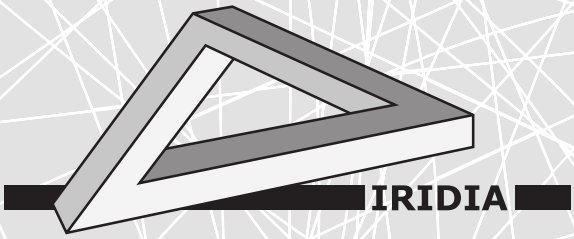

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

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

Giovanni Pini, Arne Brutschy, Marco Frison,
Andrea Roli, Marco Dorigo, and Mauro Birattari

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2011-013

May 2011

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
UNIVERSITÉ LIBRE DE BRUXELLES
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2011-013

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

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

May 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. Swarms of robots can benefit from task partitioning in the same way social insects do. Despite this, 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. The sub-tasks interface with each other in an asynchronous way through caches. We propose a method that allows a swarm of robots to decide which strategy to use for tackling the task: a strategy that makes use of task partitioning or one that does not. The method is self-organized, relies on the experience of each individual, and it does not require explicit communication. We evaluate the method in simulation experiments, using a foraging task 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 on-line the partitioning strategy. Scalability experiments show that the proposed method performs well across all the tested group sizes.

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 levels. 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 in 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, the types of 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, the reduction of interference between workers, the better exploitation of individuals skills and specializations, as well as increased energy efficiency. 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.

Swarm robotics often draws inspiration from social insects in the implementation of distributed robotic systems. As in social insects, the individuals are simple with respect to the task they have to solve, and the control of a swarm of robots 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. Many of these problems are also faced by social insects, for example foraging, collective transport of heavy objects, self-assembly, exploration, and collective decision making (Beni, 2005; Şahin, 2005). Since there are cases in which insects obtain benefits by employing task partitioning, it is interesting to study whether the same is true also 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 task: a strategy that makes use of task partitioning, and decomposes a task into a sequence of sub-tasks, or one that does not. The method is fully distributed, based on 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 a foraging task, pre-partitioned into a sequence of two sub-tasks. We study the environmental conditions under which partitioning the task is advantageous. The method proposed in this paper, allows each robot in the swarm to make a choice depending on its perception of these environmental conditions: either to perform the foraging task as a whole or to employ task partitioning by performing one of the two sub-tasks. We study environments that differ in the cost ratio between the two choices. We show that the method proposed 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. Section 4 presents the method we propose for tackling this problem. Section 5 presents the experimental framework we use. 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 works

Task partitioning and task allocation are ways of organizing work in groups of individuals. 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 partitioned tasks. The reader can find information and examples highlighting the relation between task partitioning and task allocation in Ratnieks and Anderson (1999) and Anderson et al. (2002, 2001).

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) describe the hunting strategy of the ponerine ant *Ectatomma ruidum*: The hunting task is partitioned 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 reason why this strategy is employed. This example highlights how the use of task partitioning allows to better exploit 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. This task is partitioned into two subtasks: first the garbage is removed by the ants working in the garden and then stored in the garbage heaps by other workers. This way of organizing their work allows the ants to isolate heap workers from the rest of the colony, as they are possibly contaminated by hazardous pathogens. This 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 by which this can 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 areas, where materials can be dropped and picked up. Caches allow the individuals to coordinate their activities 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 and dropping leaves to the ground, which serves as cache. On the ground, leaves are cut in pieces and transported to the nest by other workers. The advantage of partitioning stems from the fact that only few individuals need to climb the tree, thus reducing the energy requirements at swarm level. 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. (2009). Shell and Matarić (2006) and Østergaard et al. (2001) compared homogeneous foraging and bucked brigade algorithms. They focused on the importance of spatial sub-division 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 robots' working areas in response to the interference experienced by the robots.

To the best of our knowledge, no work has been published so far in which the robots autonomously decide whether to partition a given task or not, on the basis of the characteristics of the environment and of the task.

