Swarm Intell (2011) 5:283–304
DOI 10.1007/s11721-011-0060-1

Task partitioning in swarms of robots: an adaptive method for strategy selection

Giovanni Pini · Arne Brutschy · Marco Frison · Andrea Roli · Marco Dorigo · Mauro Birattari

Received: 31 October 2010 / Accepted: 22 August 2011 / Published online: 5 October 2011

Abstract Task partitioning is the decomposition of a task into two or more sub-tasks that can be tackled separately. Task partitioning can be observed in many species of social insects, as it is often an advantageous way of organizing the work of a group of individuals. Potential advantages of task partitioning are, among others: reduction of interference between workers, exploitation of individuals' skills and specializations, energy efficiency, and higher parallelism. Even though swarms of robots can benefit from task partitioning in the same way as social insects do, only few works in swarm robotics are dedicated to this subject. In this paper, we study the case in which a swarm of robots has to tackle a task that can be partitioned into a sequence of two sub-tasks. We propose a method that allows the individual robots in the swarm to decide whether to partition the given task or not. The method is self-organized, relies on the experience of each individual, and does not require explicit communication between robots. We evaluate the method in simulation experiments,

Electronic supplementary material The online version of this article (doi:10.1007/s11721-011-0060-1) contains supplementary material, which is available to authorized users.

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

A. Brutschy
e-mail: arne.brutschy@ulb.ac.be

M. Dorigo
e-mail: mdorigo@ulb.ac.be

M. Birattari
e-mail: mbiro@ulb.ac.be

M. Frison · A. Roli
DEIS-Cesena, Alma Mater Studiorum, Università di Bologna, Cesena, Italy

M. Frison
e-mail: mfrison85@gmail.com

A. Roli
e-mail: andrea.roli@unibo.it

using foraging as testbed. We study cases in which task partitioning is preferable and cases in which it is not. We show that the proposed method leads to good performance of the swarm in both cases, by employing task partitioning only when it is advantageous. We also show that the swarm is able to react to changes in the environmental conditions by adapting the behavior on-line. Scalability experiments show that the proposed method performs well across all the tested group sizes.

Keywords Task partitioning · Foraging · Swarm robotics · Self-organization

1 Introduction

Task partitioning is a way of organizing work that consists in dividing a task into two or more sub-tasks (Jeanne 1986; Ratnieks and Anderson 1999). The sub-tasks can then be tackled separately by different individuals at the same time, or by the same individual at different times.

In nature, task partitioning can be observed in many species of social animals. The most evident example are humans, that exploit the advantages of task partitioning both at the individual and at the societal level. At the level of the individual, a natural way of tackling a non-trivial and lengthy task is to decompose it into less complex sub-tasks. At the level of the society, many activities and interactions are so complex that in order to be manageable, they need to be simplified by partitioning them. Partitioning has been studied and formalized in many different disciplines. In politics and sociology it is coded in the *divide et impera* principle, born in ancient roman times and applied throughout the centuries. In computer science, this principle consists of recursively breaking down problems into sub-problems, till the sub-problems become solvable, and then re-combining their solutions (Aho 1983).

As we move to simpler forms of individuals, such as social insects, problems and needs also become simpler. Nonetheless, also in insect societies task partitioning can be an advantageous way of organizing work. In fact, task partitioning can be observed in several species of ants and bees, for example in material transportation, nest construction and waste removal (Ratnieks and Anderson 1999). Benefits of task partitioning in all the mentioned activities are, among others, reduction of interference between workers, better exploitation of individuals' skills and specializations, increased energy efficiency, and higher parallelism. On the other hand, it has to be noticed that task partitioning also has associated costs, typically due to coordination efforts, delays and overheads where two sub-tasks interface with each other. Nonetheless, the fact that many examples of task partitioning can be observed in nature suggests that the gain of using partitioning often overcomes its costs.

In this paper, we study task partitioning in the context of swarm robotics. Swarm robotics deals with the study and the implementation of distributed robotic systems composed of a large number of autonomous robots. Such systems are implemented drawing inspiration from social insects. Therefore, they have many similarities with swarms of insects. As in social insects, the individuals are simple with respect to the task they have to solve, and the control of the swarm is decentralized. Complex behaviors result from individuals' decisions based on local perceptions, local interactions, and local communication. Parallels between swarm robotics and the world of social insects are not limited to the characteristics of the individuals, but they can also be drawn at the level of the problems to be solved. In fact, many of the problems faced by social insects have also a relevance for swarm robotics. Examples are: foraging, collective transport of heavy objects, self-assembly, exploration, and collective decision making (see Beni 2005 and Şahin 2005 for a review). The fact that there are cases

in which insects obtain benefits by employing task partitioning, makes appealing the study of task partitioning in swarms of robots facing similar situations.

In this paper, we propose a method that allows a swarm of robots to decide which strategy to use for tackling a given task: a strategy that makes use of task partitioning, and decomposes a task into a sequence of sub-tasks, or one that does not, and tackles the given task as a single, unpartitioned task. The method is fully distributed, based on the individuals' local perception and requires no explicit communication between the robots. We test the method using a swarm of robots in simulation-based experiments. The testbed for these experiments is foraging, where the task of retrieving each object is pre-partitioned into a sequence of two sub-tasks. We test different experimental conditions that vary in terms of the cost of employing task partitioning. The method proposed in this paper allows each robot in the swarm to make a choice depending on this cost. The choice is either to perform the object retrieval task as single, unpartitioned task, or to employ task partitioning and perform one of the two sub-tasks. We show that the proposed method gives good performance in all the cases, with the robots selecting an advantageous strategy. The method is able to react to changes in the environment by adapting the strategy to new conditions. Additionally, the method scales well with the number of robots, providing good performance across different swarm sizes.

The rest of the paper is organized as follows. Section 2 provides a review of related works. Section 3 defines the task-partitioning problem studied in this work. Section 4 presents the method we propose for tackling this problem. Section 5 presents the experimental framework we use for evaluating the proposed method. Section 6 presents and discusses the experiments and the results we obtained. Section 7 concludes the paper by summarizing the contribution of the research and describing directions for future work.

2 Related work

Task partitioning and task allocation are ways of organizing work in groups of individuals. Gordon (1999) defines task allocation as "*the process that adjusts the numbers of workers engaged in each task*". Ratnieks and Anderson (1999) describe task partitioning as "*the phenomenon in which a piece of work is divided among two or more workers*". The difference between task partitioning and task allocation resides in the fact that, while task allocation acts at the level of the workforce, task partitioning acts on the tasks themselves by decomposing them into smaller sub-tasks. The two topics are strongly intertwined as task allocation is often applied to previously partitioned tasks.

Task allocation has been intensively studied both in biology and in robotics. Bonabeau et al. (1996) describe the response threshold model and show that it can explain the division of labor observed in *Pheidole* ants. Bonabeau et al. (1999) introduce a simple algorithm that is based on the response threshold model and can be used to obtain flexible task allocation in artificial systems.

Threshold-based task allocation methods have been studied, among others, in the context of robotic foraging (Krieger and Billeter 2000; Labella et al. 2006) and objects clustering (Agassounon and Martinoli 2002). Kalra and Martinoli (2006) compared threshold-based and market-based task allocation methods. The authors claim that, when the information is not accurate, threshold-based methods and market-based methods have similar performance. With accurate information market-based methods perform better. Threshold-based methods have the advantage of being less costly in terms of communication and computational resources.

While studies in task allocation assume the existence of two or more tasks and focus on the problem of assigning individuals to these tasks, this paper focuses on the problem of

self-organized task partitioning. In the following we give a detailed review of the literature on the topic. We start by reviewing the large body of research that has been carried out on the topic in the field of biology. Then we review the few works that have been devoted to the topic in the context of swarm robotics.

Task partitioning can be observed in many species of social insects, with variations in terms of the task being performed, the number of sub-tasks, and the modality by which it is realized (Ratnieks and Anderson 1999). Theraulaz et al. (2002) study the hunting strategy of the ponerine ant *Ectatomma ruidum*. The ant partitions the hunting task into stinging and transporting. Some individuals kill prey animals that are then transported to the nest by others. The authors mention the fact that individuals' differences and learning could be the reasons why task partitioning is employed: it allows a better exploitation of the specialization of the individuals. The work of Hart and Ratnieks (2001b) describes how task partitioning is employed by the leaf cutting ant *Atta cephalotes* when performing garbage disposal and management. This ant grows fungi in gardens located in the nest. Garbage has to be removed from these gardens and stored in garbage heaps. The garbage removal task is partitioned into two sub-tasks: first the garbage is removed by the ants working in the garden and then stored in the garbage heaps by other workers. Partitioning the garbage removal task allows the ants to isolate heap workers from the rest of the colony, as they are possibly contaminated by hazardous pathogens. The one of *Atta cephalotes* is an example of another benefit of task partitioning: it can be used to obtain physical separation of groups of workers. Lopes et al. (2003) and Hubbell et al. (1980) point out that partitioning a foraging task allows workers to return quickly to a food source, reinforcing the pheromone trail that leads to it. In this case partitioning is beneficial as it increases the performance of the swarm.

