Auton Agent Multi-Agent Syst (2014) 28:101–125
DOI 10.1007/s10458-012-9212-y

Self-organized task allocation to sequentially interdependent tasks in swarm robotics

Arne Brutschy · Giovanni Pini · Carlo Pinciroli · Mauro Birattari · Marco Dorigo

Published online: 27 December 2012

Abstract In this article we present a self-organized method for allocating the individuals of a robot swarm to tasks that are sequentially interdependent. Tasks that are sequentially interdependent are common in natural and artificial systems. The proposed method does neither rely on global knowledge nor centralized components. Moreover, it does not require the robots to communicate. The method is based on the delay experienced by the robots working on one subtask when waiting for input from another subtask. We explore the capabilities of the method in different simulated environments. Additionally, we evaluate the method in a proof-of-concept experiment using real robots. We show that the method allows a swarm to reach a near-optimal allocation in the studied environments, can easily be transferred to a real robot setting, and is adaptive to changes in the properties of the tasks such as their duration. Finally, we show that the ideal setting of the parameters of the method does not depend on the properties of the environment.

Keywords Swarm robotics · Foraging · Self-organization · Task allocation · Swarm intelligence · Multi-agent systems

1 Introduction

In this article we present a self-organized method for allocating the individuals of a robot swarm to tasks that are *sequentially interdependent*. Tasks that are sequentially interdependent are common—in real-world systems, tasks often exhibit interdependencies. For example, many strategies of division of labor partition complex tasks into simpler subtasks that exhibit dependencies between each other [4]. In nature, interdependent subtasks can be found in social insects, which are known to have the ability of subdividing complex tasks into subtasks [35]. The most common interdependency is sequential: subtasks must be completed one after

A. Brutschy (✉) · G. Pini · C. Pinciroli · M. Birattari · M. Dorigo
IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
e-mail: arne.brutschy@ulb.ac.be

the other in order to complete the overall task once. An example of interdependent subtasks can be found in the leaf-cutter ant *Atta sextens*, which forages for leaves by first cutting the leaves from the tree, and second transporting the leaves previously cut to the nest [18]. The sequential dependency resides in the fact that the cutting of the leaves must be performed before they can be transported. Interdependent subtasks present a task allocation problem for the swarm: how should the individuals of the swarm allocate to the different subtasks in order to maximize the performance? Social insects employ self-organized strategies in order to tackle task allocation problems of this kind (see [35] for examples of such strategies).

Despite the common occurrence of interdependent tasks, task allocation strategies employed in artificial systems rarely consider tasks that exhibit interdependencies, much less a sequential interdependency. Most of the task allocation algorithms proposed in the swarm robotics literature assume independent tasks and do not take group dynamics between subtasks into consideration [11], with the notable exception of the work published by Nouyan et al. [30].

We propose a novel method for allocating robots to sequentially interdependent tasks. The method is proposed in the context of swarm robotics, and therefore relies only on the local interactions and perceptions of the individual robots. Additionally, the method does not require the robots to communicate. In contrast to most traditional, market-based approaches to distributed task allocation problems [13], the proposed method does not employ complex coordination schemes to allocate robots to tasks. Instead, the allocation is the result of a self-organized process that originate from the local decisions made by each robot of the swarm. *Self-organized task allocation* has certain advantages over traditional methods, most notably in terms of robustness to failures in communication [24].

The task we use as a testbed for this article is a foraging task that consists of two subtasks: *harvest* and *store*. Each robot has to decide which of the two subtasks to perform, depending on the performance of the swarm and on its perception of the environment. The foraging task is illustrated in Fig. 1 using a snapshot of the video recording of the real robot experiment presented in this article.

The rationale behind the method proposed in this article is that the maximum performance of a sequence of interdependent subtasks can be reached when the allocation of robots to subtasks is optimal. Therefore, robots should assess the current allocation and switch between subtasks in order to reach the optimal allocation. Robots using the proposed method base their decision to switch between subtasks on the delay they experience when waiting for

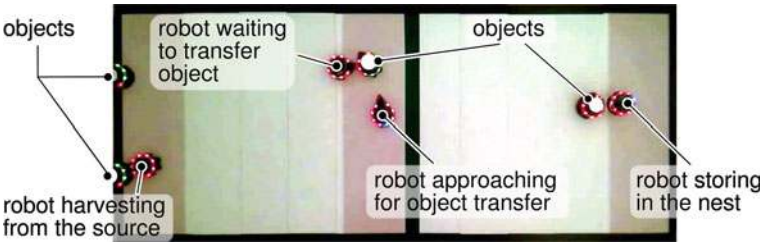

Fig. 1 Snapshot of the video recording of the real robot experiment. The overall task of the robots is to harvest objects from the source (*left*) and store them in the nest (*right*). This overall task consists of two subtasks: *harvest* and *store*. In the area located in the center, the robots working on the *harvest* subtask can transfer objects to the robots working on the *store* subtask. In this artice, we consider the problem of finding the optimal allocation of robots to such a sequence of subtasks. Objects are represented by robots marked by a *white circle* on *top*

robots working on another subtask. We evaluate the performance of a swarm using the proposed method in extensive simulation experiments. Additionally, we evaluate the method in a proof-of-concept experiment using real robots.

The article is structured as follows. In Sect. 2, we review related work and existing studies in the domain of task allocation. In Sect. 3, we describe the problem of task allocation with sequentially interdependent tasks and the method that we propose for tackling this problem in a self-organized manner. In Sect. 4, we present the experimental framework we use to evaluate the proposed method: the chosen problem instance, a foraging task, and the robots. In Sects. 5 and 6, we report the results of the simulation and real robot experiments, respectively. In Sect. 7, we conclude the article and discuss some possible future research directions.

2 Related work

Task allocation is inherently considered in many robotics and artificial intelligence applications. Thus, it is not surprising that the definitions of what task allocation actually is are as diverse as the research directions present in the literature. For the purpose of this article we will make the following distinction: works using *orchestrated* task allocation and works using *self-organized* task allocation. In the following, we briefly discuss the differences between these two classes of approaches.

Classical robotics and artificial intelligence traditionally regard task allocation as an optimization problem. Usually, one or more robots assess the problem at hand and derive a plan for allocating robots to tasks, using either centralized or decentralized methods. This plan is subsequently communicated to all robots and executed, either once at the beginning of the mission or incrementally during the course of the mission [19]. An example are market-based methods which employ auctions to allocate robots to tasks [13,21]. We refer to this type of task allocation as *orchestrated* task allocation.[1] As such methods are based on active negotiation among robots, they usually have strong communication requirements. A thorough discussion of the work on orchestrated task allocation is outside the scope of this article. The interested reader can find a taxonomy that covers orchestrated task allocation methods in [20].

Task allocation methods proposed in the context of swarm intelligence, on the contrary, usually do not rely on hierarchies, preassigned roles, or explicit communication. Instead, allocations at the level of the swarm result from local, stochastic decisions of the individual robots. We refer to this kind of methods as *self-organized* task allocation methods. The amount of works in self-organized task allocation is considerably lower when compared to works in orchestrated task allocation, and, to the best of our knowledge, most works in self-organized task allocation tackle simple problems without task interdependencies [11].

Most works that study self-organized task allocation employ threshold-based methods, taking inspiration from models initially proposed to describe the behavior of insect societies [6]. While simple methods use fixed thresholds for triggering different behaviors (see, for example, Krieger and Billeter [25]), most works employ methods in which thresholds are adapted over time. Examples of methods using adaptive thresholds can be found in Labella et al. [26], Campo and Dorigo [8], and Liu et al. [27].

A common formalization of adaptive thresholds are the so-called *reinforced response threshold* models, which have initially been proposed for modeling the behavior of insect

[1] Also referred to as *intentional* approaches to task allocation [24].

societies as well [37]. In robotics, methods based on these models have been proposed, among others, by Ferreira et al. [17] and Ikemoto et al. [23]. Ferreira et al. [17] compare a reinforced response threshold method to a token-based method in a RoboCup rescue scenario, while Ikemoto et al. [23] use a reinforced response threshold method to obtain division of labor in a foraging task.

Kalra and Martinoli [24] compare an orchestrated, auction-based approach to a self-organized, threshold-based one. Their results indicate that the auction-based approach performs better than the threshold-based one when the information available to the individuals of the group about the tasks and the environment is accurate. On the other hand, when this information is not accurate, the threshold-based approach performs as well as the auction-based approach at a fraction of the expense in communication.

Agassounon and Martinoli [1] study the problem of allocating robots to tasks that require close cooperation. In their "stick-pulling" experiment, two robots are required to cooperate closely in order to complete a task. The task allocation method proposed in their work is also based on thresholds, which are continuously adapted by the robots on the basis of their perception of their own performance.

Dahl et al. [11] presented a work that focuses on the group dynamics of task allocation. In their work, the authors propose a method inspired by vacancy chain scheduling, a resource distribution processes commonly found in real-world systems. The authors model the group dynamics of a swarm both at the microscopic and at the macroscopic level, and argue that by using these two levels, their method can reason about the global performance of the group as well as make decisions at the individual level.

A work that employs a purely macroscopic model to form coalitions has been presented by Berman et al. [5]. In their work, robots of a swarm are allocated to multiple surveillance tasks. However, the proposed method is not strictly decentralized, as it employs a central controller to regularly broadcast information about task transitions to all robots.

Few works in the literature tackle the problem of allocation to tasks with sequential interdependencies in self-organized systems. Scheidler et al. [36] study a two-task partitioning problem in an organic computing setup, conceptually similar to the problem presented in this article. They also use a threshold-based method to decide on the allocation of reconfigurable computing units.

Dasgupta [12] studies a system in which robots can only partially complete tasks that are available in the environment. If a robot has completed his share of a task, it communicates the location and the progress of the task to other robots in the group, which then continue to work on the task until it is completed. Thus, robots have to collaborate to perform tasks, and this type of collaboration is sequential in nature, similar to the "stick-pulling" presented by Agassounon and Martinoli [1].