3 Definition of the task-partitioning problem

This work focuses on the case in which a task can be partitioned into two sub-tasks. These sub-tasks are sequentially interdependent, meaning that they have to be executed in order. This situation is depicted in Figure 1: a task Φ can be divided into two sub-tasks φ_1 and φ_2 that have to be performed in a certain sequence in order to perform the overall task once. The output of the sub-task φ_1 serves as input to the sub-task φ_2. We consider the case in which the sub-tasks interface with each other in an asynchronous way: the output of φ_1 can be stored at an interface location, I, of finite storage capacity, waiting to be processed in the sub-task φ_2.

As an example, let us consider the testbed employed in this work: foraging. Foraging is a specific instance of material transportation, where the overall task consists in

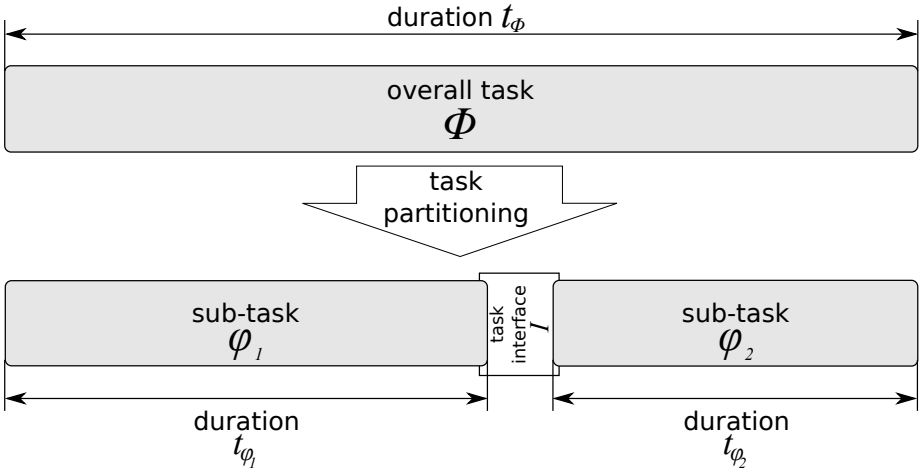

Figure 1: Representation of sequential task partitioning with asynchronous coordination. An overall task Φ, which requires a time t_Φ to be completed as a whole, is partitioned into a sequence of 2 sub-tasks φ_1 and φ_2, that require respectively a time t_{φ_1} and t_{φ_2} for their completion. The two sub-tasks are linked one to the other by means of an interface I. In general, the time needed to perform the sub-task φ_1 differs from the one needed to perform φ_2.

harvesting objects from the environment and storing them in a target location. Consider the case in which the overall foraging task is partitioned into an harvest and a store sub-tasks. The use of a cache, as described in Section 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 complete the foraging task once, the harvest and the store sub-tasks have to be performed once in the correct order.

Returning to Figure 1, the overall task Φ can be performed using two different strategies. One strategy is to employ task partitioning, and tackle each of the two sub-tasks φ_i separately. An alternative is not to employ task partitioning and tackle directly the whole task Φ. In the following, we will refer to these strategies as the *partition strategy* and *non-partition strategy*, respectively. Each strategy entails a cost that is related to the nature of the tasks and of the environment. This cost can be expressed in different ways, depending on the context. Examples of possible cost measures are energy, time, and other resources required to perform the tasks. In this study we use the total time required to perform the overall task as the cost of a strategy. Under this definition, the cost of the non-partition strategy is the time t_Φ needed to perform the task Φ; 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 shortest time to complete the overall task. Notice that the time needed to perform a (sub-)task not only depends on the nature of the task itself, but also depends on the strategy employed. For example, t_Φ can increase due to conflicts in accessing shared resources, while each t_{φ_i} includes possible overheads at the interface I.

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 objects or the target location. The partition strategy, by distributing individuals along the path, can reduce these costs. On the other hand it entails costs in terms of inefficiencies and delays when transferring objects between individuals. Which of the two strategies is better depends on the parameters of the environment and on the costs involved.