When the task being partitioned involves the transportation of materials, the material needs to be transferred between individuals working on the different sub-tasks. There are two possible ways to make this happen: either through *direct* transfer between workers, or through *indirect* transfer (Hart et al. 2002; Ratnieks and Anderson 1999). Anderson et al. (2002) review partitioned foraging through bucket brigades: material is transferred directly from one individual to another. Transfers usually happen in locations that are not determined a priori, but depend on the occasional encounters between workers. A benefit of direct transfer is that it allows the adaptation of the weight of the load to the strength of the transporter. The result is an increase of the throughput of objects delivered to the nest (Reyes and Fernández Haeger 1999; Anderson and Jadin 2001). Most of the examples cited by Anderson et al. (2002) concern ants, but direct material transfer has also been reported in the foraging activity of social wasps (Jeanne 2002) and bees (Seeley 1989; Anderson and Ratnieks 1999).

When material transfer is indirect, specific locations called *caches* serve as temporary storage, where materials can be dropped and picked up. Caches allow the individuals to exchange material in an asynchronous fashion. The use of caches has been reported in several species of social insects. Fowler and Robinson (1979) describe the strategy employed in leaf foraging by the *Atta* ants: some individuals work on the tree, cutting the leaves and then dropping them to the ground, which serves as cache. On the ground, leaves are cut in pieces and transported to the nest by other workers. Here the advantage of partitioning stems from the fact that the individuals do not need to repeatedly climb the tree. Partitioning the task therefore reduces the energy requirements at the level of the swarm. Hart and Ratnieks (2000, 2001a) point out that caches can be beneficial because they can reduce material losses due to imbalances between foraging and processing rates.

Despite the potential advantages, task partitioning received only marginal attention in swarm robotics research. In the work of Fontan and Matarić (1996), a foraging task is pre-partitioned into several sub-tasks; each robot works on one of the sub-tasks in an exclusive

area, assigned a priori. The study shows that the partitioning strategy increases task efficiency by reducing robots' competition for space. A similar result, with non-exclusive and dynamically assigned working areas was presented by Pini et al. (2011a). Shell and Matarić (2006) and Østergaard et al. (2001) compared homogeneous foraging and bucket brigade algorithms. They focused on the importance of spatial subdivision in reducing interference and improving performance. Lein and Vaughan (2008) extended the work of Shell and Matarić (2006) by introducing a mechanism that adapts the size of the working areas of the robots in response to the interference experienced by them. Parker and Zhang (2010) studied the case in which a group of robots has to perform a predefined sequence of two mutually exclusive sub-tasks: a sub-task should begin only when the preceding one is completed and no robot is working on it anymore. The focus of the study is on the decision making process that allows the robots to collectively estimate whether a sub-task is complete and the group can start working on the following one. Lately, Scheidler et al. (2008) studied the influence of partitioning in a task allocation system. More specifically, they studied the basic properties of task-partitioning methods, like stability and efficiency, in a self-organized service system.

To the best of our knowledge, the few works listed here are the only ones that have been devoted to the topic of task partitioning in swarm robotics. No work has been published so far in which the robots autonomously decide, on the basis of the characteristics of the environment and of the task, whether to partition a given task or not.

3 Definition of the task-partitioning problem

This work focuses on the case in which a given task can be partitioned into two sub-tasks. These sub-tasks are sequentially interdependent, meaning that they have to be executed in a specific order. This situation is depicted in the bottom part of Fig. 1: a given task can be partitioned into two sub-tasks φ_1 and φ_2 that have to be performed in sequence. The output of sub-task φ_1 serves as input to sub-task φ_2. We consider the case in which the sub-tasks

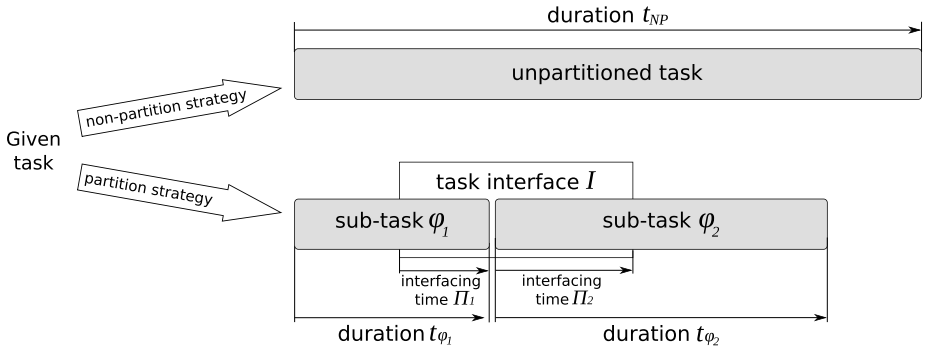

Fig. 1 Representation of the sequential task-partitioning problem. A given task can be completed either using a non-partition or a partition strategy. When the non-partition strategy is employed, the given task is tackled as a single, unpartitioned task, that requires a time t_{NP} to be completed. When the partition strategy is employed, the given task is partitioned into a sequence of two sub-tasks φ_1 and φ_2, that require, respectively, a time t_{φ_1} and t_{φ_2} for their completion. The two sub-tasks are linked one to the other by means of an interface I, whose usage imposes an overhead in terms of interfacing times Π_1 and Π_2. In general, the time needed to perform the sub-task φ_1 differs from the one needed to perform φ_2. Typically, t_{NP} differs from the sum of t_{φ_1} and t_{φ_2}. This makes task partitioning advantageous or not; the figure represents the case in which partitioning is advantageous

interface with each other in an asynchronous way: the output of φ_1 can be stored at an interface I, of finite storage capacity, and has then to be processed in the sub-task φ_2.

As an example, let us consider the testbed employed in this work: foraging. Foraging is the activity that consists in multiple (and in general parallel) repetitions of the object retrieval task, that is harvesting an object from the environment and storing it in the nest. Consider the case in which the object retrieval task can be partitioned into an harvest and a store sub-tasks. The use of a cache, as described in Sect. 2, allows the transfer of objects from the harvest to the store sub-tasks. The sequential dependency resides in the fact that, in order to retrieve an object to the nest (thus completing once the object retrieval task), the harvest and the store sub-tasks have to be performed once in the correct order.

Returning to Fig. 1, the given task can be performed using two different strategies. One strategy is to tackle the given task as single, unpartitioned task. The other strategy is to employ task partitioning, and tackle each of the two sub-tasks φ_i separately. In the following, we will refer to these strategies as the *non-partition strategy* and the *partition strategy*, respectively. Each strategy entails a cost that can be measured in different ways, depending on the context; examples are: energy, time, and resources required to perform the unpartitioned task or the sub-tasks.

In this study we use time as the cost of a strategy. Under this definition, the cost of the non-partition strategy is the time t_{NP} needed to perform the unpartitioned task; the cost of the partition strategy is the sum of the times t_{φ_1} and t_{φ_2} needed to perform the two sub-tasks φ_1 and φ_2. The optimal strategy is the strategy that requires the least time to complete the given task. Notice that the time needed to perform the unpartitioned task and the sub-tasks not only depends on the nature of the tasks themselves, but it also depends on the strategy employed. For example, t_{NP} can increase due to conflicts in accessing shared resources, while t_{φ_1} and t_{φ_2} include possible overheads at the interface I (i.e., interfacing times Π_1 and Π_2).

In the example of foraging, the non-partition strategy could entail costs in terms of interference between individuals along the path, as well as conflicts in accessing the objects or in entering the nest. The partition strategy, by constraining the movement of individuals within areas of the environment, can reduce these costs. On the other hand, it entails costs in terms of overheads and delays when transferring objects between individuals. Which of the two strategies is better depends on the characteristics of the environment.

4 The proposed method

In this work we propose an adaptive method that allows a swarm of robots to tackle the problem depicted in Fig. 1. The method allows each robot to autonomously decide whether to partition a given task into the sequence of two sub-tasks or not. We study different cases in which the partition strategy can be more or less advantageous than the non-partition one. The method does not make use of explicit communication: each robot takes its own decision on the basis of the perceived costs of the two strategies.[1] If a robot chooses to employ task partitioning, it works only on one of the possible sub-tasks φ_i. If a robot chooses not to employ task partitioning, it performs the given task as a single, unpartitioned task. After completing a task, be it either the unpartitioned task or one of the two sub-tasks φ_i, a robot decides which strategy to employ next. Note that, while for a single robot the choice of the

[1] The fact that the method does not employ active communication increases its applicability in contexts where agents have none or limited communication capabilities. This is the case of this study, as we employ e-puck robots. For more details about the robots please refer to Sect. 5.1.

strategy is binary, at the level of the swarm there can be simultaneously some robots that adopt the partition strategy and other robots that adopt the non-partition strategy.

In the proposed method, each robot has a probability P_p of employing task partitioning. This probability is defined by the following sigmoid functions:

$$P_p = \begin{cases} [1 + e^{-S(\hat{t}_{\mathrm{NP}}/(\hat{t}_{\varphi_1}+\hat{t}_{\varphi_2})-1)}]^{-1}, & \text{if } \hat{t}_{\mathrm{NP}} > (\hat{t}_{\varphi_1} + \hat{t}_{\varphi_2}), \\ [1 + e^{-S(1-(\hat{t}_{\varphi_1}+\hat{t}_{\varphi_2})/\hat{t}_{\mathrm{NP}})}]^{-1}, & \text{if } \hat{t}_{\mathrm{NP}} \leq (\hat{t}_{\varphi_1} + \hat{t}_{\varphi_2}), \end{cases} \tag{1}$$

where S is a steepness factor, $\hat{t}_{\mathrm{NP}}$ is an estimate of the time t_{NP} required to complete the unpartitioned task. In other words, $\hat{t}_{\mathrm{NP}}$ is the estimated cost of the non-partition strategy. On the other hand, $\hat{t}_{\varphi_1}$ and $\hat{t}_{\varphi_2}$ are estimates of the times t_{φ_1} and t_{φ_2} required to perform each of the two sub-tasks. In other words, the sum of $\hat{t}_{\varphi_1}$ and $\hat{t}_{\varphi_2}$ is the estimated cost of the partition strategy. The robots are never inactive, meaning that if a robot decides not to employ task partitioning then it works on the unpartitioned task (with a probability $P_{\mathrm{np}} = 1 - P_p$). By using the functions given in (1), there is always a non-null probability of selecting the strategy perceived as the worst. Nonetheless, this probability decreases sharply with the difference between the costs of the two strategies. The exploration-exploitation balance can be regulated by varying the parameter S: the higher the value of S, the higher the amount of exploitation.