Pini et al. [33] present a work that focuses on the impact of interference on swarm performance and allocation in a setting similar to the one studied in this article; however, the focus of the work was not on task allocation methods. In a closely related work, Pini et al. [34] study the problem of collectively deciding in which cases it is advantageous to partition a task into sequentially dependent subtasks.

3 Problem statement and proposed method

In this section, we describe in detail the problem of task allocation in sequentially interdependent tasks that we consider in this paper and the method that we propose for tackling it in a self-organized and decentralized way.

3.1 Problem statement

We assume that a task $\mathcal{T}$ is composed of two *subtasks* τ_1 and τ_2. The whole task $\mathcal{T}$ is completed if both subtasks have been completed. We call these subtasks *sequentially interdependent* if one subtask must be completed before the start of the other. In this case, we refer to the whole task $\mathcal{T}$ as a *task sequence*. We define a subtask τ_1 to be the *predecessor* of τ_2 if τ_1 must be executed before τ_2. Analogously, τ_2 is the *successor* of τ_1. The ordering relation between the two subtasks is represented with $\tau_1 \succ \tau_2$. We refer to these subtasks as being *adjacent*. Two adjacent subtasks share a *task interface*, where robots working on different subtasks exchange task-related information or objects and where they can decide whether to change type of subtask on which to work. We refer to the robots working on a subtask τ_i as *group* g_i, with $|g_1| + |g_2| = N_1 + N_2 = N$ being the numbers of robots working in each group g_i and in the swarm, respectively. Each robot works on one of the two subtasks, that is, there are no idle robots.

As an example, let us consider a foraging swarm. A foraging swarm typically searches for objects (e.g., food items or building material) present in the environment. Subsequently, the swarm transports the objects to the nest, where they are stored. We can consider the task of foraging to be composed of two subtasks, harvesting and storing. The two subtasks exhibit a sequential interdependency: robots working on the storing subtask depend on the robots working on the predecessor subtask, harvesting, because objects need to be harvested before they can be stored.

Figure 2 illustrates such a foraging task that consists of two subtasks. The foraging task can be tackled in two different ways. One way is to let the robots work as a single group (see Fig. 2, top). In this case, when a robot gets to the interface it necessarily switches to the other subtask. The second way is to tackle the foraging task by employing two different groups of robots, each group taking care of one of the two subtasks (see Fig. 2, center and bottom). When robots arrive at the task interface, they have to decide whether to continue to work on the same subtask, or to switch to the other one. If they decide to continue to work on the same subtask, they must transfer the carried object to a robot performing the other subtask.

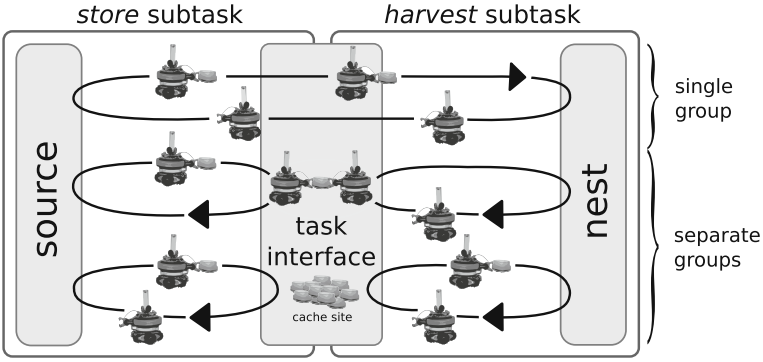

Fig. 2 Possible ways of tackling a foraging task with a group of four robots. *Top* tackling the task in its monolithic form using a single group of robots (i.e., robots always perform both subtasks by switching immediately between them); *middle* tackling the two subtasks separately by using different groups of robots for each subtask with direct transfer of objects at the *task interface*; *bottom* tackling the two subtasks separately, with indirect transfer of objects at the *task interface* by using a *cache site*. In this article, we consider the case in which objects are transferred directly

Objects can be transferred between robots by direct transfer from robot to robot, or indirectly, by using a cache site. In this paper we consider the first case.

The performance of the swarm depends on the number of robots in each group g_i working on the two subtasks $\tau_i, i \in \{1, 2\}$. We refer to the assignment of the N robots of the swarm to the two groups g_i as *allocation*. Different allocations yield different performances at the swarm level. There exists an allocation of robots to subtasks such that the overall performance of the swarm is maximized. We refer to this allocation as *optimal allocation*. The optimal allocation depends on the nature of the subtasks and of the environment. In dynamically changing conditions, it can vary in time. Our problem is how to let the robots allocate to tasks in an optimal or near-optimal way.

Robots switching between subtasks must pay a *task switching cost* c_s. The task switching cost depends on the environment and on the nature of the task interface. The task switching cost is commonly a time-cost, even though other types of costs have been studied in the literature (see Cicirello and Smith [10] for an example of a system with monetary costs). Please note that robots always have to pay the task switching cost regardless of whether they tackle the subtask as a single group or as two groups. Common examples of task switching costs in real systems are costs due to tool changes, spatially dispersed subtasks, or inherent costs due to specialization in a subtask (or the lack thereof).

Whether it is more efficient to perform the task using a single group of robots or splitting it over two groups depends on the value of c_s. Pini et al. [34] have proposed a method to let the robots choose between the two alternatives.

In this article, we assume a task switching cost sufficiently high so that tackling the task with two groups of robots is the best choice, and we focus on the problem of how to allocate the robots to the two groups.

3.2 Proposed method

In the following we describe the method proposed in this article. We first present a high-level description of the method, followed by a detailed description of its implementation.

3.2.1 High-level description

The goal of the method is to determine an allocation of robots to subtasks that maximizes the performance of the swarm in terms of number of objects retrieved per time unit. The method we propose does not require the robots to have global knowledge of the task or of the environment. Additionally, the robots do not need to communicate explicitly. The proposed method does not rely on a central controller; each robot autonomously decides whether and when to switch between subtasks using the information available locally. As a result, the global allocation results from the behavior of the individual robots in a self-organized manner.

The rationale behind the proposed method is that the overall performance for the task sequence $\mathcal{T}$ is optimal when the swarm uses an allocation of robots to subtasks that is optimal. In order to reach the optimal allocation, the robots of the swarm should assess the current allocation and switch between subtasks if necessary.

As mentioned earlier, we consider the case in which object transfer at the task interface is direct. A robot that arrives at the task interface has to wait for a transfer partner before it can return to work on its current subtask. The time the robot has to wait depends on the number of robots working on the other subtask. The optimal allocation is such that in

neither subtask there is an accumulation of robots waiting for the robots working on the other subtask. In other words, robots working on the two subtasks have to arrive at the task interface at an approximately equal rate to maximize the overall performance. Therefore, the method proposed in this article relies on equalizing the arrival rates of the robots at the task interface.

The robots estimate the arrival rates at the task interface by measuring the waiting time at the task interface for both subtasks. We refer to the waiting time as the *interface delay* d_{ij}, which is defined as the delay a robot working on subtask τ_i experiences when waiting at the task interface with the adjacent subtask τ_j.[2] In a foraging task in which robots directly transfer objects, the interface delay consists usually of the time spent waiting for a transfer partner.

If robots of the two groups arrive at an equal rate at the task interface, the interface delays experienced by the two groups of robots working on the two subtasks are approximately equal. If this is not the case, the interface delays of the two groups can be equalized by transferring part of the workforce from the group that experiences higher interface delay to the other group. To this end, a robot working in group g_i should switch to the adjacent group g_j if the interface delay experienced by robots in g_i is higher than the one experienced by robots in g_j. Hence, in the proposed method, robots decide whether to switch from one subtask to the other as a function of the interface delays experienced in the two subtasks.

To implement the task switching behavior, each robot maintains an average of the interface delays it experienced in both subtasks. The average interface delay of a robot working on subtask τ_i that is waiting for the robots of subtask τ_j is denoted by $\hat{d}_{ij}$, while the average for the other subtask is denoted by $\hat{d}_{ji}$. The average interface delays give the robots an estimate about arrival rate of their own group at the task interface relative to the arrival rate of the other group. The proposed method is characterized by the cumulative distribution $\mathcal{P}_{ij}(d_{ij})$. This cumulative distribution $\mathcal{P}_{ij}(d_{ij})$ is the probability that a robots switches from subtask τ_i to subtask τ_j before waiting d_{ij} seconds at the task interface (analogously for $\mathcal{P}_{ji}$). In other words, $1 - \mathcal{P}_{ij}(d_{ij})$ is the probability that a robot is waiting d_{ij} seconds at the task interface.

$\mathcal{P}_{ij}(d_{ij})$ is a parametric function of the current interface delay d_{ij}. The parameters are the average interface delays $\hat{d}_{ij}$ and $\hat{d}_{ji}$. Formally, $\mathcal{P}_{ij}(d_{ij}) = \mathcal{P}_{ij}(d_{ij}; \hat{d}_{ij}, \hat{d}_{ji})$. In the following, for the sake of brevity, we will simply write $\mathcal{P}_{ij}(d_{ij})$.

Figure 3a shows $\mathcal{P}_{ij}(d_{ij})$ for several situations in which the average interface delays of the two groups are equal.[3] The plot shows that the method adapts $\mathcal{P}_{ij}(d_{ij})$ so that the robots switch at low interface delays if the average interface delay are low as well (see Fig. 3a, $\hat{d}_{ij} = \hat{d}_{ji} = 50\,\text{s}$). On the other hand, in situations in which the average interface delays are high, robots switch at higher interface delays d_{ij} (see Fig. 3a, $\hat{d}_{ij} = \hat{d}_{ji} = 100\,\text{s}$). As the proposed method is independent of the absolute values of the subtask durations and delays, it can be applied to problem instances that exhibit different task durations or swarm sizes without the necessity of adaptation of the parameters.

In case the average interface delays of the two groups are not equal (see Fig. 3b for an example), the method adapts the cumulative switching probability so that the robots with the lower average interface delay are less likely to switch than the robots with the higher average interface delay, at a given interface delay d_{ij}. This asymmetry causes more robots to switch

[2] In biology, the interface delay is commonly referred to as "queuing delay" (cf. [2,3]). We do not use the term "queue" as it implies an order in the arrival of robots at the task interface, which is not present in the stochastic system that we consider.

