LAS) for three α values (i) low influence $\alpha = 0.5$, (ii) high influence $\alpha = 2.5$, and (iii) extremely high influence $\alpha = 5.0$, all with $\beta = 1.0$. In this figure, we can notice similar dynamics to the one observed in Fig. 4, by favoring tasks based on their relative priorities. Nevertheless, having $\beta > 0$ imposes the exploitation of the experiences collected by the robots—during their visits to the task sites—in the decision process (see Eq. (4)), and hence LAS continues allocating a number of robots > 0 to the tasks with lower priorities as well. In this figure, we can see that even with the lowest value used for $\alpha = 0.5$, tasks with higher priorities are favored, although the number of assigned robots is far below the optimal number of robots for all tasks. Conversely, for the highest value of $\alpha = 5$, the number of assigned robots is far above the optimal number.

Consequently, tuning the parameter α for an efficient performance of LAS is a critical task that needs to be addressed for (i) static environments by assigning higher importance to the heuristic information of the ant system (i.e. the α value is set higher than the β value), and (ii) identifying the upper-bound of α, so that the robots' experiences reflected in the pheromone trails still have an influence on the robots' allocation. Such an upper-bound can be found by using an automatic parameter tuning [40] over the data generated from robotics simulations.

Fig. 6 depicts the total number of items executed of each task using the different three values of $\alpha \in \{0.5, 2.5, 5.0\}$ and $\beta = 1.0$. In this figure, we can notice that LAS was able to meet all deadlines by executing (and in most cases exceeding) the minimum number of items S_i required of each task. The blue task is the one with the highest relative priority and was satisfied for two values of $\alpha \in \{2.5, 5.0\}$, but has missed its deadline for $\alpha = 0.5$. Interestingly, we can notice that having extremely high values of α can lead in some cases to over-assign robots to tasks, so that the physical interferences between robots lead to decrease the number of items executed, such as the case of the green task with $\alpha = 5.0$.

Finally, for static environments, we conduct a set of experiments to verify that, both averages of the seeking and avoidance times, which are estimated

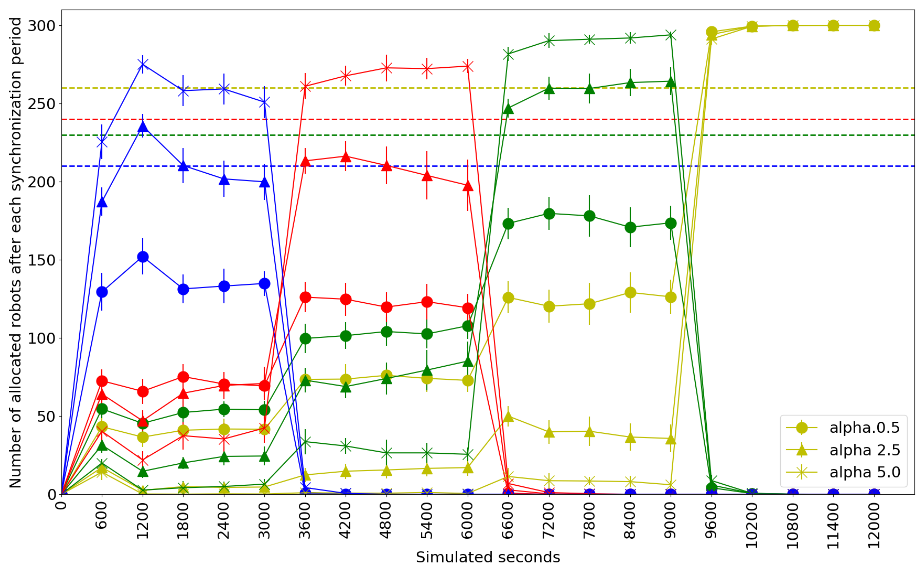

Figure 5: The number of allocated robots to the different tasks using 3 values of $\alpha \in \{0.5, 2.5, 5\}$ and $\beta = 1.0$, experiments performed in a static environment (i.e. no change in the number of items available at the site of each task).