Each estimate is computed as a weighted average of the time it takes to perform the sub-tasks or the unpartitioned task. For the sub-task φ_i, the time estimate is updated as follows:

$$\hat{t}_{\varphi_i} \leftarrow (1 - \alpha)\, \hat{t}_{\varphi_i} + \alpha\, t_M, \tag{2}$$

where t_M is the robot's measure of the time that took the last execution of the sub-task φ_i. $\alpha \in (0, 1]$ is a weight factor that influences the responsiveness to changes: a high value of α leads to a faster responsive behavior. An analogous formula is used for the estimate $\hat{t}_{\mathrm{NP}}$.

The estimates $\hat{t}$ depend not only on the intrinsic characteristics of the environment and of the tasks, but also on the choices made by the other robots. Interference among robots impacts on the time needed to perform the sub-tasks and the unpartitioned task, and depends on how many robots are performing the same (sub-)task. In addition, in case the partition strategy is employed, the time required to complete a sub-task φ_i also depends on the performance of the robots working on the other sub-task. In fact, poor performance of one of the sub-tasks reduces the throughput at the interface; this impacts negatively on the waiting time experienced by the robots working on the other sub-task.

In the example of foraging previously mentioned in Sect. 3, each robot would keep an estimate of the time it takes to complete the object retrieval task when performed as a single, unpartitioned task, as well as when performed as sequence of harvest and store sub-tasks. This information would be used by the robot to decide whether to employ task partitioning: if the sum of the times needed to perform the two sub-tasks is less than the time needed to perform the unpartitioned task, then the probability of employing the partition strategy is higher than the one of employing the non-partition strategy.

As each robot has only estimates of the real costs, noise might lead to sub-optimal decisions. Furthermore, the optimal choice that a robot can make can change over time as result of the choices made by the other robots or changes in the environment. Therefore, the different strategies need to be sampled by the robots from time to time, in order for the system to be reactive.

When a partition strategy is employed, a robot working on φ_1 might be waiting for a free store spot at the interface; analogously a robot working on φ_2 might be waiting for input at

the interface. When a robot finds itself in one of the two waiting situations, it can decide to give up performing its sub-task with a probability defined as

$$Pg_{\varphi_i}(w_I) = \left[1 + e^{\Theta(w_I, \hat{t}_{\varphi_i})}\right]^{-1}, \tag{3}$$

where w_I is the current measured time the robot has been waiting at the interface I. $\Theta(w_I, \hat{t}_{\varphi_i})$ is computed as

$$\Theta(w_I, \hat{t}_{\varphi_i}) = K\left(\frac{w_I - \hat{t}_{\varphi_i}}{(\hat{t}_{\varphi_1} + \hat{t}_{\varphi_2})} + O\right), \tag{4}$$

where K and O are a steepness and an offset factor, respectively, and $\hat{t}_{\varphi_i}$, computed using (2), is the estimated time required for performing sub-task φ_i. When a robot gives up performing a sub-task φ_i, and its current waiting time w_I is greater than $\hat{t}_{\varphi_i}$, the value of $\hat{t}_{\varphi_i}$ is updated using (2) (w_I replaces t_M in the formula). The probability function defined in (3) ensures that a robot, while performing a sub-task, always has a non-nullprobability of giving up waiting at the interface.

Returning to the example of foraging, a robot working on the store sub-task might wait very long for an object to become available at the cache. A long waiting time means that the performance of the rest of the swarm on the harvest sub-task is poor. In this case, the robot should switch to perform that sub-task. Analogously, a robot working on the harvest sub-task should switch to the store sub-task when it has been waiting too long for an empty spot in the cache to deliver an object. The concept of giving up on a task when waiting too long is not a novelty proposed in the work here presented. It has been adopted many times before both in swarm robotics (Labella et al. 2006) and in collective robotics (Parker 1998).

5 Experimental framework

This section describes the experiments we employ to assess the validity of the method presented in Sect. 4. We test the method with a swarm of robots performing a single nest, single source homogeneous foraging activity (see Winfield 2009, for a taxonomy). This activity consists in the parallel repetition of the object retrieval task: harvesting an object from the source and storing it in the nest. Objects can be found at the source, which is at a known location and never depletes. The objects need to be stored in the nest, which is assumed to have unlimited storage capacity. A way of partitioning the object retrieval task is by separating the *harvest* and *store* sub-tasks (φ_{harvest} and φ_{store}, respectively) and allowing material transfer between individuals working on the two sub-tasks.

Figure 2 provides a conceptual representation of the environment. The environment is divided into two separate areas, referred to as *source area* and *nest area*. The two areas are separated by a temporary storage area referred to as *cache* in the rest of the paper. The cache cannot be crossed by the robots. A direct path, referred to as *corridor*, links the source area and the nest area and can be used by the robots to reach the source area from the nest area and vice versa.

The problem described above is an instance of the problem described in Sect. 3 and represented in Fig. 1, with the given task being the object retrieval task. The two sub-tasks φ_1 and φ_2 correspond to the φ_{harvest} and φ_{store} sub-tasks. The cache serves as the interface I between the two sub-tasks. The object retrieval task can be performed once by executing once the φ_{harvest} and φ_{store} sub-tasks in this order. The choice the robots face about the strategy to be employed translates in choosing whether to use the cache or the corridor: a partition

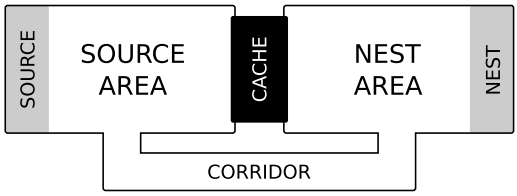

Fig. 2 Conceptual representation of the environment used to study the partitioning problem on a swarm of robots performing foraging. The environment is divided into two areas (source and nest area), separated by the cache. The robots cannot cross the cache: to reach the source area from the nest area and the other way around the robots must take the corridor. In this case the given task is retrieving objects from the source to the nest. This task can be performed as single, unpartitioned task or partitioned into two sub-tasks. The corridor allows a robot to perform the object retrieval task as unpartitioned task: an object can be directly retrieved from source to nest. The cache allows the object retrieval task to be partitioned into sub-tasks: φ_{harvest} consists in harvesting objects from the source and dropping them at the cache, φ_{store} in picking up objects from the cache and storing them in the nest

strategy makes use of the cache, a non-partition strategy makes use of the corridor. Which of the two strategies is advantageous depends on the length of the corridor and on the costs for using the cache, most notably the interfacing times (Π_1 and Π_2) that the robots have to spend at the cache for dropping or picking up an object. In the rest of the paper, we will consider the interfacing times Π_1 and Π_2 to be equal, and will indicate both of them with Π.

Note that the situation here described can be generalized to all those cases where the sub-tasks develop in physically separated areas and material can flow from one sub-task to the following on a path that differs from the one the workers can use. An example is the leaf foraging task performed by the *Atta* ants, described in Sect. 2. In this case, the two working areas are the top of the tree and the ground. The ants can reach one area from the other by climbing the tree—i.e., without partitioning. The trunk plays the same role as the corridor in Fig. 2. The leaves can instead be dropped from the top of the tree to the ground—i.e., when the foraging task is partitioned. The time spent searching the leaves defines the interfacing time Π_2, while Π_1 is null in this example. Another example is a robotic warehouse that extends on two floors, with materials that can be transferred from one floor to the other by using a dedicated lift and the robots task being transporting items from one floor to the other. The path the robots need to cover for reaching one floor from the other plays the role of the corridor. The interfacing times Π_1 and Π_2 in this case are linked to the time it takes to load and unload the lift and the time it takes by the lift to move from one floor to the other.

The state diagram of Fig. 3 represents the behavior of each individual. Each robot takes its own decision on the strategy to employ. After storing an object in the nest, a robot has to decide where to retrieve the next object. The robot has a probability P_p of trying to pick up the object from the cache (i.e., partition the task). This choice can be abandoned with a probability $Pg_{\varphi_{\text{store}}}$ that grows at each time-step (see (3)). Analogously, after harvesting an object from the source a robot has to decide where to leave it. The robot has a probability P_p of trying to drop the object in the cache (i.e., partition the task). Again, this choice can be abandoned with a probability $Pg_{\varphi_{\text{harvest}}}$ that grows at each time-step.