[3] Differences in average interface delays can be caused by differences in the properties of the problem such as subtask duration or swarm size.

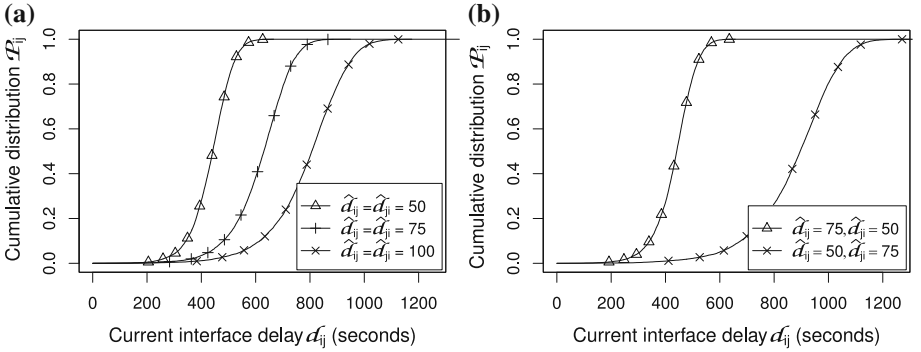

Fig. 3 The cumulative distribution of the switching probability $\mathcal{P}_{ij}(\hat{d}_{ij})$ for a robot working on subtask τ_i, evaluated using 10,000 Monte-Carlo experiments for a rising interface delay d_{ij}. **a** The switching probability is a function of the average interface delays $\hat{d}_{ij}$ and $\hat{d}_{ji}$. **b** The robots use a different switching probability depending on the average interface delays they experienced in the two subtasks: in case the average interface delay experienced by the robots working on subtask τ_i is higher than the one in the adjacent subtask, the robots are more likely to switch to subtask τ_j (values obtained using Eq. 2 with $m = 10$ and $k = 1$)

from the group with the higher average interface delay to the group with the lower average interface delay, thereby equalizing the arrival rates of both groups.

3.2.2 Implementation

In the following, we assume that τ_i is the current subtask of the robot. The robots used in this article make decisions at discrete points in time. The frequency with which decisions are made is the frequency ν of the control cycle of the robots. For a robot, time is discrete: $d = d/\nu$, with d being an integer that counts the number of control cycles that the robot has spent waiting at the task interface. At each control cycle, a robot switches from subtask τ_i to subtask τ_j with a probability p_{ij}. In the following, we refer to p_{ij} as *switching probability*. The relationship between $\mathcal{P}_{ij}$ and p_{ij} is as follows:

$$\mathcal{P}_{ij}(d_{ij}; \hat{d}_{ij}, \hat{d}_{ji}) = 1 - \prod_{q=1}^{d_{ij}} \left(1 - p_{ij}(q; \hat{d}_{ij}, \hat{d}_{ji})\right) \tag{1}$$

In words, $\mathcal{P}_{ij}$ is one minus the probability that the robot does not switch subtask the first d_{ij} control cycles. In our implementation, p_{ij} is defined by the following sigmoid function:

$$p_{ij}(d_{ij}; \hat{d}_{ij}, \hat{d}_{ji}) = \frac{1}{1 + e^{-\theta(d_{ij}; \hat{d}_{ij}, \hat{d}_{ji})}} \cdot \gamma \tag{2}$$

where γ is a parameter that is specified in the next section. The function θ is defined as:

$$\theta(d_{ij}; \hat{d}_{ij}, \hat{d}_{ji}) = \frac{1}{k}\left(\frac{d_{ij}}{r(\hat{d}_{ij}, \hat{d}_{ji})} - m\right) \tag{3}$$

where $r(\hat{d}_{ij}, \hat{d}_{ji})$ is a function of the average interface delays, which is explained in the following. Analogously to $\mathcal{P}_{ij}$, for the sake of brevity we use $p_{ij}(d_{ij})$ and $\theta(d_{ij})$ to refer to $p_{ij}(d_{ij}; \hat{d}_{ij}, \hat{d}_{ji})$ and $\theta(d_{ij}; \hat{d}_{ij}, \hat{d}_{ji})$, respectively.

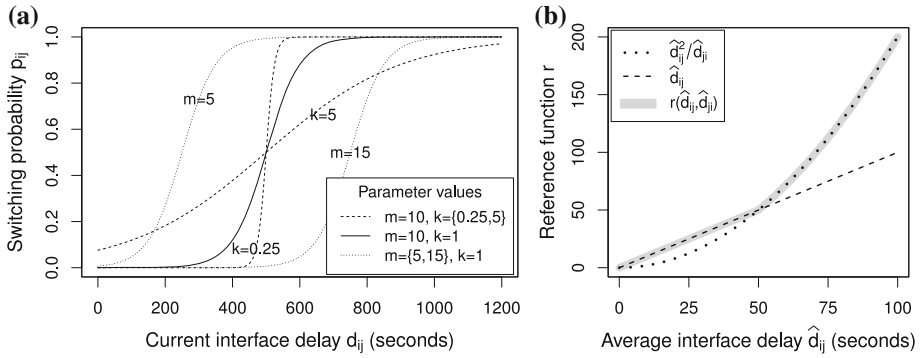

Fig. 4 The switching probability p_{ij} is a function of the discrete interface delay d_{ij} as defined by Eq. 2. **a** Resulting probability p_{ij} for different values for the parameters m and k, with $\hat{d}_{ij} = \hat{d}_{ij} = 50\,\text{s}$. **b** Behavior of the function r for different average interface delays $\hat{d}_{ij}$, with $\hat{d}_{ji} = 50\,\text{s}$

The average interface delays are computed as a weighted average with a fixed weight factor, $\hat{d}_{ij} = 0.2 \cdot \hat{d}_{ij} + 0.8 \cdot d_{ij}$. The average interface delays are updated whenever a robot completes a subtask. Due to the stochastic nature of the decision process, robots regularly switch between subtasks and update their estimates of the average interface delays. The average interface delays are initialized with values drawn from a random distribution that is specified in Sect. 5.1.

The two parameters m and k of Eq. 3 are used to adjust the switching behavior of the robot. The *shift* parameter m influences how quickly the robot reacts to an increasing interface delay d_{ij}. The higher the value of m, the more the S-shaped curve is shifted to the right. The *steepness* parameter k controls if the increase in switching probability p_{ij} for an increasing interface delay d_{ij} is gradual or immediate. The lower the value of k, the steeper the slope of the function p_{ij}. See Fig. 4a for a visual representation of the influence of the parameters m and k on the switching probability p_{ij}.

The function r used in Eq. 3 relates the two average interface delays to each other:

$$r(\hat{d}_{ij}, \hat{d}_{ji}) = \frac{\hat{d}_{ij} \cdot \max(\hat{d}_{ij}, \hat{d}_{ji})}{\hat{d}_{ji}} \tag{4}$$

$$= \begin{cases} \hat{d}_{ij}^2/\hat{d}_{ji} & \text{if } \hat{d}_{ij} > \hat{d}_{ji} \\ \hat{d}_{ij} & \text{if } \hat{d}_{ij} \leq \hat{d}_{ji} \end{cases} \tag{5}$$

The function r implements the asymmetry mentioned earlier: In case the group g_j has a lower average interface delay than g_i, that is $\hat{d}_{ij} > \hat{d}_{ji}$, r increases quadratically. This reinforces smaller differences in the average interface delays and causes robots to switch already at small interface delays d_{ij}. In the case of $\hat{d}_{ij} \leq \hat{d}_{ji}$, on the other hand, r increases linearly, which causes robots to switch at higher interface delays. This mechanism ensures that more robots switch from the group with the higher arrival rate at the task interface to the group with the lower arrival rate than the other way around. Figure 4b gives a visual representation of the behavior of function r for different average interface delays $\hat{d}_{ij}$.

Returning to the example of foraging, consider the case in which the group storing objects in the nest has a higher arrival rate at the task interface than the group harvesting objects from the source. In this case, the group storing objects experiences, on average, a higher interface

delay than the harvesting group. The function r amplifies this difference; the robots working on storing, therefore, have a higher probability to switch to the harvesting subtask. On the other hand, the harvesting group experiences, on average, a lower interface delay than the storing group. The function r causes the robots of this group to have a low, although non-zero, probability of switching to the storing subtask. Robots storing objects will therefore be more likely to switch between subtasks than robots that are harvesting. This will cause arrival rates of the two groups at the task interface to gradually equalize. Additionally, the switching probabilities of the two groups will equalize such that both exhibit few switches between subtasks.

4 Experimental framework

In the following, we introduce the experimental framework that we employ for validating the method presented in Sect. 3.2. As mentioned before, we carry out two types of experiments: simulation experiments and real robot experiments. In the following, we describe the problem instance studied and the experimental setup common to both types of experiments. Then, we discuss the metrics used to evaluate the proposed method, and we present the robot and its controller. Finally, we describe the simulation framework used.

4.1 Problem instance and experimental setup

As testbed we chose a foraging task in which the robots have to harvest objects from one location, the *source*, and store them in another location, the *nest*. All experiments use the same layout for the environment: the source is located on the left-hand side and the nest is located on the right-hand side of a rectangular space (see Fig. 5). Different ground colors mark the different locations in the environment. Additionally, a light source marks the location of the nest. The robots can sense the direction of the light source and use it to determine the direction of the nest. In order to study the influence of different subtask durations on the performance of the proposed method, we run experiments using environments with different dimensions.