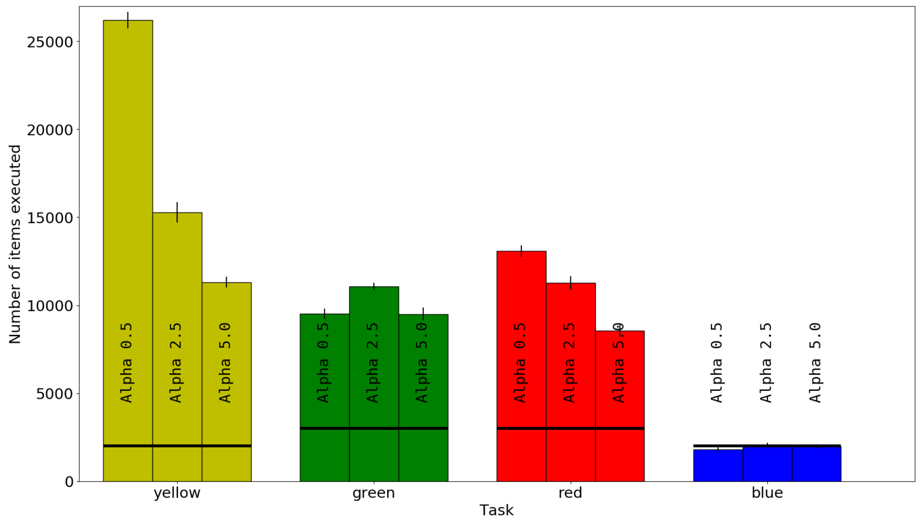

Figure 6: The number of items executed of each of the four tasks using $\alpha \in \{0.5, 2.5, 5.0\}$ and $\beta = 1.0$. Experiments performed in a static environment (i.e. no change in the number of items available at the site of each task).

by the individual robots (during their execution phases) are distributed around the total average (computed over all robots), and to infer the variances of the individual estimations. Fig. 7 and Fig. 8 show our results, which verify a minimal variance for the total average overall tasks for both types of times.

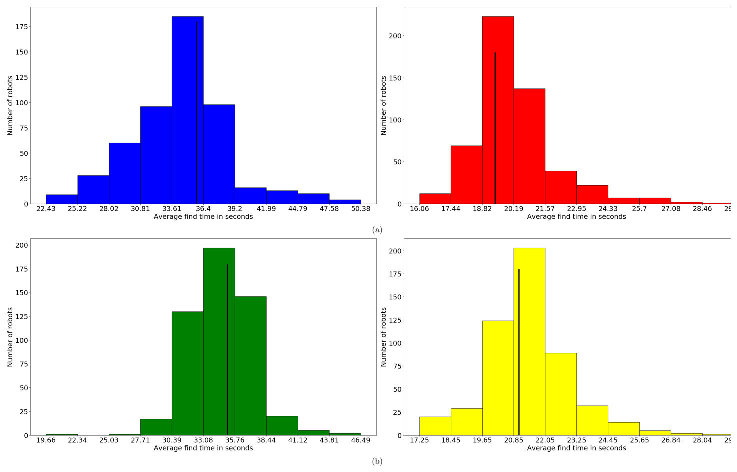

Figure 7: A comparison between the average seeking time estimated by the robots individually and the total average computed over all robots (the vertical line).

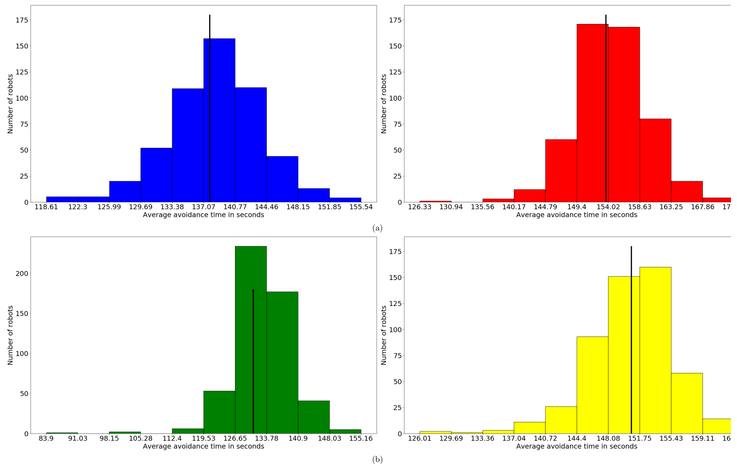

Figure 8: A comparison between the average avoidance time estimated by the robots individually and the total average computed over all robots (the vertical line).