As mentioned in Sect. 4, the probability P_p depends on the perceived cost of the partition strategy (see (1)). This cost is computed as the sum of the estimates $\hat{t}_{\varphi_{\text{harvest}}}$ and $\hat{t}_{\varphi_{\text{store}}}$; the cost of the non-partition strategy is the estimate $\hat{t}_{\text{NP}}$ of the time it takes to retrieve an object to the nest without partitioning. The estimate $\hat{t}_{\varphi_{\text{harvest}}}$ is updated when a robot chooses to use the cache for dropping an object. It is updated with the time it takes from the moment the robot harvested an object from the source, to the moment it harvests the next one, after using the

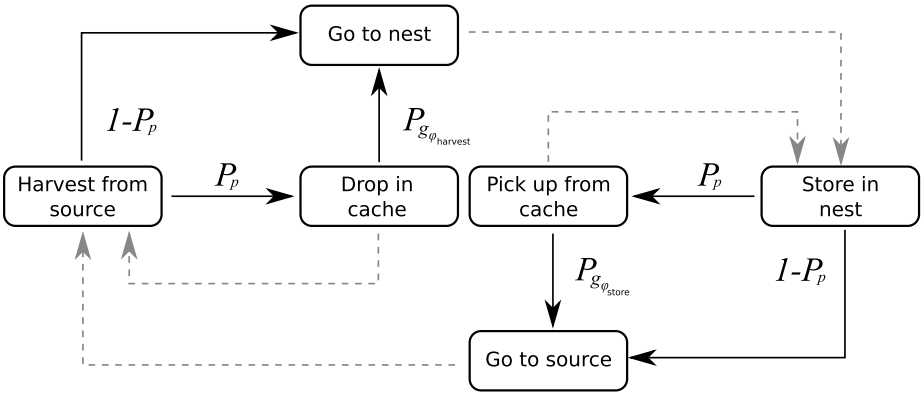

Fig. 3 Finite state machine describing the behavior of each robot. *The black continuous arcs* marked with a label represent stochastic choices that a robot can make. P_p is the probability of employing task partitioning. $Pg_{\varphi_{\text{store}}}$ and $Pg_{\varphi_{\text{harvest}}}$ are the probabilities of giving up the store and the harvest sub-tasks, respectively (see Sect. 4 for more details). *Dashed gray arcs* represent transitions when the robot does not make a choice

cache. The estimate $\hat{t}_{\varphi_{\text{store}}}$ is updated when a robot chooses to use the cache for picking up an object. It is updated with the time it takes from the moment the robot stores an object in the nest, to the moment the robot stores the next object, taken from the cache. $\hat{t}_{\varphi_{\text{harvest}}}$ and $\hat{t}_{\varphi_{\text{store}}}$ are also updated when the robot gives up dropping or picking up an object, respectively. The estimate $\hat{t}_{\text{NP}}$ is updated each time the robot chooses to use the corridor, either for reaching the source or the nest. It is updated with twice the time it took to cross the corridor, including the time to enter the source or the nest.[2] $\hat{t}_{\text{NP}}$ is not updated when a robot crossed the corridor after giving up using the cache. Giving up is a consequence of selecting the partition strategy and therefore impacts on $\hat{t}_{\varphi_{\text{harvest}}}$ and $\hat{t}_{\varphi_{\text{store}}}$.

Robots do not communicate explicitly, none of them knows what the others are doing or has any notion of the swarm's performance. The strategy observed at the level of the swarm is the result of the individuals' decisions and is a self-organized process.

5.1 Simulation

The experiments described in this article have been carried out in simulation. The simulation framework we employed is ARGoS (Pinciroli et al. 2011), developed within the Swarmanoid project.[3] ARGoS is a discrete time, physics-based simulation environment that allows the real-time simulation of large swarms of heterogeneous robots. The main characteristic of ARGoS is that it allows the user to tune the level of detail of the simulation. The level of detail can go from bi-dimensional environments governed by kinematic rules to three-dimensional environments governed by dynamics. For the work presented in this article we chose to employ a kinematics model of the robots in a two dimensional space.

The robots simulated in the experiments have a real counterpart: the e-puck[4] robot (Mondada et al. 2009). The e-puck is a small wheeled robot, developed at EPFL, Lausanne,

[2]The estimate $\hat{t}_{\text{NP}}$ is computed this way because a robot using the non-partition strategy in one direction could decide to change strategy on the way back.

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

[4]http://www.e-puck.org/.

Switzerland. ARGoS simulates all the sensors and actuators available on the e-puck. In the experiments presented in this paper we employed the wheel actuators, the eight infrared proximity sensors for light and proximity detection, the VGA camera, and the ground sensors.

5.2 Abstraction of objects

The e-pucks do not have the ability of grasping and transporting objects. Therefore, we had to overcome this limitation by using an abstraction: instead of actual objects we simulated a physical device, referred to as a *booth*, that has been developed and prototyped by us (Brutschy et al. 2010). Each booth features a light barrier and two RGB LEDs. The robot enters the booth, attracted by the LEDs that it can perceive using its RGB camera. The booth detects the presence of a robot using the light barrier. When a robot is perceived, the booth reacts by executing a user-defined logic. Reactions consist in changing the status of the RGB LEDs. In the work presented in this article booths are organized into arrays that implement the source, the nest, and the cache. Figure 4 shows a schematic representation of the cache, which is composed of three booths on each of its two sides. In the experiments, a booth whose LEDs are lit up in green represents an object available at the booth. On the other hand, a booth whose LEDs are lit up in blue, represents a free spot where an object can be dropped. With this representation, when a robot enters a booth whose LEDs are lit up in green, we assume that the robot will pick up an object from that booth. Analogously, when a robot transporting an object enters a booth whose LEDs are lit up in blue, we assume that the object being carried will be dropped in that booth. In both cases the booth acknowledges the robot presence by temporarily turning the LEDs to red, until the robot has left. The behavior of the booths changes with their location in the environment. The nest booths are always blue, representing an unlimited number of spots where objects can be stored. The source booths are always green, representing the fact that objects can always be found at the source.

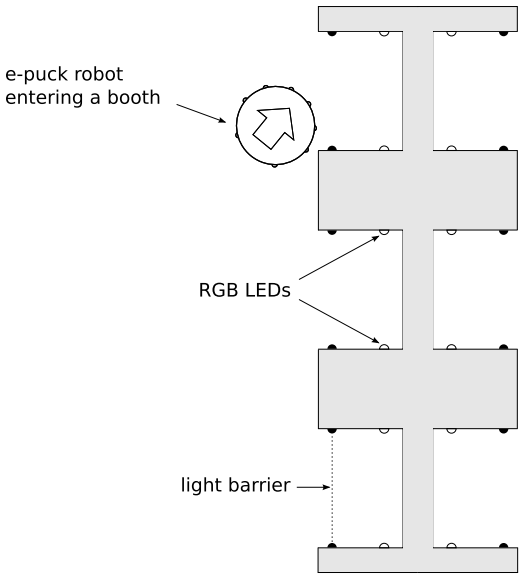

Fig. 4 Representation of an array of booths, composed of three booths on each of the two sides. Each booth can detect the presence of a robot through a light barrier, represented with the *black semicircles* and the *dotted line* in one of the booths. RGB LEDs, represented with the *blank semicircles*, can signal objects or drop spots by lighting up in different colors

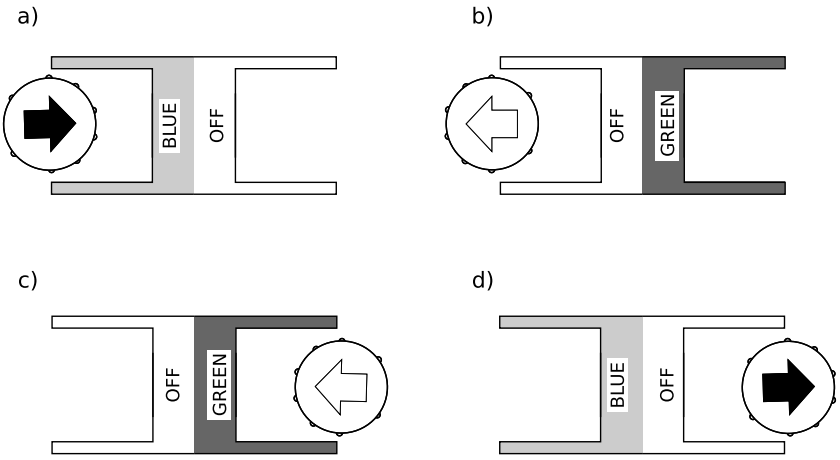

Fig. 5 The behavior of a cache implemented with two paired booths. The source (not shown) is located at the left and the nest (not shown) is located at the right. The robots carrying an object are marked with a *black arrow*, and the robots not carrying objects with a *white arrow*. (**a**) Initially the cache is empty: LEDs are blue on the source side and off on the nest side. (**b**) A robot working on the harvest sub-task enters the booth to drop an object that was harvested in the source. Once finished, the LEDs of the booth on the source side turns off while the LEDs of the booth on the nest side turn to green to represent an available object. (**c**) A robot working on the store sub-task enters the booth to pick up the object. (**d**) Once the robot leaves toward the nest to store the object, the booth returns to its initial configuration. When a booth perceives the presence of the robot, its LEDs temporarily turn to red, so that the robot realizes it is inside the booth (not shown in the above sequence of events)

The behavior of the cache is more complex. At the beginning the cache booths facing the source are lit up in blue, and those facing the nest have their LEDs turned off (Fig. 5a). In this configuration the cache is empty. Once a robot has dropped an object in the cache, by subsequently entering and exiting a free (i.e., blue) booth facing the source side, the booth turns off so that it is no longer available to accept an object, and the corresponding booth on the nest side turns to green (Fig. 5b). This represents an object deposited in the cache that becomes immediately available on the nest side. A robot working on the store sub-task can pick up this object by subsequently entering and exiting the booth on the nest side (Fig. 5c). Once the object has been picked up, the nest side of the booth turns off and the source side turns to blue, to signal that the booth is available again for dropping an object (Fig. 5d).[5]

The robots themselves keep track of whether they are carrying an object or not. A robot without an object assumes it has picked up an object from a booth if it perceives the acknowledgment (i.e., the booth's LEDs turn to red) and if the booth's LEDs were green before. Conversely, a robot carrying an object assumes it has dropped an object in a booth if it perceives the acknowledgment and the booth's LEDs were blue before. Given the geometry of the robots and of the booths, there is no room for errors in the acknowledgment mechanism: the booth can perceive a robot only when it is completely inside and only in that case its LEDs turn to red and remain red until the robot has left (i.e., the robot perceived the acknowledgment).