The environment is partitioned into two areas by the task interface. We refer to the two areas as the *harvest area*, containing the *source*, and the *store area*, containing the *nest*. By partitioning the environment in two areas, the overall foraging task $\mathcal{T}$ is also partitioned

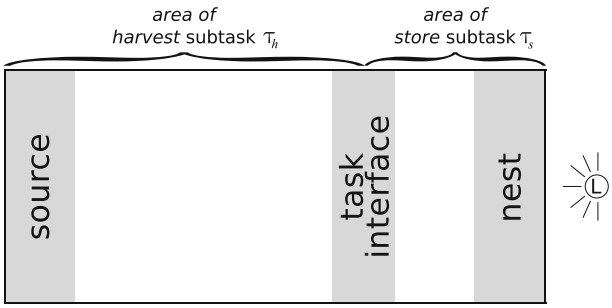

Fig. 5 Schematic representation of the environment used for the experiments. The environment is partitioned into two areas by the task interface. We refer to the two areas as the *harvest area*, containing the *source*, and the *store area*, containing the *nest*. Each area corresponds to one of the subtasks. The light source, used by the robots for navigation, is marked with "L"

into two subtasks: the *harvest subtask* τ_h and the *store subtask* τ_s. The harvest and the store subtasks have a sequential interdependency as they have to be performed one after the other in order to perform the overall task sequence $\mathcal{T} = \tau_h \succ \tau_s$ once: transporting an object from the source to the nest. Robots working on τ_h harvest objects in the source, transport them to the task interface and transfer them to the robots working on τ_s. The robots working on τ_s receive objects in the task interface and transport them to the nest, where they are stored. In the following, we call robots working in the group g_h on subtask τ_h *harvesters*, and we call robots working in the group g_s on subtask τ_s *storers*.

Since objects are transferred directly, a successful transfer requires both the passing and the receiving robot in the task interface at the same time. As mentioned, this setting corresponds to the second case shown in Fig. 2. Robots experience an interface delay d_{ij} when waiting for a transfer partner in the task interface. Robots can switch between subtasks by crossing the task interface. Upon switching, a robot has to wait for a given number of seconds c_s, which corresponds to the task switching cost introduced in Sect. 3.1. In this article, the task switching cost c_s is a parameter of the experiment; its abstract representation allows us to freely modify it and study its effect on the performance of a swarm using the proposed method.

In order to evaluate the performance of the proposed task allocation method, the knowledge of the optimal allocation of the swarm for each experimental setting is required a-priori. The optimal ratio for each environment is determined experimentally in the first experimental set described in Sect. 5.2. As stated in Sect. 3.1, the optimal allocation depends on the average duration e of the two subtasks. The average duration e_i of a subtask τ_i, in turn, depends on many factors—in this problem instance, among others, the travel times of the robots, the interface delays at the task interface, the time-costs of harvesting, storing or transferring an object and the losses due to interference or navigation errors. Thus, by modifying the position of the task interface in the environment, we can modify the ratio between the durations of the two subtasks and therefore the optimal allocation for the given experiment. In the following, we call *arena ratio* the ratio $e_s/(e_s + e_h)$, that is, the ratio between the duration e_s of the store subtask τ_s and the total duration $e_s + e_h$ of the task sequence.

4.2 Metrics

In this section, we describe the metrics that we use to evaluate the method presented. Besides evaluating the performance obtained by a swarm using the proposed method for task allocation, we also evaluate other metrics such as the number of robots per subtask and the number of times robots switch between subtasks.

The total number of objects collected by the robots, the number of storers N_s, and the number of harvesters N_h is recorded every 5 s. The total number of objects collected by the swarm at a given time is called the *swarm performance* P at that time. The swarm performance is used to evaluate the quality of the allocation of the robots to subtasks: the higher the swarm performance, the better the quality of the allocation.

We define P_{max_N} to be the maximum number of objects that can be harvested by a swarm of N robots. It indicates the maximum value that can be achieved when the task sequence $\mathcal{T}$ is tackled by two separate groups of robots and it is determined experimentally in the first experimental set described in Sect. 5.2.

We can express the swarm performance P relative to the maximal performance P_{max_N}, which allows us to compare performance values across different experimental conditions

(e.g., different arena ratios and swarm sizes). We refer to this metric as the *relative swarm performance* P/P_{max_N}.

We measure the allocation of robots to subtasks in terms of the *allocation ratio* R, which is computed as $R = N_s/N$, that is, R reports the fraction of robots working on the storing subtask τ_s. In order to assess the quality of the allocation obtained using the proposed method, we use the optimal ratio R_{opt} of robots as a reference value. The optimal ratio, defined as the allocation ratio at which the swarm reaches P_{max_N}, is determined experimentally in the first experimental set described in Sect. 5.2. We qualify a given allocation ratio R by using the mean absolute error *MAE*:

$$MAE = \frac{1}{M}\sum_{i=0}^{M}|R_t - R_{opt}|, \tag{6}$$

with R_t being the ratio value sampled at time t, and M being the total number of samples collected during the experiment. As the *MAE* is a cumulative error measure, smaller values indicate a ratio that is closer to the optimal ratio R_{opt}.

We also evaluate the number of times the robots switch between subtasks. As switching between subtasks might be costly, an efficient method should provide a stable allocation of robots to subtasks, with few robots switching between subtasks. To this end, we record the number of switches in a given time window. We denote this metric as $S_{\Delta t}$, with Δt being the length of the time window. The evolution of $S_{\Delta t}$ over time allows us to evaluate the allocation speed of the swarm: the sooner the robots cease to switch between subtasks, the faster the swarm reaches a stable allocation. We refer to the total number of task switches during the course of the whole experiment as S_{tot}.

4.3 The robot and its controller

The robot used for the experiments presented in this article is the *s-bot*, a robotic platform that has been developed within the *Swarm-bots* project[4] [14,28]. Figure 6 shows the s-bot and gives a description of its principal sensors and actuators. The robot is of round shape with a diameter of 12 cm. It features a gripper that can be used to connect to other robots and form assembled structures [9,16,22] or to transport objects [29]. The robot is equipped with 8 infra-red sensors, used to perceive obstacles up to a distance of 15 cm and the direction of a light source up to a distance of 5.0 m. Ground sensors positioned underneath the robot can be used for detecting ground color and holes. The robot features a transparent ring with 8 RGB LEDs that can be used to convey visual information. An omni-directional camera allows the robot to perceive objects and other robots up to a distance of about 60 cm. The s-bot can move in the environment by means of a differential drive system that combines tracks and wheels.

Each robot of the swarm is controlled by an instance of the same finite state machine controller, which is depicted in Fig. 7. The controller consists of two primary behaviors, each corresponding to one of the subtasks a robot can perform: either the harvest subtask τ_h or the store subtask τ_s. The behavior of each robot is defined by the subtask it is currently performing. Harvesters not carrying an object move towards the source, where they can harvest one. After harvesting an object, harvesters move to the task interface and search for an available storer. Upon the arrival of a storer, the harvester passes the object to it by direct

[4] http://www.swarm-bots.org/

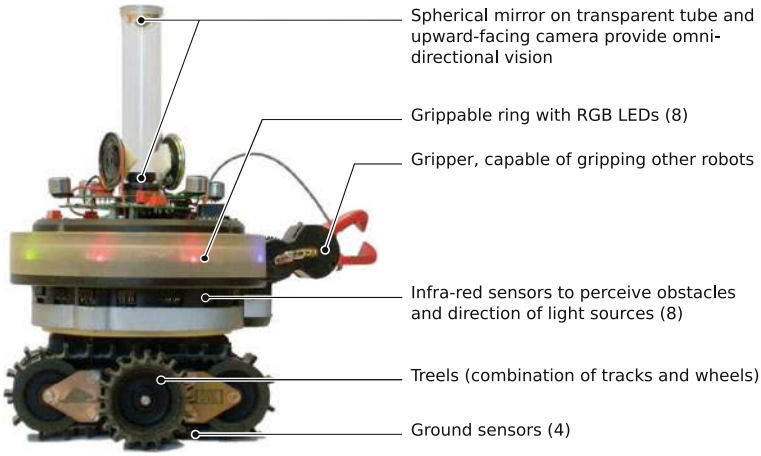

Fig. 6 The robot used in the experiments, the s-bot, shown with its principal sensors and actuators. Most distinctive is the gripper, which allows the robot to connect to other robots

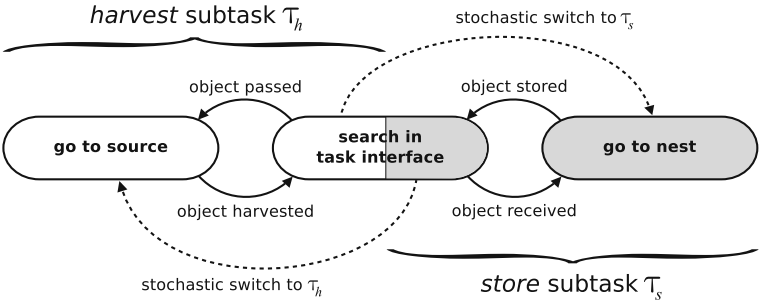

Fig. 7 Simplified state diagram of the controller of the robots. *White states* belong to the *harvest* subtask τ_h, while *gray states* to the *store* subtask τ_s. The search in task interface state is present for both subtasks. An obstacle avoidance state is present but has been omitted for clarity, as it is reachable from all states. The *dashed arrows* represent stochastic switches between subtasks (see Eq. 2)

transfer. Storers carrying an object move towards the nest, where they can store the object. Storers not carrying an object move to the task interface and search for a harvester ready to transfer an object. Robots can detect the location of objects and other robots by using their omni-directional camera, and the direction of the nest by using their infrared sensors. While moving, each robot avoids obstacles (walls, objects, and other robots).

The robots use the method introduced in Sect. 3.2 for deciding whether to switch from one subtask to the other. A harvester carrying an object can stochastically decide to become a storer, and a storer not carrying an object can stochastically decide to become a harvester (see Eq. 2). A robot has to be in the task interface in order to switch subtask. The duration of a control-cycle is 0.2 s in simulation, and 0.15 s on the real robot. The value of the parameter γ of Eq. 2 is 0.01 and 0.013 for simulated and real robots, respectively. After switching, a robot must wait for a certain time before it can continue to work on its new subtask. This time represents the task switching cost c_s.