Task	**Deadline**	**# of items**	**Size**	**Execution time**	**Priority**
Yellow	18000	400	400	200	0,02
Green	13500	600	600	200	0,04
Red	9000	600	600	200	0,06
Blue	4500	400	400	200	0,08

Table 3: Task parameters including from left to right: task deadline, initial number of items available at the task site, the minimum number of items need to be executed of that task, the execution time of each item, and the task's initial priority computed using Eq. (6).

6.4. Influence of the exploitation exponent (β)

After highlighting the influence of the exploration exponent α on the performance of the allocation algorithm LAS, we investigate the influence of the exploitation exponent β. β reflects the importance dedicated to the pheromone amount assigned to each task in defining the number of robots to allocate to that task. The pheromone is allocated by the algorithm in response to the estimated averages of (i) the item's seeking time and (ii) the avoidance time associated with each task, see Eq. (7). Therefore, for a better investigation of the influence of the β parameter, we consider for this set of experiments dynamic environments, in which the density of items change over time, while the robots are executing new items. Each time a robot executes an item, this item disappears from the task's site, hence the density of items decreases over time. The task parameters are shown in Tab. 3, in which the execution time of single items is set much higher than in the experiments conducted in Sec. 6.3. These special settings is performed to avoid having all the items of a particular task disappearing immediately during the first execution phase of the assigned robots (items disappear when robots execute them). Setting the execution time of single items higher implicates setting the deadlines of the tasks higher as well (in comparison to the tasks' deadlines in Sec. 6.3).

We use the function in Eq. (8) to predict the change in the optimal number of robots over time, and we depict this change in Fig. 9. In the same figure, we can notice the number of robots assigned by LAS with $\alpha = 1.0$ and $\beta = 1.5$. In spite of assigning the highest number of robots to the blue task (the one with the highest initial priority), however, the synchronization periods performed before its deadline is reached were not enough to allow coping with the dynamic change of the seeking and avoiding times. Differently, for tasks with longer deadlines (e.g., the red, green, and yellow tasks) the allocated number of robots hits and even exceeds the current optimal number for robots over most of the experiment time. Remarkably, the priority in assigning robots to the different tasks stays in alignment with the tasks' initial priority.

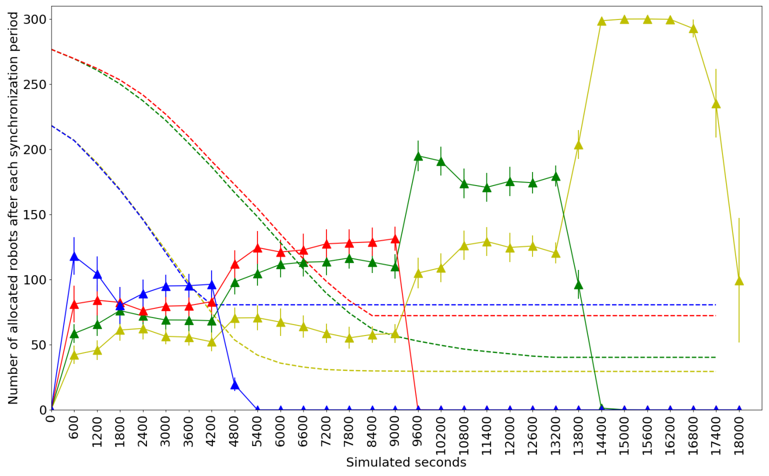

Figure 9: The number of allocated robots to the different tasks for $\alpha = 1.0$ and $\beta = 1.5$ in comparison to the optimal number computed under spatial interferences. Experiments performed in a dynamic environment (i.e. density of items decreases over time while robots are executing new items).