The only possible inconsistency happens if a robot enters a booth unintentionally. This is not a problem for the source and nest booths: they would simply temporarily turn their

[5]A video illustrating the behavior of the cache can be found in the online supplementary material, see also Pini et al. (2011b).

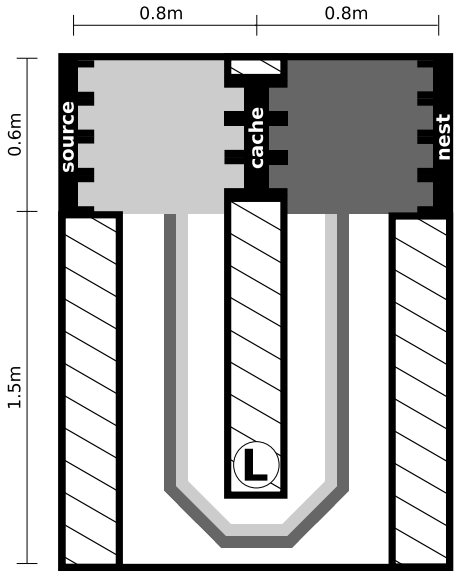

Fig. 6 The environment used in the simulation experiments. Nest and source arrays are composed of four booths, the cache array is composed of three booths for each side. The color of the ground is used by the robots as a clue to recognize their location in the arena. The light source, marked with "L", is used by the robots as a landmark for navigation. A colored path in the corridor is followed by the robots when navigating through the corridor

LEDs to red and then return to their default states. The cache instead is affected by such an event, as its state could become inconsistent. The event here mentioned is very unlikely, but still observable; all the results reported in this work are taken from the experimental runs in which cache inconsistencies were not observed.

5.3 Environment

The environment in which the robots have to perform foraging is shown in Fig. 6. The source booths are located at the top-left and the nest booths at the top-right corners of a 1.6 m by 2.1 m rectangular arena surrounded by walls. The cache booths are located between source and nest. The different areas of the arena are marked with a specific ground color, which can be perceived by the robots and used to determine their location in the arena. A light source, located on the bottom of the arena, is used as a landmark for navigation. A colored path marks the floor in the corridor, and is followed by the robots when moving from one area to the other.

6 Experiments and results

To test the properties of the proposed method we run four sets of experiments. The goal of the first set of experiments is to understand how the parameters of the proposed method impact on the system and derive guidelines on how to assign values to them. The goal of the second set of experiments is to assess the performance of the method. The goal of the third set of experiments is to study the responsiveness of the method to changes in the environment. Finally, the goal of the fourth set of experiments is to study the scalability of the method.

Each experimental run lasts a total of 600,000 simulation steps of 0.1 s each. This amounts to a simulated time of sixteen hours and forty minutes. In the experiments we simulate a swarm of ten e-pucks; in the scalability experiment we study the performance of

Table 1 Parameters space used for the analysis of variance. The values define a grid of points in the parameters space; ANOVA then performs a regression using these points

Parameter	Values
Interfacing time Π (seconds)	0, 100, 200
Weighted average factor α	0.5, 0.75, 1.0
Strategy selection steepness S	0.5, 5, 10
Give up partitioning steepness K	-0.5, -1.5, -2.5
Give up partitioning offset O	-1, -5, -9

swarms of various size. At the beginning of each run half of the swarm is randomly positioned in the nest area, the rest of the swarm in the source area. The speed of the robots is set to 3.5 cm/s, a value determined while performing tests with the real robots. This relatively low speed is due to the limited computational capabilities of the e-puck. The camera data cannot be processed at high speeds and therefore the robots need to move slowly in order to obtain precise movements toward a booth. To avoid bias in the robots' behavior, the estimates $\hat{t}_{\varphi_{\text{harvest}}}$, $\hat{t}_{\varphi_{\text{store}}}$, and $\hat{t}_{\text{NP}}$ are randomly initialized: $\hat{t}_{\varphi_{\text{harvest}}}$ and $\hat{t}_{\varphi_{\text{store}}}$ are uniformly sampled in [50, 100], $\hat{t}_{\text{NP}}$ in [100, 200].

The rest of the section describes in detail each set of experiments and presents the results obtained within each of them.

6.1 First set of experiments: ANOVA

To get an insight on the impact of each parameter of the adaptive method, we analyze the system using a well known design of experiments method: analysis of variance (ANOVA) (Cox and Reid 2000). Table 1 summarizes the parameters included in ANOVA and the possible values they can assume in the analysis. For each combination of the parameters values, we run 15 randomly seeded simulations. What follows is a summary of the results of ANOVA. For the complete results we refer the reader to Pini et al. (2011b).

We perform four analyses, one including Π as factor, and other three fixing Π to a value and using the remaining parameters as factors. The four analyses agree in indicating that a high value of the weight factor α should be used, independently of the value of the interfacing time Π. This is due to the fact that for low values of α, the swarm is not able to quickly estimate the costs of each strategy; this leads to poor performance. The effect of the steepness S depends on the value of Π: the higher Π, the lower S should be, and vice versa. Therefore the value of S should be chosen depending on the specific environment. If the value of Π is known, and it does not vary in time, S can be selected accordingly. On the other hand, if the value of the Π is unknown or can vary significantly over time, then S should be set to an intermediate value. An alternative, not implemented in this work, would be to adaptively select the value of S on the basis of a measure of the value of Π. The parameters K and O have a strong impact on the performance of the system. Unfortunately, the results of the analyses do not recommend any particular value for these parameters. To address this issue, ANOVA needs more data and higher order interactions should be studied to have deeper insights concerning how to select the two parameters values.

Please note that the goal of the analysis of variance is not the optimal selection of the parameters, but the study of the main effects each parameter has on the system. In accordance with the results of ANOVA, we chose a high value for the parameter α (0.8). Since we want our method to be able to deal with both high and low values of Π, we set S to 5.0. As mentioned, our analysis did not provide enough information for selecting the parameters K and O, with the exception that some combinations of the two were shown to be bad choices.

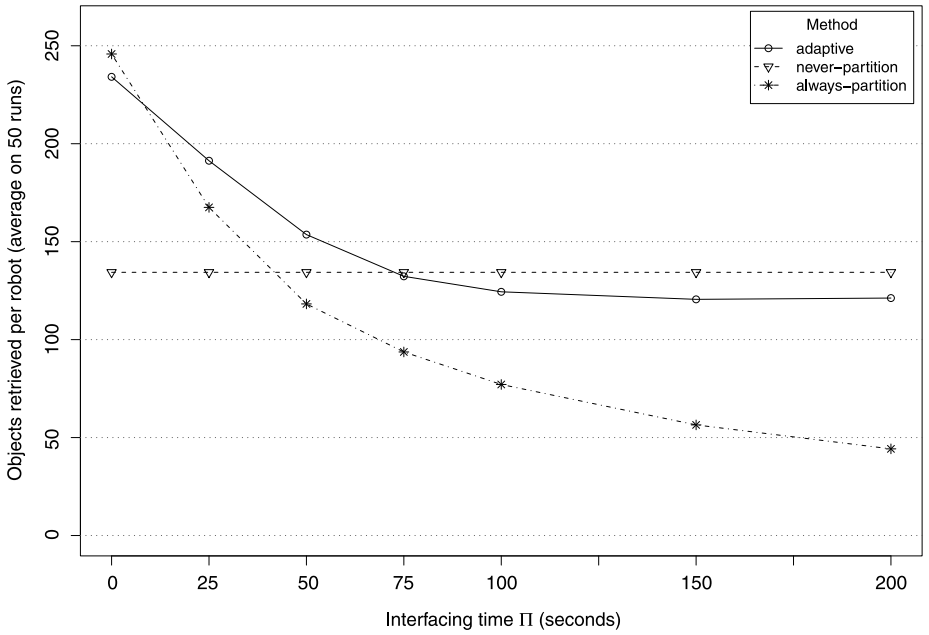

Fig. 7 Average amount of objects retrieved per robot at the end of the experiment for different values of the interfacing time Π. The same plot, also reporting the 95% confidence intervals (omitted for clarity here), can be found online (see Pini et al. 2011b)

The values of K and O have been determined through trial and error, and their values have been set to $K = -0.5$ and $O = -1$. All the results presented in the following have been obtained with this set of parameters.