4.4 The simulation framework

The simulation framework used to carry out the simulation experiments presented in this article is called ARGoS[5] [31,32]. ARGoS was developed within the Swarmanoid project[6] [15], and is an open-source simulation framework that was specifically designed to support the real-time simulation of large swarms of heterogeneous robots. The simulator that is part of the ARGoS framework is a discrete time, physics-based simulator, which allows the user to choose the desired level of detail of the simulation. The simulation detail can range from simple simulations with few details (e.g., two-dimensional environments governed by kinematic rules) to highly detailed and complex simulations (e.g., three-dimensional environments governed by dynamics). For the work presented in this article, we employ a kinematics model of the robots simulated in a two-dimensional environment. A uniform random noise of 10 % has been added to all sensor readings at each time-step.

While the logic related to working on and switching between subtasks is the same in simulation and real robot experiments, there are some substantial differences between simulation and real robots in the manner objects are represented. In simulation, objects are represented in an abstracted manner and not as physical objects. That is, objects are not simulated as embodied entities. Instead, the fact that a robot carries an object is stored internally in the simulator. This abstracted simulation allows us to omit the simulation of complex physics involved in gripping objects.

5 Simulation experiments

The goal of the simulation experiments is to evaluate the performance of the method proposed in this article in different experimental environments. In the following, we describe the experimental setup and the results obtained in the experiments.

5.1 Experimental setup

As mentioned in Sect. 4.1, all experiments are carried out in two rectangular environments. In the simulation experiments, we use the following parameters for the two environments. The dimensions of the environments are 4.5 m by 2.0 m. The nest and the source are each 0.3 m wide, while the task interface is 0.5 m wide. These locations are marked with a different ground color: the nest and the source are gray, the task interface is black, the harvest area is white, and the store area is light gray. The two environments differ for what concerns the position of the task interface. The first environment is partitioned by the task interface into two areas of equal size (2.25 m by 2.0 m), which results in an arena ratio of 0.5. We refer to this environment as the *symmetric environment*. The second environment is partitioned into two areas of different size (3.0 m by 2.0 m for the harvest area and 1.5 m by 2.0 m for the store area), which results in an arena ratio of 0.67. We refer to this environment as the *asymmetric environment*.

In all the experiments, we measure the time t in minutes unless specified otherwise. The robots of the swarm are randomly positioned in the nest area at time $t = 0$. The estimates of the average interface delays $\hat{d}_{hs}$ and $\hat{d}_{sh}$ are initialized with a random value uniformly sampled from the interval (0 s, 10 s]. The experiments run for $t_{tot} = 60$ min. For each experimental

[5] http://iridia.ulb.ac.be/argos/

[6] http://www.swarmanoid.org/

condition, we run 100 repetitions, each with a different seed for the pseudo random number generator.

5.2 Results and discussion

In this section, we present and discuss the results of the simulation experiments. We first determine the optimal allocation R_{opt} for each of environments. We then study the effect of the task switching cost c_s on the performance of the proposed method. Next, we study the parameters m and k of the method and their influence on the performance. Last, we study the scalability and adaptability of the method.

As the underlying distributions of the different metrics introduced in Sect. 4.2 are not known, we report each metric using a non-parametric approach. We report the quantiles of the distribution as follows. In the text, we represent the median (50 % quantile) and the 25–75 % quantile in the format 25/50/75 %. In the plots, we represent the 25–75 % quantile by a dark gray area, with the median represented as a black line inside the gray area. Where necessary, we additionally report the 1 and 99 % quantiles, represented in the format 1/25/50/75/99 % in the text and by a light gray area in the plots.

In the following, we report the results obtained in the asymmetric environment. This is because in most cases analogous conclusions can be drawn from the results obtained in the symmetric environment. When this is not the case, we report the results for both environments. The complete results are included in the supplementary on-line material [7].

5.2.1 Optimal allocation

The first set of experiments is dedicated to determine the optimal allocation R_{opt} by conducting an exhaustive search over all possible allocations for a given swarm size and for each environment. More specifically, we run experiments for different numbers of storers N_s and of swarm sizes N in each environment, and evaluate each resulting configuration ($N \in [4,30]$ and $N_s \in [1, N-1]$, both by steps of 1). The allocation ratio $R = N_s/N$ remains fixed during a single experimental run; that is, the robots of the swarm cannot switch between subtasks.

Figure 8 reports the results for all tested configurations. We can observe that the best allocation ratio R for each of the tested swarm sizes N is approximately the same as the arena ratio. Nevertheless, we can observe exceptions to this behavior: for a large number of robots N, the swarm tends to assume an allocation ratio that is slightly higher than the arena ratio. For convenience, we use a swarm size of $N = 18$ in the following experimental sets as this swarm size allows the robots to allocate in a ratio that matches the arena ratio of the two environments studied.

The experiments allowed us to determine, for each environment, the optimal allocation ratio R_{opt} and the maximal performance P_{max_N} for all tested N. Table 1 reports the numeric values for R_{opt} and $P_{max_{18}}$, which are used in conjunction with the metrics presented in Sect. 4.2 to evaluate the proposed method.

5.2.2 Task switching cost

In the second set of experiments, we study the influence of the task switching cost c_s on the performance of the swarm. In this study, the task switching cost c_s is the time taken by robots to switch from one subtask to the other. We run experiments with different values for c_s, taken from the interval [0 s, 50 s] by steps of 2 s. We use two different swarms: one, in

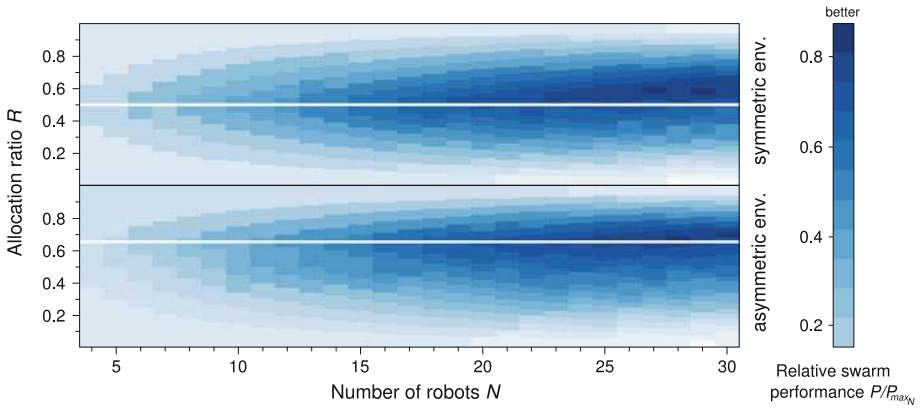

Fig. 8 The relative swarm performance P/P_{max_N} of the swarm for different allocation ratios R and number of robots N. Robots were not allowed to switch, that is, the allocation remained fixed during the course of an experiment. *Top* results for the symmetric environment, *bottom* results for the asymmetric environment. The *white horizontal line* indicates the arena ratio for the given environment (0.5 and 0.66, respectively). The median of P/P_{max_N} over 100 experimental runs is reported. *Darker* segments indicate better performance

Table 1 The optimal allocation ratios R_{opt} and the maximal swarm performance $P_{max_{18}}$ as determined empirically for each environment

Environment	R_{opt}	$P_{max_{18}}$
Symmetric	0.43/0.50/0.50/0.56/0.60	234.6/383.5/397.0/404.0/415.0
Asymmetric	0.52/0.62/0.67/0.67/0.72	191.7/366.0/386.0/394.5/414.0

Values are given as 1/25/50/75/99 % quantiles of the distribution

which there is a single group of robots that performs the two subtasks, and another one, in which two separate groups of robots work on the two subtasks and robots use the proposed method to decide when to switch between subtasks. In the first swarm, robots entering the task interface experience a task switching cost. In the second swarm, two types of costs can be experienced at the task interface: task switching costs or *transfer costs*. Transfer costs are induced by the transfer of objects and are the sum of the interface delay and the time needed to physically transfer the object from one robot to the other. A trade-off exists between task switching costs and transfer costs which determines whether or not it is advantageous to use separate groups of robots for the two subtasks. As shown in Fig. 9, the performance of a swarm in which a single group of robots performs the two subtasks linearly decreases with an increasing task switching cost, while the performance of a swarm composed of two groups of robots that use the proposed method stays approximately constant. This shows that the proposed method is advantageous in situations in which task switching costs are higher than a given threshold. Additionally, the method yields a consistent performance over different task switching costs. A swarm with a single group of robots outperforms a swarm composed of two groups that use the proposed method only in cases in which the task switching cost is lower than the transfer costs, which is the case at $c_s \approx 22\,\text{s}$. In the following simulation experiments, we use a task switching cost of $c_s = 30\,\text{s}$.

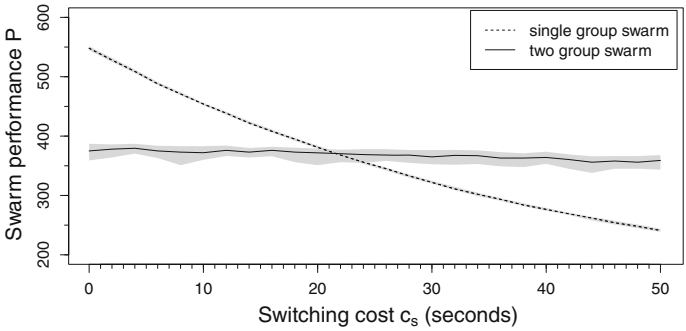

Fig. 9 The effect of the switching cost c_s on the swarm performance P in the asymmetric environment, measured as the number of objects collected at the end of the experiment. We use two different swarms: in the first a single group of robots performs both subtasks of the task sequence $\mathcal{T}$; in the second the robots work in two separate groups and use the proposed method to decide when to switch between subtasks. The results are collected over 100 experimental runs and are given as observation median (*black line*) and the 25–75 % quantiles (*gray area*)

5.2.3 Parameter study

In the third set of experiments, we study the influence of the parameters m and k (see Eq. 3) on the performance of a swarm that uses the proposed method. We explore the parameter space as follows: the shift parameter m is taken from the interval [0,50] by steps of 5 while the steepness parameter k is taken from the set $\{0.01, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$. For each condition, we perform 100 experimental runs. Figure 10 shows the results of the experiments for the symmetric environment (left column) and the asymmetric environment (right column). We evaluate the system using the metrics defined in Sect. 4.2: Fig. 10a reports the mean absolute error *MAE*, Fig. 10b the total number of task switches S_{tot}, and Fig. 10c the relative swarm performance $P/P_{max_{18}}$. In the plots, a darker square indicates a better value of the reported metric. The plots show that the system is mainly influenced by the shift parameter m, while the steepness parameter k has a lower impact. Figure 10a shows that the mean absolute error is low around a value of $m \in [5, 15]$, as well as for small values of k. Similarly, the number of task switches, reported in Fig. 10b, is constantly low for all larger values of m. The relative swarm performance $P/P_{max_{18}}$, reported in Fig. 10c, behaves similar to the mean absolute error, with lower values of m giving better results. In case of Fig. 10a, c, we can identify an interval of the parameter $m \in [5, 15]$ at which the method exhibits consistently good results with respect to the metric shown. In case of Fig. 10b, this can be observed for values of $m \geq 5$. These observations apply to both environments, indicating that the behavior of the method is little dependent on the environment.