4 The proposed method

In this work we propose an adaptive method that allows a swarm of robots to tackle the problem depicted in Figure 1, and autonomously decide whether to partition a task Φ into the sequence of two sub-tasks or not. We study different cases in which a partitioning strategy can be more or less advantageous with respect to a non-partitioning one. The strategy observed at the level of the swarm is the result of the individual choices made by each robot, that depend on comparisons of the perceived costs of the two options. 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 overall task Φ without any decomposition into sub-tasks. After completing a task, be it either the overall task Φ or one of the two sub-tasks φ_i, a robot decides which strategy to employ next. In the proposed method, each robot has a probability P_p of employing task partitioning defined by the following sigmoid functions:

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

where S is a steepness factor, $\hat{t}_\Phi$ is an estimate of the time t_Φ required by the overall task when performed as a whole without partitioning. $\hat{t}_{\varphi_1}$ 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. The robots are never inactive, meaning that if a robot decides not to employ task partitioning then it works on the overall task (with a probability $P_{np} = 1 - P_p$).

Each time estimate is computed as a weighted average of the time it takes to perform each (sub-)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 was needed to perform the sub-task φ_i the last time. $\alpha \in (0, 1]$ 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 $\hat{t}_\Phi$.

It is important to notice that the costs of a strategy perceived by a robot are estimates of the real costs. In fact, each robot derives these costs by using the estimates $\hat{t}$. These depend not only on the intrinsic characteristics of the environment and the tasks, but also on the choices made by the other robots. Interference among robots also impacts on the time needed to perform a task, and depends on how many robots are performing the same 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 the delay experienced at the task interface I depends on how well the other sub-task is being performed.

In the example of foraging, previously mentioned in Section 3, each robot would keep an estimate of the time it takes to complete the overall foraging task, as well as the harvest and store sub-tasks. This information would be used by the robot to decide whether to employ task partitioning: the lower the time needed to complete the foraging task when partitioned into sub-tasks, the higher the probability of employing task partitioning.

As each robot has only estimates of the real costs, noise might lead to sub-optimal decisions. Furthermore, dynamic changes in the system can render preferable a strategy that previously was not. Therefore, the different strategies need to be sampled by the robots from time to time, in order for the system to be reactive to changes in the environmental conditions. Thus, a switching mechanism between tasks is also required. When a partition strategy is employed, a robot working on φ_1 might be waiting for a free store location at the interface; analogously a robot working on φ_2 might be waiting for input at the interface. When a robot is experiencing one of the two situations, it has a probability of giving up performing its sub-task and switching to the other. This probability is defined as:

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

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

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

with K and O being a steepness and an offset factor respectively and $\hat{t}_{\varphi_i}$, computed using Equation 2, being 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 tw_I is greater than $\hat{t}_{\varphi_i}$, the value of $\hat{t}_{\varphi_i}$ is updated using Equation 2. The probability function defined in Equation 3 ensures that a robot eventually gives up waiting at the interface while performing a sub-task.

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 on the harvest sub-task is poor, and 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 place 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, for example in swarm robotics (Labella et al., 2006) and collective robotics (Parker, 1998).

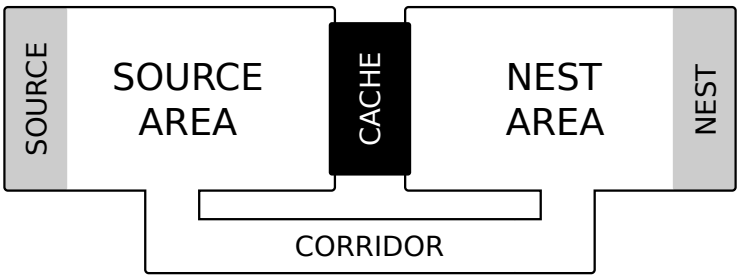

Figure 2: Conceptual representation of the environment used to study the partitioning problem on a foraging task. The environment is divided into two areas, separated by the cache. The source area contains the source and hosts the harvest sub-task $\varphi_{harvest}$: harvesting objects from the source and dropping them at the cache. The nest area contains the nest and hosts the store sub-task φ_{store}: picking up objects from the cache and store them in the nest. The corridor can be used by the robots to reach the source area from the nest area and the other way around. The robots cannot cross the cache.