We have ran our experiments for the same task set using three β values (i) low influence $\beta = 0.5$, (ii) high influence $\beta = 1.5$, and (iii) extremely high influence $\beta = 5.0$, all with $\alpha = 1.0$. Fig. 10 shows very similar behavior for the three values of β. The number of executed items for the three tested values of β is illustrated in Fig. 11. In this figure, we can notice that changing the β value doesn't have a remarkable influence on the performance of LAS, and hence on

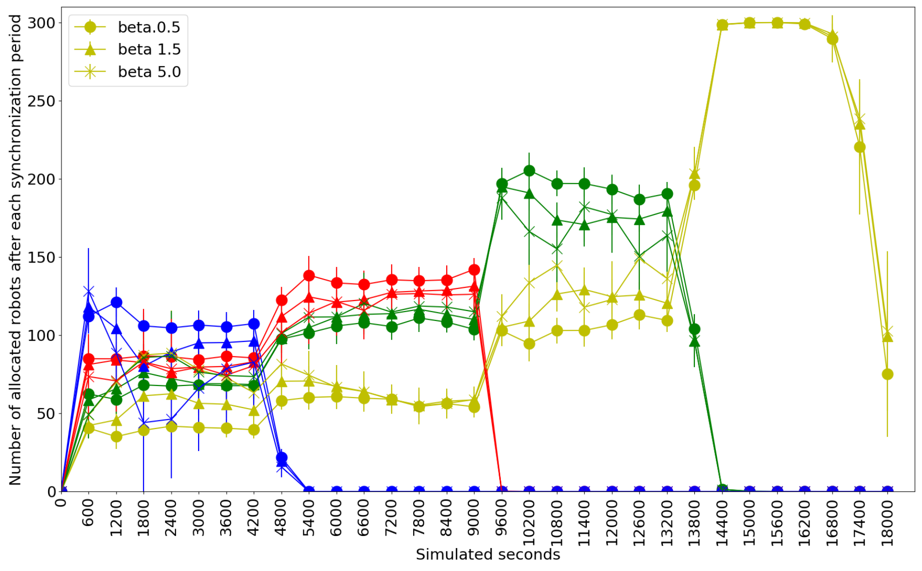

Figure 10: The number of allocated robots to the different tasks using 3 values of $\beta \in \{0.5, 1.5, 5.0\}$ and $\alpha = 1.0$, experiments performed in a dynamic environment (i.e. density of items decreases over time while robots are executing new items).

the number of items executed up to the task's deadline.

For further investigation of the influence of the β exponent, we have run experiments with different β values but also with different length of the execution phase at the different tasks, and consequently for a different number of synchronization periods included within the task's deadlines. we have concluded that tasks with relatively short deadline benefit from long execution phases and low β values. This is because, for such tasks, it is better for the robots to spend the time in executing items rather than in performing synchronization period to tune their allocation using their short experiences to update the pheromone trails. Whereas, for tasks with relatively long deadlines, a higher number of synchronization periods and shorter execution phases are of a higher benefit and hence large β value can be useful. This hypothesis was verified by the results captured in Fig. 12 and Fig. 13, which show the number of executed items of the four tasks for $\beta = 1.5$ and $\beta = 10.0$, respectively. We have changed the duration of the execution phases to constants and set them to $\{5000, 10000, 15000\}$

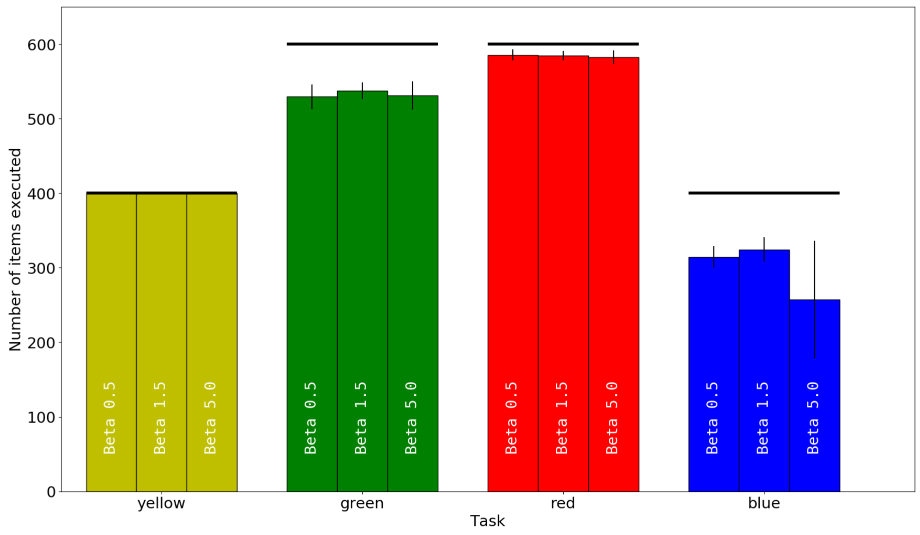

Figure 11: The number of items executed of each of the 4 tasks using $\beta \in \{0.5, 1.5, 5.0\}$ and $\alpha = 1.0$. Experiments performed in a dynamic environment (i.e. density of items decreases over time while robots are executing new items).