In the following, we focus on the effects that the shift parameter m has on the behavior of the method. To this end, we set the steepness parameter k to 1. Figure 11 shows the influence of m on the behavior of the method in the asymmetric environment for $m \in \{0, 8, 50\}$ (Fig. 11a, b, c, respectively).

The plots in the left column of Fig. 11 report the relative swarm performance $P/P_{max_{18}}$. Setting $m = 0$ or $m = 50$ results in a poor relative swarm performance $P/P_{max_{18}}$ (0.79/0.81/0.82 and 0.60/0.68/0.76, respectively). On the other hand, setting $m = 8$ results in a near-optimal performance (0.91/0.95/0.97).

The plots in the middle column of Fig. 11 report the allocation ratio R of the swarm. Additionally, numerical results for the mean absolute error *MAE* are reported in the plot. Recall

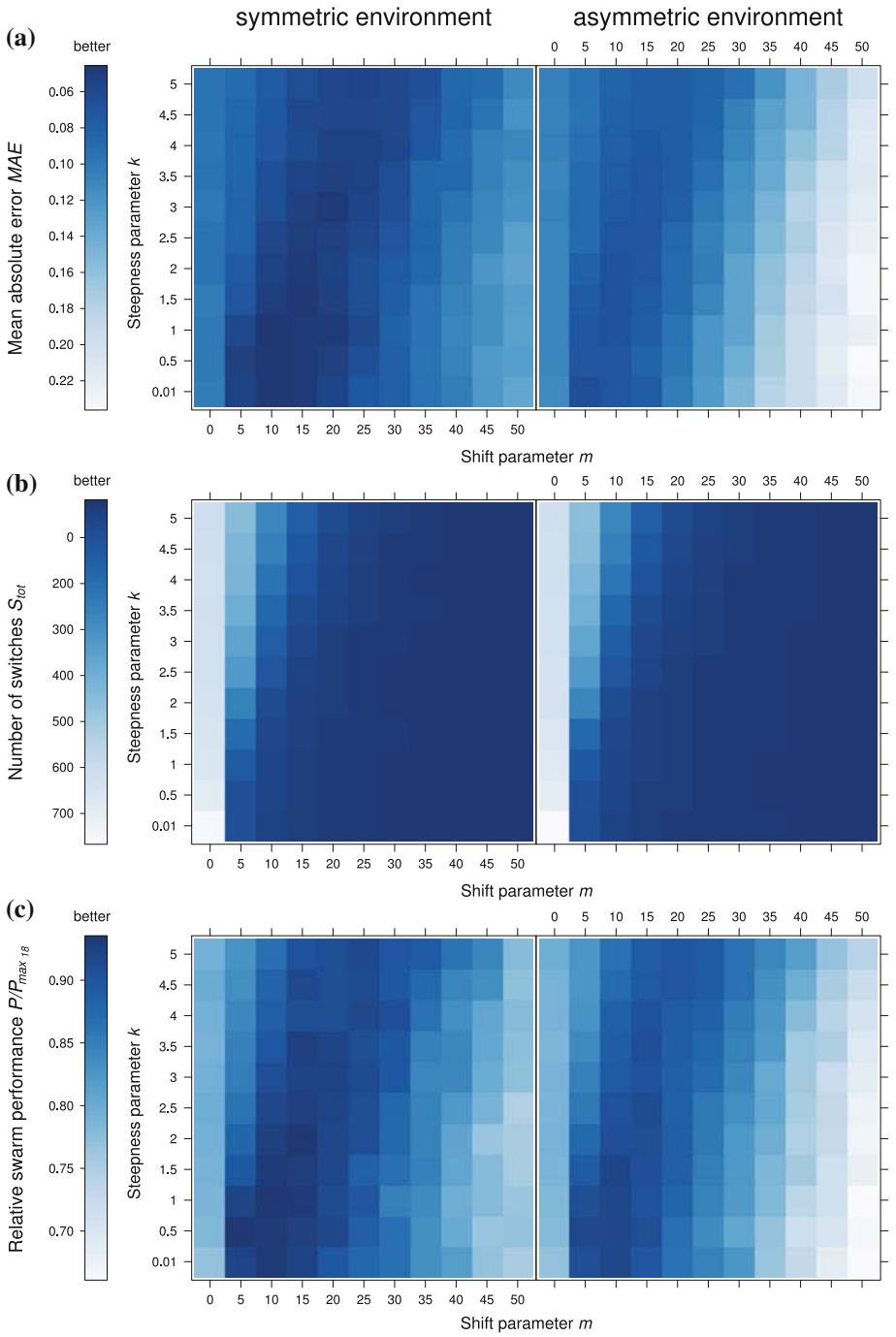

Fig. 10 The effect of the parameters of the method as evaluated by the metrics **a** mean absolute error *MAE*, **b** total number of switches between subtasks S_{tot}, and **c** relative swarm performance $P/P_{max_{18}}$. *Left column* results for the symmetric environment. *Right column* results for the asymmetric environment. The median of the respective metric over 100 experimental runs is reported. *Darker* segments indicate better values

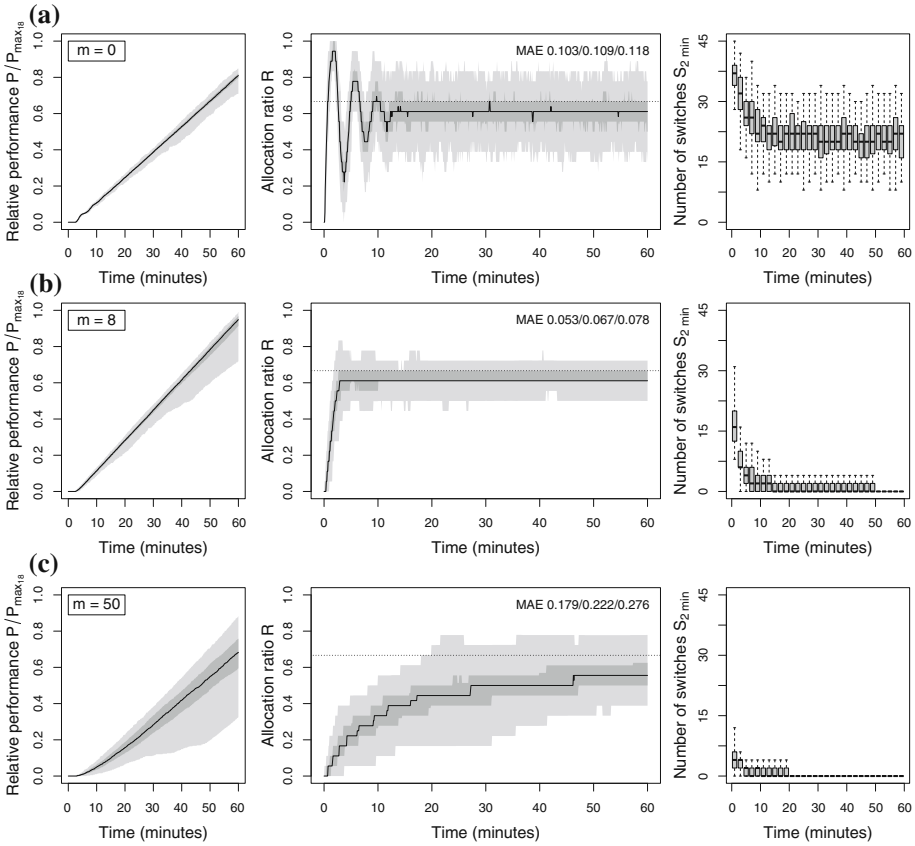

Fig. 11 Influence of the shift parameter $m = \{0, 8, 50\}$ and $k = 1$ shown in the rows **a**, **b**, and **c**, respectively. Results obtained in the asymmetric environment and evaluated using the following three metrics. *Left column* relative swarm performance $P/P_{max_{18}}$, given as observation median and the 1–99 % quantiles (*gray areas*). *Middle column* allocation ratio R, given as observation median (*black line*) and the 1–99 % quantiles (*gray areas*). The *horizontal dotted line* at $R_{opt} = 0.67$ indicates the optimal allocation for this environment. *Right column* box plot that reports the number of switches between subtasks S_{2min} for a time-window of 2 min (the *black horizontal line* indicates the median, the *box* indicates the interquartile range (IQR) and the *whiskers* indicate the lowest and highest datum within 1.5 IQR). The results are collected over 100 experimental runs

that the *MAE* measures the quality of the allocation: a low value indicates that the swarm reaches an allocation that is close to the optimal allocation R_{opt}. This is the case when setting $m = 0$ or $m = 8$, which results in an *MAE* of 0.103/0.109/0.118 and 0.053/0.067/0.078, respectively. On the other hand, setting $m = 50$ leads to a higher *MAE* (0.179/0.222/0.276).

The plots in the right column of Fig. 11 report the number of task switches S_{2min} observed in a time-window of 2 min. Figure 11a shows that setting $m = 0$ leads to strong oscillations in the allocation, as the robots tend to switch a lot between subtasks. On the other hand, setting $m = 0$ or $m = 50$ results in few task switches.