5 Experimental framework

This section describes the experiments we employ to assess the validity of the method presented in Section 4. We test the method in a single nest, single source homogeneous foraging task (see Winfield, 2009, for a taxonomy). The overall task Φ_{forage} consists in harvesting objects from the source and storing them 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 foraging task is by separating the *harvest* and *store* sub-tasks ($\varphi_{harvest}$ and φ_{store}, respectively) and allowing material transfer between individuals working on the two sub-tasks. Figure 2 provides a conceptual representation of the partitioned foraging task. The environment is divided into two separate areas, referred to as *source area* and *nest area*. The source area is the part containing the source, the nest area is the part of the environment containing the nest. The two areas are separated by a temporary storage area that cannot be crossed by the robots and that is referred to as *cache* in the rest of the paper. $\varphi_{harvest}$ develops in the source area and consists in harvesting objects from the source and dropping them at the cache. φ_{store} develops in the nest area and consists in picking up objects from the cache and storing them at the nest. 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 Section 4 and represented in Figure 1, with Φ being the foraging task (Φ_{forage}). The two sub-tasks φ_1 and φ_2 correspond to the $\varphi_{harvest}$ and φ_{store} sub-tasks. The cache serves as the interface I between the two sub-tasks. The choice the robots face about the strategy to be employed translates in choosing whether to use the cache or the corridor: a partition strategy makes use of the cache, a non-partition strategy makes use of the corridor. In case a robot decides to employ a non-partition strategy, it performs the task Φ_{forage} by itself: an object is directly transported from the source to the nest. In case a robot

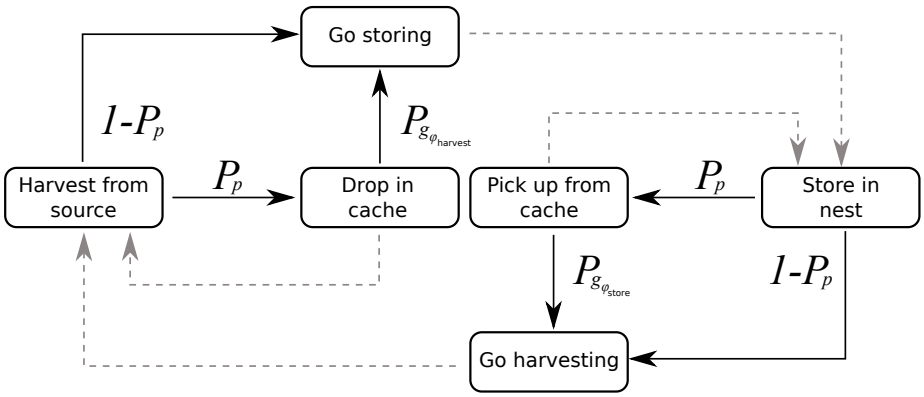

Figure 3: Finite state machine describing the behavior of each robot. The black continuous arcs marked with a label represent stochastic choices that a robot can make. P_p is the probability of employing task partitioning. $Pg_{\varphi_{store}}$ and $Pg_{\varphi_{harvest}}$ are the probabilities of giving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

decides to employ a partition strategy, it either works on $\varphi_{harvest}$ or on φ_{store}. The cost of the non-partition strategy is the time needed to travel along the corridor. The cost of the partition strategy is computed as the sum of the costs for using the cache on both its sides. For each side, the cost consists of the time needed to reach the cache plus a delay D the robots have to wait inside the cache each time they use it for dropping or picking up an object. The value of D can be used to regulate the relation of the costs of the two strategies and to model cases in which the cache usage imposes a delay. An example is loading and unloading shelves in a robotic warehouse, which are lengthy operations when complex manipulation abilities and dexterity are required from the robots. Each robot takes its own decision on the strategy to employ. The behavior of each individual is represented in the state diagram of Figure 3. The probabilities marked on the arcs are regulated by the mechanisms described in Section 4. No explicit communication is employed, none of the robots knows what the others are doing and has any notion of the swarm's performance. The strategy observed at the level of the swarm is the result of the individual's 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[1]. ARGoS is a discrete time, physics-based simulation environment that allows the simulation of objects at different levels of detail. For the work presented in this article, is has been sufficient to simulate the kinematics of the robots in two dimensions.