6.2 Second set of experiments: performance of the adaptive method

To assess the performance of the adaptive method, we compare it with the performance of two reference methods, referred to as *never-partition* and *always-partition* methods. The never-partition method always makes use of the corridor ($P_p = 0$), while the always-partition method always makes use of the cache ($P_p = 1$) without giving up ($Pg_{\varphi_{\text{store}}} = Pg_{\varphi_{\text{harvest}}} = 0$). We tested seven different settings, each corresponding to a different value of the interfacing time Π. The values we studied for Π are 0, 25, 50, 75, 100, 150, and 200 (values are in seconds, simulated time). The upper bound for the value of Π was selected so that the resulting cost for using the cache is considerably higher than the one for using the corridor. In fact, a robot takes an average time that is roughly 400 s for traveling along the corridor, while using the cache requires roughly 120 s when $\Pi = 0$ s.[6]

The graph in Fig. 7 shows the results of the second set of experiments. The graph plots the average performance of the three methods for different values of the interfacing time Π. As expected, each reference method performs well only for a subset of values of Π.

[6]The time needed for traveling along the corridor was measured using the never-partition method, the time for using the cache using the always-partition method. In a more general case, where the robots can choose the strategy to employ, the values can differ as they depend on how cache and corridor are employed.

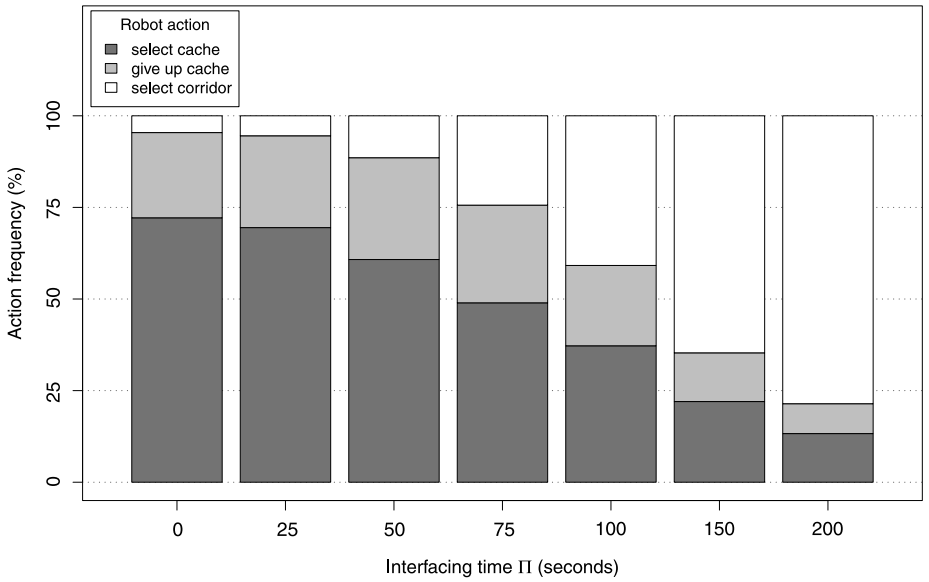

Fig. 8 Actions performed by the robots using the adaptive method, for different values of the interfacing time Π. *Each bar* reports, for each value of Π, the percentage of times an action was performed. The actions reported are: selection and actual usage of the cache (*dark gray*), selection of the cache and give up (*light gray*), and selection of the corridor (*white*). The percentage of times the robots chose to employ the cache is the sum of the values reported by the *light gray* and the *dark gray bars*. The values reported are averages computed over 50 experimental runs

The always-partition method performs well when Π is low. In this case, the time needed for traveling along the corridor is much higher than the one for using the cache, therefore the corridor should be avoided and the given task should be partitioned into sub-tasks. On the other hand, the never-partition method performs better when the cache is costly: using the corridor is preferable for high values of Π. The adaptive method is able to perform well in both cases, showing good performance on the whole spectrum of the parameter Π.

Figure 8 provides a summary of the strategy employed by the robots in the swarm when using the adaptive method, for the different values of Π. Each bar reports the percentage of times that each action was performed by the robots; the reported values are averages over 50 runs. The graph confirms that the robots select the corridor with a higher frequency for increasing values of Π. For small values of Π, the preferred choice is the cache.

Figure 9 reports the time needed by the robots to use the corridor (t_{NP}) and the cache ($t_{\varphi_{\mathrm{harvest}}}$ and $t_{\varphi_{\mathrm{store}}}$) employing the three different methods, when the interfacing time is 0 s (left), 25 s (center), and 50 s (right). When $\Pi = 0$ s the always-partition method performs better than the never-partition method (see Fig. 7). Figure 9 (left) shows that the time needed to pick up an object from the cache ($t_{\varphi_{\mathrm{store}}}$) is considerably lower than the one needed to cross the corridor(t_{NP}). This means that the objects are delivered to the nest faster by using the cache. Despite the fact that less robots are contributing to depositing objects in the nest (half of the swarm works on the source side), the boost in terms of speed that comes from using the cache makes the always-partition strategy preferable to the never-partition strategy. For increasing values of Π, the relative gain in terms of speed progressively decreases (Fig. 9, center and right), and for $\Pi = 50$ s always using the corridor becomes more advantageous than never using it (see performance in Fig. 7).

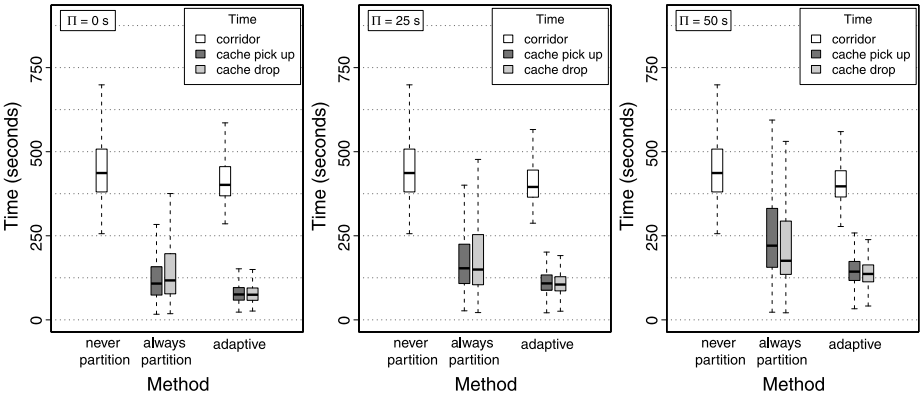

Fig. 9 Time needed for using the corridor and the cache for the three different methods. The *graphs* report the data for $\Pi = 0$ s (*left*), $\Pi = 25$ s (*center*), and $\Pi = 50$ s (*right*)

Figure 9 shows that with the adaptive method the times needed to access the cache ($t_{\varphi_{\text{store}}}$ and $t_{\varphi_{\text{harvest}}}$) are always lower than with the always-partition method. This can be explained by the fact that with the always-partition method all the robots try to access the cache. This increases the competition for accessing the cache booths, and it is likely that a robot finds the cache busy when trying to use it. For large values of Π, the time the robots have to wait for a free cache booth is also large, as the robots using the cache spend more time inside the booths (as result of a higher Π). This effect can be seen on Fig. 9 in the fact that the variance of the pick up and drop time increases with the value of Π, when using the always-partition method. The adaptive method can reduce the competition when the cache is not very advantageous (intermediate values of Π) by increasing the frequency at which the robots use the corridor (see actions reported in Fig. 8). Additionally, the robots employing the adaptive method can benefit from the give up mechanism which prevents them for waiting too long when the cache is not immediately available.

6.3 Third set of experiments: adaptivity to changes

We test the adaptiveness of the method in response to a sudden variation in the environmental conditions. The effect of the variation is to modify the relative costs of the two strategies. In order to achieve this, we change the value of the interfacing time Π when the experiment reaches half of its duration. We consider two cases: one in which Π changes from 200 s to 0 s and another in which it changes from 0 s to 200 s. In the first case it is expected that the corridor is used more frequently by the swarm in the initial phase and, once Π changes to zero, the usage of the cache becomes more frequent; the other way around is expected to happen in the second case. Figure 10 reports the results of the two experiments, showing that by employing the method proposed in this paper the swarm is able to adapt the strategy. The graph of Fig. 10 (top) refers to the first case described (decrease of Π); the graph of Fig. 10 (bottom) to the second (increase of Π). Each graph provides a summary of the data collected in the corresponding 50 experimental runs. In the graphs of Fig. 10, the total time frame of the experiment has been divided into windows of 1 hour. Each point of the plot reports, for each strategy, the percentage of usage in the time window preceding the value indicated on the X axis. The cases in which the robots give up using the cache are counted as usage of the corridor.