We can conclude that, while setting $m = 0$ results in a good allocation (i.e., low *MAE*), the performance of the method is degraded by the high cost of task switching. On the other hand, setting $m = 50$ results in low task switching costs, but the swarm does not attain a good allocation. Both cases yield a sub-optimal performance. We can conclude that only the combination of the metrics *MAE* and S_{2min} allows one to identify methods with the

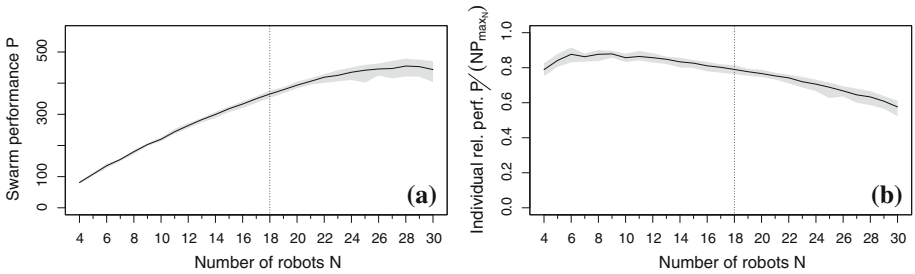

Fig. 12 Performance of the proposed method for different swarm sizes N in the asymmetric environment. **a** Swarm performance P, measured as the number of objects collected at the end of the experiment. **b** Individual relative performance $P/(N\,P_{max_N})$. The results are collected over 100 experimental runs and are given as observation median (*black line*) and the 25–75 % quantiles (*gray areas*). The *vertical gray line* at $N = 18$ indicates the swarm size used in the other experimental sets

desired behavior as shown in Fig. 11b: the performance of the swarm is near-optimal, as robots allocate with a ratio close to the optimal allocation R_{opt} while exhibiting few switches between subtasks.

On the basis of the results of this experimental set, we set the steepness parameter k to 1 and the shift parameter m to 8 in the following experiments.

5.2.4 Scalability

In the fourth set of experiments, we study the scalability of the proposed method. To this end, we run experiments for different swarm sizes, with N taken from the interval [4,30] by steps of 1. For each value of N we run 100 experiments. Figure 12 shows the results of the scalability experiments in terms of swarm performance P and individual relative performance $P/(N\,P_{max_N})$, defined as the average performance of a single individual relative to the maximal performance of a swarm of N robots. Figure 12a shows that the swarm performance increases sub-linearly for large swarm sizes and starts to decrease at the higher end of the interval studied. This is due to physical interference among robots: as we keep the size of the environment constant, the density of robots increases and therefore interference increases, causing the degradation of the swarm performance P. The individual relative performance, reported in Fig. 12b, confirms this phenomenon as its value decreases when the swarm size increases. Additionally, small swarms exhibit a low individual performance as they have difficulties to attain the optimal allocation due to the small number of robots. These results are consistent with what was observed in Fig. 8. For an in-depth study of the effect of physical interference in a similar system refer to Pini et al. [33].

5.2.5 Adaptivity

In the last set of simulation experiments, we study the adaptivity of a swarm using the proposed method to changing environmental settings. More specifically, we modify the position of the task interface at time $t = 30$ min so that the symmetric environment is transformed into the asymmetric environment. The results of these experiments are reported in Fig. 13. The swarm quickly adapts to the change in the environment by reaching an allocation that matches the new arena ratio of the environment. The relative swarm performance $P/P_{max_{18}}$ is near-optimal (0.90/0.93/0.95). The results confirm that the proposed method adapts well to changing environmental conditions.

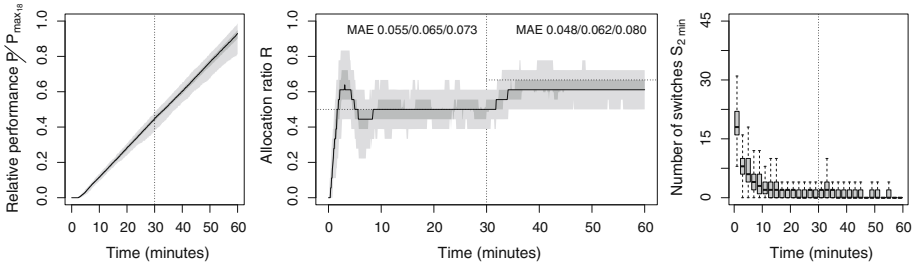

Fig. 13 Adaptivity of the allocation of the swarm when the environment changes from the symmetric environment to the asymmetric environment at $t = 30$ min (*vertical dotted lines*). *Left* relative swarm performance $P/P_{max_{18}}$, given as observation median and the 1–99 % quantiles (*gray areas*). *Middle* allocation ratio R, given as observation median (*black line*) and the 1–99 % quantiles (*gray areas*). The *horizontal dotted line* indicates the optimal allocation for each environment (at $R_{opt} = 0.5$ and 0.67, respectively). *Left* number of switches between subtasks S_{2min} for a time-window of 2 min (*box plots* as in Fig. 11). The results are collected over 100 experimental runs

6 Real robot experiment

In this section, we present the experiment that we conducted using real robots. The goal of this experiment is to validate the method proposed in this article in a realistic setting. The results presented in this section serve as a proof of concept, showing that the method for task allocation can be directly transferred to real robots. Therefore, we executed only a single experimental run.

We use some of the available robots to act as objects. Given the size of the environment and the number of s-bots available, we use a total of 8 robots, half of which are used to act as objects. For simplicity, in the rest of the section, we will refer to the robots acting as objects simply as objects.

The dimensions of the environment are 2.95 m by 1.25 m. The source, the nest and the task interface are each 0.4 m wide. The ground color is white, except in the source, the nest and the task interface, where the ground has been covered with a reflective material to ease detection. The store side of the task interface is marked with a green stripe that is 0.1 m wide. This helps the robots to sense in which area of the arena they are upon leaving the task interface. The environment is delimited by black tape 0.05 m wide.

At the beginning of the experiment, 4 s-bots are randomly positioned in the harvest area. Additionally, four objects are positioned in the source, close to the border of the environment. Due to the low wheel-speed of the robots, the total duration of an experiment exceeds the lifetime of the batteries of the robots by far. We were therefore forced to divide the experiment in multiple segments. We divided the experiment in 18 segments of 5 min each,[7] for a total duration of $t_{tot} = 90$ min. At the end of each segment, the experiment is stopped, the state of each robot is saved, and the robots are recharged. At the beginning of the following segment, the robots are placed at the same position they had at the end of the previous segment and the internal state of each robot is restored. The values of the parameters used for the proposed method are the same as determined in the simulation experiments, $m = 8$ and $k = 1$.

[7] The short duration of the segments is due to two reasons. First, we used robots equipped with old and non-replaceable batteries. Second, robots depleted their batteries in an inhomogeneous way—robots that transported many objects depleted their battery at a much faster pace—and we had to stop the experiment as soon as the first robot ran out of energy.

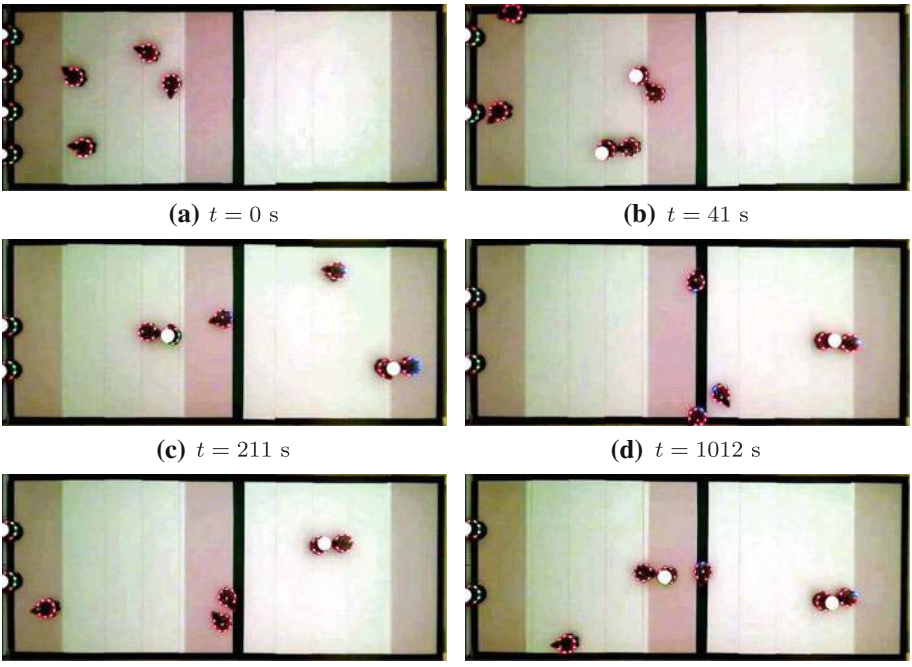

Fig. 14 Snapshots of the video recording of the real robot experiment that show the evolution of the experiment over time *t*. Objects are marked by a *white circle* on *top*. **a** All robots start in the harvest area of the arena, that is, they are working on the harvest subtask; **b** the robots start transporting objects to the task interface; **c** three robots switched to the store subtask: one is delivering an object to the nest, one is returning to the task interface after storing an object in the nest, and one is waiting in the task interface to receive an object; **d** the whole swarm switched to the store subtask; **e** robots start switching back to the harvest subtask in order to equalize the performance across the two subtasks; **f** two robots work on each subtask, the swarm reached the optimal allocation

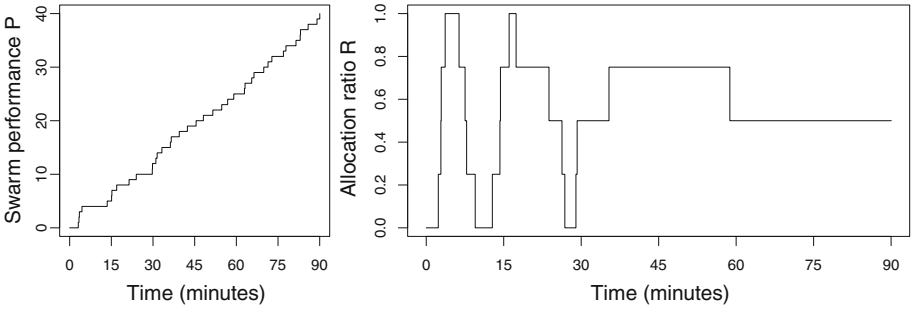

Fig. 15 Results of the real robot experiment. *Left* swarm performance *P*, measured as the total number of objects collected. *Right* allocation ratio *R*. The results are obtained in a single proof-of-concept experimental run

The experiment has been recorded by an overhead camera. The resulting 18 segments have been joined into a single video that can be accessed in the supplementary on-line material [7]. Figure 14 shows the evolution of the experiment over time using snapshots taken from the video.