simulated tick—i.e. 0.1 of the simulated second. In Fig. 13, we can see how for tasks with longer deadlines such as the red or the green tasks, shorter execution phases (i.e. 1000 ts or 5000 ts) can lead with higher β values (i.e. $\beta = 10.0$) to larger number of executed items.

7. Conclusion

In this paper, we have proposed an efficient task allocation algorithms (LAS) for robot swarms to execute tasks under specific time constraints. The allocation algorithm extends the well-known Ant Colony Optimization (ACO) for a local computation and update of the pheromone trails using the experiences collected by the individual robots while executing their tasks. To our best knowledge, this was the first approach to implement the ant system on robot swarms to optimize their task allocation. Our results demonstrate a high benefit particularly in cases where (i) the size of the swarm is smaller than the summation of the optimal number of robots under physical interference over all tasks, or (ii) for cases where

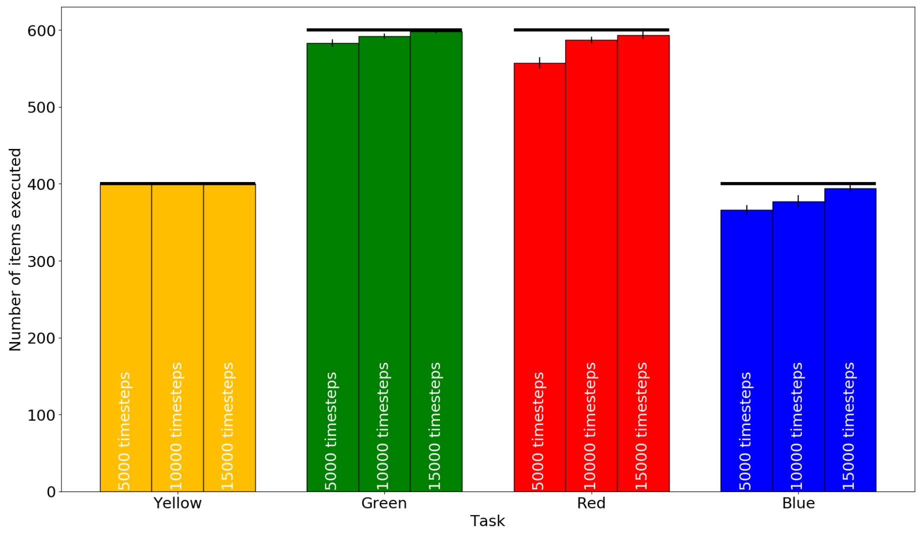

Figure 12: Number of executed items over the four tasks with $\beta = 1.5$.

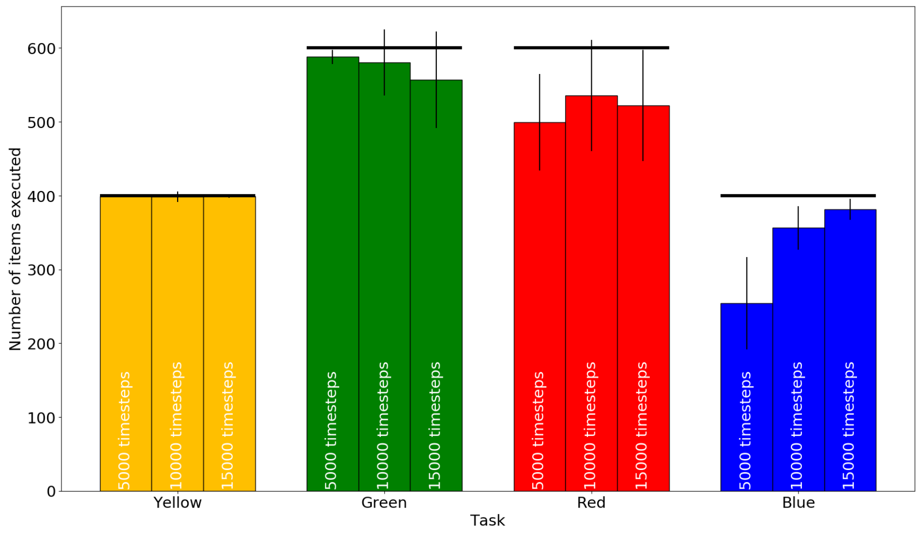

Figure 13: Number of executed items over the four tasks with $\beta = 10.0$.