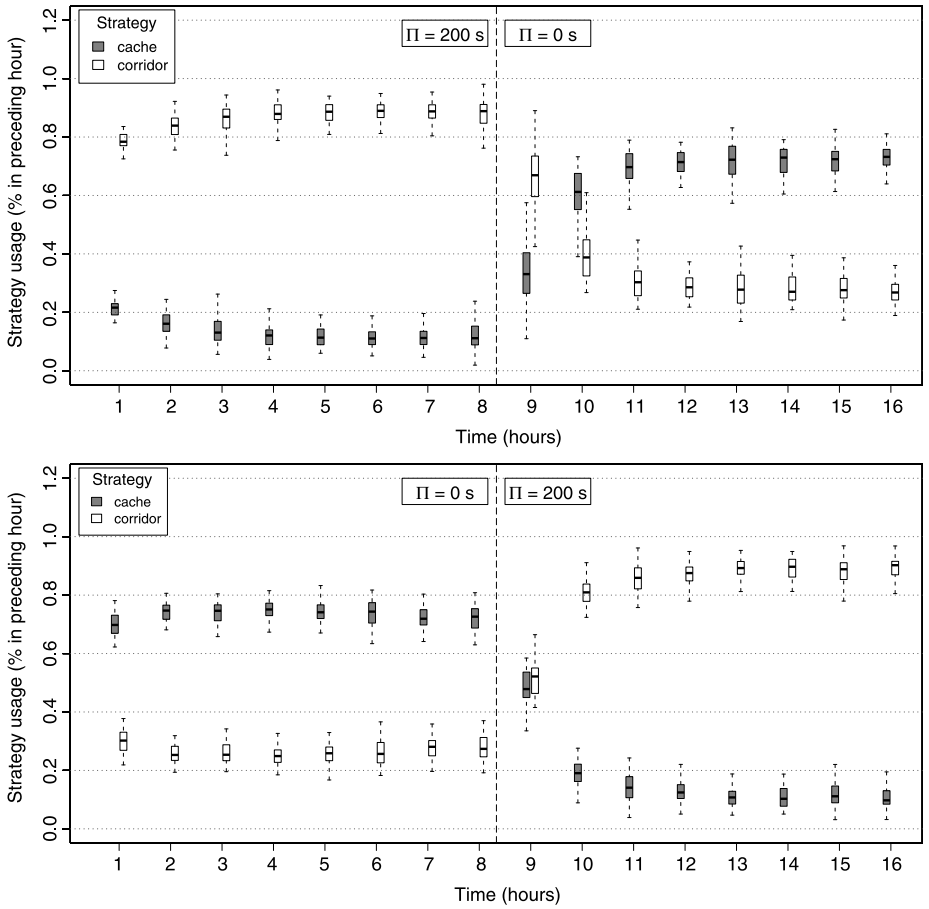

Fig. 10 Strategy used in time when the interfacing time Π is varied from 200 s to 0 s (*top*) and from 0 s to 200 s (*bottom*). The *vertical dashed line* marks the instant in which Π is changed (time $t = 8$ hours and 20 minutes). The total duration of the experiment is divided into windows of 1 hour, *each box* reports the percentage of usage (over 50 experimental runs) of each strategy in the time window preceding the value reported on the X axis

In both cases, at half the experiment time, one of the two strategies is employed more frequently than the other, indicating that the swarm identifies it as the best strategy. The results at the end of both experiments show that the swarm reacts to the change in the environmental conditions. In fact, the strategy of the swarm changes after the value of Π is modified. The swarm converges to a new strategy more suited to the new value of Π.

6.4 Fourth set of experiments: scalability

We test the scalability of the system when using the adaptive method and the two reference methods, for $\Pi = 0$ s and $\Pi = 25$ s. We vary the swarm size using the following values: 4, 6, 8, 10, 14, 18, 22, 26, and 30. In the following, we will refer to *area coverage* as the percentage of the total area actually covered by the robots. The area coverage depends on the size of the swarm. As the total area of the environment is roughly 2.5 m^2 and the radius

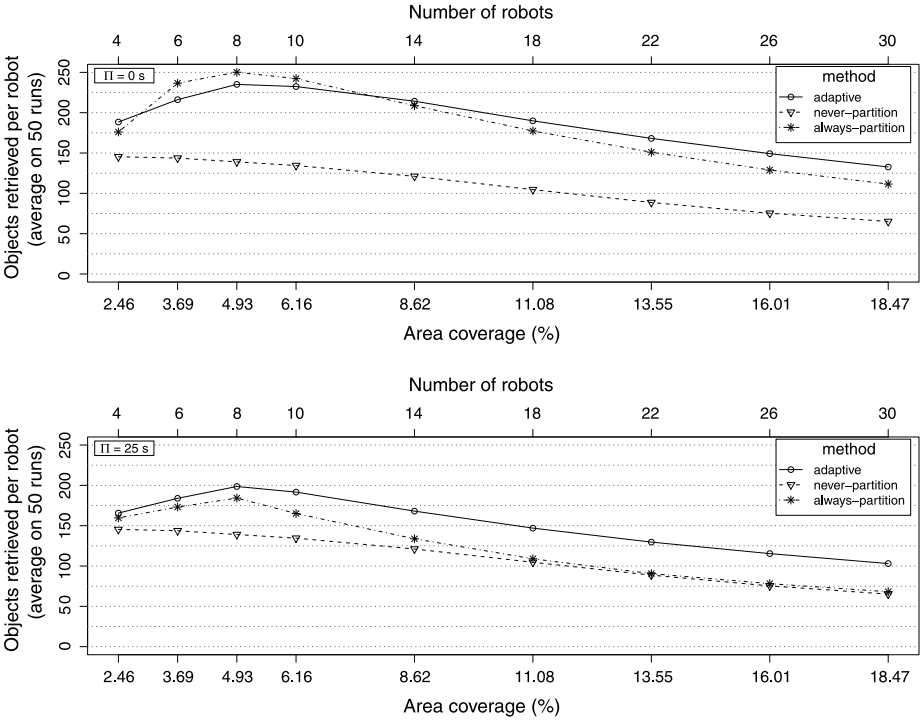

Fig. 11 Results of the scalability experiment for $\Pi = 0$ s (*top*) and $\Pi = 25$ s (*bottom*). The *plots* report, for each of the three methods, the average individual performance for different area coverages (the *top axis* reports the corresponding number of robots). The *same plot*, also reporting the 95% confidence intervals (omitted for clarity here), can be found online (see Pini et al. 2011b)

of an e-puck is 70 mm, the area coverage varies from a minimum of 2.46%, when the swarm is composed of four robots, to a maximum of 18.47%, when the swarm is composed of 30 robots.[7] Figure 11 reports the results of the scalability experiments for $\Pi = 0$ s (top) and $\Pi = 25$ s (bottom). The graphs report, for each of the three methods, the average number of objects retrieved by each robot for different area coverages. The graphs show that, using the never-partition method, the individual performance constantly decreases for increasing swarm sizes. This is due to a growing interference, that progressively makes it harder to navigate through the corridor. The adaptive method and the always-partition method instead, have a peak in performance when the area coverage is 4.93% (eight robots), for both the values of Π here considered. This suggests that, in the experimental setup presented in this work, a number of eight robots is the optimal size for the swarm in order to exploit the cache at the maximum efficiency. With more than eight robots, both the adaptive method and the always-partition method decrease in performance, but the adaptive method does so at a slower rate. Thus, the method shows to be more scalable. The reason for which the performance of the adaptive method is higher than the one of the always-partition method resides in the fact that it can balance the number of robots trying to use the cache and the corridor. This reduces the competition for accessing the cache (thus the expected waiting time) and increases the navigability in the corridor with respect to the never-partition method.

[7]The area of the arena is computed excluding the space available inside each booth.

7 Conclusions

The work presented in this article is part of a broader research that aims at conceiving methods that allow a swarm to partition complex tasks in smaller, manageable sub-tasks. Partitioning tasks into sub-tasks and defining interfaces of these sub-tasks in an autonomous way is a major challenge. Nevertheless, we are confident that task partitioning can make a significant contribution to the future of swarm robotics. Swarms of robots can obtain benefits from task partitioning in terms of reduction of interference between individuals, better exploitation of specialization and heterogeneity, parallelism, and efficiency gains. Task partitioning also entails costs in terms of synchronization overheads where sub-tasks interface with each other.

In the work presented in this paper, we focused on a task pre-partitioned into a sequence of two sub-tasks that interface with each other in an asynchronous way. We tested different experimental conditions, considering settings in which it is, or it is not, advantageous to use a partition strategy. The settings are different in terms of the cost of using a partition strategy for solving the overall task.

We proposed an adaptive method that allows a swarm of robots to decide whether to employ task partitioning or not depending on the environmental conditions. The method is fully distributed and relies on the individuals' perception of the cost of employing task partitioning. Task partitioning at the level of the swarm is a self-organized process that results from individual decisions. We tested the method in simulation, using foraging as testbed and comparing the proposed method to two reference methods. The results show that with the proposed method the swarm selects a strategy that suits the specific environment, leading to good performance in the different cases we considered. We also performed experiments with the aim of assessing the capability of the swarm to react to changes in the environmental conditions. The results show that the swarm is able to respond to changes and to adapt the strategy being employed. Scalability tests show that the adaptive method performs well across different swarm sizes.

The work presented here is a first step toward a more general method for tackling a general task-partitioning problem, in which there are more than two sub-tasks. Short term goals for future research will concern studying how the general N sub-tasks case can be derived from the two sub-tasks case presented here and what modifications need to be done to the method we proposed in this work. A straightforward approach would be to apply the method to a series of N sub-tasks, with each robot deciding either to perform the overall task, or one of the sub-tasks. We believe the method can be transferred to such a situation without any modification. Nevertheless, this needs to be verified experimentally. Another possibility would be a top-down recursive approach, where (sub-)tasks can be partitioned at different levels of granularity, independently of each other. This would allow the swarm to adopt "hybrid" strategies, with some parts of the overall task partitioned in atomic sub-tasks and others in higher-level sub-tasks. We think such an approach would be extremely powerful and flexible, but it would also require major modifications to the proposed method for explicitly taking into account how (sub-)tasks relate to each other.