Figure 15 shows the result of the experiment using the same metrics we employed for evaluating the results of the simulation experiments (see Sect. 4.2). Figure 15 shows that, after an initial period of time in which switches between subtasks are frequent, the allocation of the swarm stabilizes with two robots working as harvesters and two robots working as storers. During the course of the 90 min of the experimental run, the robots harvested a total of 40 objects.

7 Conclusions and future work

The self-organized method proposed in this article allows a swarm of robots to allocate to subtasks that are sequentially interdependent. Being often a result of strategies that partition complex tasks into smaller ones, such subtasks are common in natural and artificial systems. We evaluated the method using a swarm of s-bot robots in simulation experiments as well as in a proof-of-concept experiment with real robots.

From the results we can conclude that a swarm using the proposed method is able to allocate its individuals to two sequentially interdependent subtasks in a near-optimal way. The method relies only on the robots' individual perception of the delays experienced when waiting for robots working on the other subtask. The method does not require communication capabilities and can therefore be employed in swarms of simple robots. Additionally, the ability of the robots to allocate in a near-optimal way does not depend on the properties of the problem at hand. More precisely, each robot's decision to switch between subtasks does not depend on an absolute measure of a property of the problem such as the duration of the subtasks, but only on the interface delays experienced by the robots in one subtask relative to the interface delays experienced in the other subtask. As a result, the method can be used in different contexts without the need of parameter changes. We have shown that the method is adaptive as the robots successfully re-allocate in case the environmental settings change.

Although we limited the study presented to a problem instance with two subtasks, we believe that the method can directly be transferred to problems with more than two subtasks. Preliminary studies on an agent-based model indicate that the method can indeed be applied to such problems. We plan to conduct further experiments for an instance of such a problem, using a new generation of robots.

A further direction of work concerns the way subtasks interface with each other. The method presented in this article has been tested exclusively on a foraging task where the task interface can be represented by a physical area that requires direct transfer of objects. A possible extension of the work presented in this article concerns other ways of performing object transfer, such as indirect transfer of objects by using a cache site.

Acknowledgments The research leading to the results presented in this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 246939. Marco Dorigo, Mauro Birattari, and Arne Brutschy acknowledge support from the Belgian F.R.S.–FNRS. Giovanni Pini acknowledges support from Université Libre de Bruxelles through the "Fonds David & Alice Van Buuren".

References

1. Agassounon, W., & Martinoli, A. (2002). Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. *Proceedings of the first international joint conference on autonomous agents and multi-agent systems (AAMAS-02)* (pp. 1090–1097). New York: ACM Press.

2. Anderson, C., & Ratnieks, F. L. W. (1999a). Task partitioning in insect societies. I: Effect of colony size on queueing delay and colony ergonomic efficiency. *The American Naturalist*, *154*(5), 521–535.
3. Anderson, C., & Ratnieks, F. L. W. (1999b). Task partitioning in insect societies. II: Use of queueing delay information in recruitment. *The American Naturalist*, *154*(5), 536–548.
4. Anderson, C., & Ratnieks, F. L. W. (2000). Task partitioning in insect societies: Novel situations. *Insectes Sociaux*, *47*(2), 198–199.
5. Berman, S., Halasz, A., Hsieh, M. A., & Kumar, V. (2009). Optimized stochastic policies for task allocation in swarms of robots. *IEEE Transactions on Robotics*, *25*, 927–937.
6. Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems*. New York: Oxford University Press.
7. Brutschy, A., Pini, G., Pinciroli, C., Birattari, M., & Dorigo, M. (2011). Self-organized task allocation to sequentially interdependent tasks in swarm robotics—Online supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2011-002/.
8. Campo, A., & Dorigo, M. (2007). Efficient multi-foraging in swarm robotics. In M. Capcarrere, A. A. Freitas, P. J. Bentley, C. G. Johnson, & J. Timmis (Eds.), *Advances in artificial life: Proceedings of the VIIIth European conference on artificial life (ECAL 2005)* (Vol. 4648, pp. 696–705). Berlin: Springer.
9. Christensen, A. L., O'Grady, R., & Dorigo, M. (2007). Morphology control in a multirobot system. *IEEE Robotics and Automation Magazine*, *11*(6), 732–742.
10. Cicirello, V. A., & Smith, S. F. (2004). Wasp-like agents for distributed factory coordination. *Autonomous Agents and Multi-Agent Systems*, *8*(3), 237–266.
11. Dahl, T. S., Matarić, M. J., & Sukhat, G. S. (2009). Multi-robot task allocation through vacancy chain scheduling. *Robotics and Autonomous Systems*, *57*, 674–687.
12. Dasgupta, P. (2011). Multi-robot task allocation for performing cooperative foraging tasks in an initially unknown environment. In L. C. Jain, E. V. Aidman, & C. Abeynayake (Eds.), *Innovations in defence support systems 2*. Studies in computational intelligence (Vol. 338, pp. 5–20). Berlin: Springer.
13. Dias, M. B., Zlot, R., Kalra, N., & Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, *94*, 1257–1270.
14. Dorigo, M. (2005). SWARM-BOT: An experiment in swarm robotics. In P. Arabshahi & A. Martinoli (Eds.), *2005 IEEE swarm intelligence symposium (SIS-05)* (pp. 192–200). Piscataway, NJ: IEEE Press.
15. Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., et al. (2013). Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine* (in press).
16. Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella, T. H., Baldassarre, G., et al. (2004). Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, *17*(2–3), 223–245.
17. Ferreira, P. R., Boffo, F. S., & Bazzan, A. L. C. (2008). Using Swarm-GAP for distributed task allocation in complex scenarios. In N. Jamali, P. Scerri, & T. Sugawara (Eds.), *Massively multi-agent technology*. LNCS (Vol. 5043, pp. 107–121). Berlin: Springer.
18. Fowler, H. H., & Robinson, S. W. (1979). Foraging by *Atta sexdens* (Formicidae: Attini): Seasonal patterns, caste and efficiency. *Ecological Entomology*, *4*(3), 239–247.
19. Gerkey, B. P., & Matarić, M. J. (2003). Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proceedings of the IEEE international conference on robotics and automation (ICRA 2003)* (pp. 3862–3867). Pitscataway, NJ: IEEE Press.
20. Gerkey, B. P., & Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, *23*(9), 939–954.
21. Goldberg, D., Cicirello, V., Dias, M. B., Simmons, R., Smith, S., & Stentz, A. (2003). Task allocation using a distributed market-based planning mechanism. In Proceedings of the second international joint conference on autonomous agents and multiagent systems (pp. 996–997). New York, NY: ACM Press.
22. Groß, R., Bonani, M., Mondada, F., & Dorigo, M. (2006). Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, *22*(6), 1115–1130.
23. Ikemoto, Y., Miura, T., & Asama, H. (2010). Adaptive division-of-labor control algorithm for multi-robot systems. *Journal of Robotics and Mechatronics*, *22*(4), 514–525.
24. Kalra, N., & Martinoli, A. (2006). A comparative study of market-based and threshold-based task allocation. In *Distributed autonomous robotic systems 7* (pp. 91–102). Berlin: Springer.
25. Krieger, M. J. B., & Billeter, J.-B. (2000). The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Journal of Robotics and Autonomous Systems*, *30*, 65–84.
26. Labella, T. H., Dorigo, M., & Deneubourg, J.-L. (2006). Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems*, *1*(1), 4–25.
27. Liu, W., Winfield, A., Sa, J., Chen, J., & Dou, L. (2007). Towards energy optimization: Emergent task allocation in a swarm of foraging robots. *Adaptive Behavior*, *15*(3), 289–305.

28. Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I. V., Floreano, D., Deneubourg, J.-L., et al. (2004). SWARM-BOT: A new distributed robotic concept. *Autonomous Robots*, *17*(2–3), 193–221.
29. Nouyan, S., Campo, A., & Dorigo, M. (2008). Path formation in a robot swarm. Self-organized strategies to find your way home. *Swarm Intelligence*, *2*(1), 1–23.
30. Nouyan, S., Groß, R., Bonani, M., Mondada, F., & Dorigo, M. (2009). Teamwork in self-organized robot colonies. *IEEE Transactions on Evolutionary Computation*, *13*(4), 695–711.
31. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., et al. (2011a). ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS 2011)* (pp. 5027–5034). Los Alamitos, CA: IEEE Computer Society Press.
32. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., et al. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence, 6*(4), 271–295.
33. Pini, G., Brutschy, A., Birattari, M., & Dorigo, M. (2011a). Task partitioning in swarms of robots: Reducing performance losses due to interference at shared resources. In J.-L. Ferrier & J. Filipe (Eds.), *Informatics in control, automation and robotics: Selected papers from the international conference on informatics in control, automation and robotics 2009*. LNEE (Vol. 85). Berlin: Springer.
34. Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., & Birattari, M. (2011b). Task partitioning in swarms of robots: An adaptive method for strategy selection. *Swarm Intelligence*, *5*(3–4), 283–304.
35. Ratnieks, F. L. W., & Anderson, C. (1999). Task partitioning in insect societies. *Insectes Sociaux*, *46*(2), 95–108.
36. Scheidler, A., Merkle, D., & Middendorf, M. (2008). Stability and performance of ant queue inspired task partitioning methods. *Theory in Biosciences*, *127*(2), 149–161.
37. Theraulaz, G., Bonabeau, E., & Deneubourg, J.-L. (1998). Response threshold reinforcement and division of labour in insect societies. *Proceedings: Biological Sciences*, *265*(1393), 327–332.