environments are dynamic, i.e. the optimal number of robots changes over time as a result of e.g., a changing item density (i.e., task availability). As a future work, it would be of a high interest to implement individual mechanisms (at the robot's level) to distinguish static from dynamic environments, and hence decides individually whether to increase the weight of the exploration exponent (α) or the exploitation one (β) for achieving a more efficient allocation.

[1] M. Bakhshipour, M. J. Ghadi, F. Namdari, Swarm robotics search & rescue: A novel artificial intelligence-inspired optimization approach, Applied Soft Computing 57 (2017) 708–726.

[2] Y. Khaluf, Edge detection in static and dynamic environments using robot swarms, in: 2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), IEEE, 2017, pp. 81–90.

[3] H. Hamann, Scenarios of swarm robotics, in: Swarm Robotics: A Formal Approach, Springer, 2018, pp. 65–93.

[4] D. Albani, J. IJsselmuiden, R. Haken, V. Trianni, Monitoring and mapping with robot swarms for agricultural applications, in: Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on, IEEE, 2017, pp. 1–6.

[5] K. Lerman, A. Galstyan, Mathematical model of foraging in a group of robots: Effect of interference, Autonomous Robots 13 (2) (2002) 127–141.

[6] W. Liu, A. Winfield, J. Sa, J. Chen, L. Dou, Towards energy optimization: Emergent task allocation in a swarm of foraging robots, Adaptive Behavior 15 (3) (2007) 289–305.

[7] T. Labella, M. Dorigo, J. Deneubourg, Efficiency and task allocation in prey retrieval, in: Biologically Inspired Approaches to Advanced Information Technology, Springer, 2004, pp. 274–289.

[8] G. Pini, A. Brutschy, C. Pinciroli, M. Dorigo, M. Birattari, Autonomous task partitioning in robot foraging: an approach based on cost estimation, Adaptive behavior 21 (2) (2013) 118–136.

[9] A. Turgut, H. Çelikkanat, F. Gökçe, E. Şahin, Self-organized flocking in mobile robot swarms, Swarm Intelligence 2 (2-4) (2008) 97–120.

[10] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, S. Nolfi, Self-organized coordinated motion in groups of physically connected robots, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 37 (1) (2007) 224–239.

[11] O. Soysal, E. Şahin, A macroscopic model for self-organized aggregation in swarm robotic systems, in: Swarm robotics, Springer, 2007, pp. 27–42.

[12] V. Trianni, R. Groß, T. Labella, E. Şahin, M. Dorigo, Evolving aggregation behaviors in a swarm of robots, in: Advances in artificial life, Springer, 2003, pp. 865–874.

[13] Y. Khaluf, M. Birattari, F. Rammig, Analysis of long-term swarm performance based on short-term experiments, Soft Computing 20 (1) (2016) 37–48.

[14] S. Berman, Á. Halász, M. Hsieh, V. Kumar, Optimized stochastic policies for task allocation in swarms of robots, Robotics, IEEE Transactions on 25 (4) (2009) 927–937.

[15] Y. Khaluf, M. Birattari, H. Hamann, A swarm robotics approach to task allocation under soft deadlines and negligible switching costs, in: From Animals to Animats 13, Springer, 2014, pp. 270–279.

[16] H. Hamann, G. Valentini, Y. Khaluf, M. Dorigo, Derivation of a micro-macro link for collective decision-making systems: Uncover network features based on drift measurements, in: 13th International Conference on Parallel Problem Solving from Nature (PPSN 2014), Ljubljana, Slovenia, Springer, 2014, pp. 181–190.

[17] Y. Khaluf, F. Rammig, Task allocation strategy for time-constrained tasks in robot swarms, in: Advances in Artificial Life, ECAL, Vol. 12, 2013, pp. 737–744.