Another direction for future work is the addition of a social component to the system, using explicit local communication within the swarm. For example, the robots could communicate to each other their perception of the costs associated to each strategy, and take decisions based on the information received. The swarm might benefit from communication in terms of a faster responsiveness and lower sensitiveness to noisy conditions. The method proposed in this paper would require minor modifications in order to take into account the information received and to use it to update the costs estimates.

Acknowledgements Marco Dorigo acknowledges support by the European Union through the ERC Advance Grant "E-SWARM: Engineering Swarm Intelligence Systems" (contract 246939). Marco Dorigo, Mauro Birattari, and Arne Brutschy acknowledge support from the Belgian F.R.S.–FNRS. Marco Frison acknowledges support from "Seconda Facoltà di Ingegneria", Alma Mater Studiorum, Università di Bologna. Andrea Roli acknowledges support from the "Brains (Back) to Brussels" 2009 programme, funded by the Institut d'encouragement de la Recherche Scientifique et de l'Innovation de Bruxelles (IRSIB). The authors thank Prasanna Balaprakash for his help and suggestions during the preparation of this paper. The authors also thank the reviewers for their suggestions and advice for improving the work presented here.

References

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

Aho, A. (1983). *Data structures and algorithms*. Boston: Addison-Wesley.

Anderson, C., & Jadin, J. L. V. (2001). The adaptive benefit of leaf transfer in *Atta colombica*. *Insectes Sociaux*, *48*, 404–405.

Anderson, C., & Ratnieks, F. L. W. (1999). Worker allocation in insect societies: coordination of nectar foragers and nectar receivers in honey bee (*Apis mellifera*) colonies. *Behavioral Ecology and Sociobiology*, *46*(2), 73–81.

Anderson, C., Boomsma, J. J., & Bartholdi, J. J. (2002). Task partitioning in insect societies: bucket brigades. *Insectes Sociaux*, *49*, 171–180.

Beni, G. (2005). From swarm intelligence to swarm robotics. In E. Şahin & W. M. Spears (Eds.), *Lecture notes in computer science: Vol. 3342*. *Swarm robotics* (pp. 1–9). Berlin: Springer.

Bonabeau, E., Theraulaz, G., & Deneubourg, J.-L. (1996). Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proceedings of the Royal Society of London. Series B*, *263*(1376), 1565–1569.

Bonabeau, E., Sobkowski, A., Theraulaz, G., & Deneubourg, J.-L. (1999). Adaptive task allocation inspired by a model of division of labor in social insects. In D. Lundh, B. Olsson & A. Narayanan (Eds.), *Biocomputing and emergent computation* (pp. 36–45). Skövde: World Scientific.

Brutschy, A., Pini, G., Baiboun, N., Decugnière, A., & Birattari, M. (2010). The IRIDIA TAM: a device for task abstraction for the e-puck robot. Technical Report TR/IRIDIA/2010-015, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Cox, D. R., & Reid, N. (2000). *The theory of the design of experiments*. London: Chapman and Hall/CRC.

Fontan, M. S., & Matarić, M. J. (1996). A study of territoriality: the role of critical mass in adaptive task division. In P. Maes, M. J. Matarić, J.-A. Meyer, J. Pollack & S. Wilson (Eds.), *From animals to animats 4: proceedings of the 4th international conference of simulation of adaptive behavior* (pp. 553–561). Cambridge: MIT Press.

Fowler, H. G., & Robinson, S. W. (1979). Foraging by Atta sexdens (Formicidae: Attini): Seasonal patterns, caste and efficiency. *Ecological Entomology*, *4*(3), 239–247.

Gordon, D. M. (1999). Interaction patterns and task allocation in ant colonies. In C. Detrain, J. M. Pasteels & J.-L. Deneubourg (Eds.), *Information processing in social insects* (pp. 51–67). Basel: Birkhäuser.

Hart, A. G., & Ratnieks, F. L. W. (2000). Leaf caching in *Atta* leafcutting ants: discrete cache formation through positive feedback. *Animal Behaviour*, *59*(3), 587–591.

Hart, A. G., & Ratnieks, F. L. W. (2001a). Leaf caching in the leafcutting ant *Atta colombica*: organizational shift, task partitioning and making the best of a bad job. *Animal Behaviour*, *62*(2), 227–234.

Hart, A. G., & Ratnieks, F. L. W. (2001b). Task partitioning, division of labour and nest compartmentalisation collectively isolate hazardous waste in the leafcutting *Atta cephalotes*. *Behavioral Ecology and Sociobiology*, *49*, 387–392.

Hart, A., Anderson, C., & Ratnieks, F. L. W. (2002). Task partitioning in leafcutting ants. *Acta Ethologica*, *5*, 1–11.

Hubbell, S. P., Johnson, L. K., Stanislav, E., Wilson, B., & Fowler, H. (1980). Foraging by bucket-brigade in leaf-cutter ants. *Biotropica*, *12*(3), 210–213.

Jeanne, R. L. (1986). The evolution of the organization of work in social insects. *Monitore Zoologico Italiano*, *20*, 119–133.

Jeanne, R. L. (2002). Social complexity in the Hymenoptera, with special attention to the wasps. In T. Kikuchi, N. Azuma, & S. Higashi (Eds.), *Proceedings of the 14th congress of the IUSSI* (pp. 81–130). Sapporo: Hokkaido University Press.

Kalra, N., & Martinoli, A. (2006). Comparative study of market-based and threshold-based task allocation. In M. Gini & R. Voyles (Eds.), *Distributed autonomous robotic systems 7* (pp. 91–101). Tokyo: Springer.

Krieger, M. J. B., & Billeter, J.-B. (2000). The call of duty: self-organized task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, *30*(1–2), 65–84.

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.

Lein, A., & Vaughan, R. (2008). Adaptive multi-robot bucket brigade foraging. In S. Bullock, J. Noble, R. Watson & M. A. Bedau (Eds.), *Artificial life XI: proceedings of the 11th international conference on the simulation and synthesis of living systems* (pp. 337–342). Cambridge: MIT Press.

Lopes, J. F., Forti, L. C., Camargo, R. S., Matos, C. A. O., & Verza, S. S. (2003). The effect of trail length on task partitioning in three *Acromyrmex* species (Hymenoptera: Formicidae). *Sociobiology*, *42*(1), 87–91.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J. C., Floreano, D., & Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In P. J. S. Gonçalves, P. J. D. Torres & C. M. O. Alves (Eds.), *Proceedings of the 9th conference on autonomous robot systems and competitions* (pp. 59–65). Castelo Branco: IPCB: Instituto Politècnico de Castelo Branco.

Østergaard, E. H., Sukhatme, G. S., & Matarić, M. J. (2001). Emergent bucket brigading: a simple mechanisms for improving performance in multi-robot constrained-space foraging tasks. In *AGENTS '01: proceedings of the 5th international conference on autonomous agents* (pp. 29–30). New York: ACM Press.

Parker, L. E. (1998). ALLIANCE: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, *14*(2), 220–240.

Parker, C. A. C., & Zhang, H. (2010). Collective unary decision-making by decentralized multiple-robot systems applied to the task-sequencing problem. *Swarm Intelligence*, *4*(3), 199–220.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Stirling, T., Gutiérrez, A., Gambardella, L. M., & Dorigo, M. (2011). ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems* (*IROS*) (pp. 5027–5034). Los Alamitos: IEEE Comput. Soc.

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. A. Cetto et al. (Eds.), *LNEE: Vol. 85. Informatics in control, automation and robotics: selected papers from the international conference on informatics in control, automation and robotics* (pp. 217–228). Berlin: Springer.

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—online supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2011-003/.

Ratnieks, F. L. W., & Anderson, C. (1999). Task partitioning in insect societies. *Insectes Sociaux*, *46*(2), 95–108.

Reyes, J. L., & Fernández Haeger, J. (1999). Sequential co-operative load transport in the seed-harvesting ant *Messor barbarus*. *Insectes Sociaux*, *46*, 119–125.

Şahin, E. (2005). Swarm robotics: from sources of inspiration to domains of application. In E. Şahin & W. M. Spears (Eds.), *Lecture notes in computer science: Vol. 3342*. *Swarm robotics* (pp. 10–20). Berlin: Springer.

Scheidler, A., Merkle, D., & Middendorf, M. (2008). Stability and performance of ant queue inspired task partitioning methods. *Theory in Biosciences*, *127*(2), 149–161.

Seeley, T. D. (1989). Social foraging in honey bees: how nectar foragers assess their colony's nutritional status. *Behavioral Ecology and Sociobiology*, *24*, 181–199.

Shell, D. J., & Matarić, M. J. (2006). On foraging strategies for large-scale multi-robot systems. In *Proceedings of the 19th IEEE/RSJ international conference on intelligent robots and systems* (*IROS*) (pp. 2717–2723). Pitscataway: IEEE Press.

Theraulaz, G., Bonabeau, E., Solé, R. V., Schatz, B., & Deneubourg, J.-L. (2002). Task partitioning in a ponerine ant. *Journal of Theoretical Biology*, *215*, 481–489.

Winfield, A. F. T. (2009). Towards an engineering science of robot foraging. In H. Asama, H. Kurokawa, J. Ota & K. Sekiyama (Eds.), *Distributed autonomous robotic systems 8* (pp. 185–192). Berlin: Springer.