[18] Y. Khaluf, M. Dorigo, Modeling robot swarms using integrals of birth-death processes, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 11 (2) (2016) 8.

[19] M. Dorigo, Optimization, learning and natural algorithms (in Italian), Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy (1992).

[20] M. Dorigo, T. Stützle, Ant Colony Optimization, Bradford Company, Scituate, MA, USA, 2004.

[21] M. Dorigo, C. Blum, Ant colony optimization theory: A survey, Theoretical computer science 344 (2-3) (2005) 243–278.

[22] T. Stützle, M. López-Ibánez, P. Pellegrini, M. Maur, M. M. De Oca, M. Birattari, M. Dorigo, Parameter adaptation in ant colony optimization, in: Autonomous search, Springer, 2011, pp. 191–215.

[23] T. Stützle, M. López-Ibáñez, Automated design of metaheuristic algorithms, in: Handbook of Metaheuristics, Springer, 2019, pp. 541–579.

[24] D. Martens, M. D. Backer, R. Haesen, J. Vanthienen, M. Snoeck, B. Baesens, Classification with ant colony optimization, IEEE Transactions on Evolutionary Computation 11 (5) (2007) 651–665.

[25] C. Blum, Beam-aco: Hybridizing ant colony optimization with beam search: An application to open shop scheduling, Comput. Oper. Res. 32 (6) (2005) 1565–1591.

[26] P. Toth, D. Vigo, Models, relaxations and exact approaches for the capacitated vehicle routing problem, Discrete Applied Mathematics 123 (1) (2002) 487–512.

[27] J. Belenguer, E. Benavent, A cutting plane algorithm for the capacitated arc routing problem, Computers & Operations Research 30 (5) (2003) 705–728.

[28] H. Ramalhinho, D. Serra, Adaptive search heuristics for the generalized assignment problem, Mathware & soft computing 9 (3) (2008) 209–234.

[29] M. Yagiura, T. Ibaraki, F. Glover, An ejection chain approach for the generalized assignment problem, INFORMS Journal on Computing 16 (2) (2004) 133–151.

[30] D. Albani, D. Nardi, V. Trianni, Field coverage and weed mapping by uav swarms, in: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, IEEE, 2017, pp. 4319–4325.

[31] T. Wareham, A. Vardy, Putting it together: The computational complexity of designing robot controllers and environments for distributed construction, Swarm Intelligence 12 (2) (2018) 111–128.

[32] S. Wilson, T. P. Pavlic, G. P. Kumar, A. Buffin, S. C. Pratt, S. Berman, Design of ant-inspired stochastic control policies for collective transport by robotic swarms, Swarm Intelligence 8 (4) (2014) 303–327.

[33] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: An autocatalytic optimizing process, Tech. rep., Technical report (1991).

[34] G. Theraulaz, E. Bonabeau, J. Denuebourg, Response threshold reinforcements and division of labour in insect societies, Proceedings of the Royal Society of London B: Biological Sciences 265 (1393) (1998) 327–332.

[35] E. Ferrante, A. E. Turgut, E. Duéñez-Guzmán, M. Dorigo, T. Wenseleers, Evolution of self-organized task specialization in robot swarms, PLoS computational biology 11 (8) (2015) e1004273.

[36] E. Castello, T. Yamamoto, F. Dalla Libera, W. Liu, A. F. Winfield, Y. Nakamura, H. Ishiguro, Adaptive foraging for simulated and real robotic swarms: the dynamical response threshold approach, Swarm Intelligence 10 (1) (2016) 1–31.

[37] A. Kanakia, B. Touri, N. Correll, Modeling multi-robot task allocation with limited information as global game, Swarm Intelligence 10 (2) (2016) 147–160.

[38] J. Schwarzrock, I. Zacarias, A. L. Bazzan, R. Q. de Araujo Fernandes, L. H. Moreira, E. P. de Freitas, Solving task allocation problem in multi unmanned aerial vehicles systems using swarm intelligence, Engineering Applications of Artificial Intelligence 72 (2018) 10–20.

[39] E. O. Wilson, The relation between caste ratios and division of labor in the ant genus pheidole (hymenoptera: Formicidae), Behavioral Ecology and Sociobiology 16 (1) (1984) 89–98.

[40] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.