[1] `http://www.swarmanoid.org/`

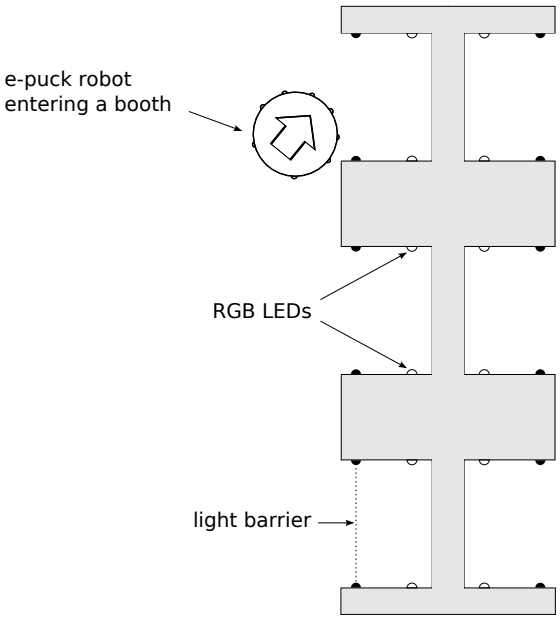

Figure 4: Representation of the an array of booths, composed of three booths on each of the two sides. Each booth can detect the presence of a robot through a light barrier, represented with the black semicircles and the dotted line in one of the booths. RGB LEDs, represented with the blank semicircles,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

The robots simulated in the experiments have a real counterpart: the e-puck[2] robot (Mondada et al., 2009). The e-puck is a small wheeled robot, developed at EPFL, Lausanne, 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 8 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

[2] `http://www.e-puck.org/`

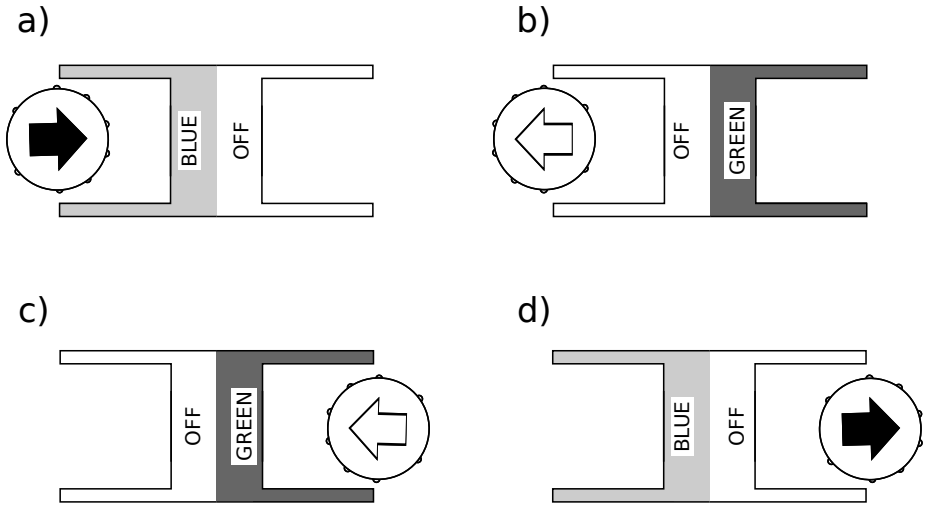

Figure 5: The behavior of a cache implemented with two paired booths. The source (not shown) is located at the left and the nest (not shown) is located at the right. The robots carrying an object are marked with a black arrow, and the robots not carrying objects with a white arrow. (a) Initially the 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 towards 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).

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's location. 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 with LEDs lit up in blue, we assume that the object being carried is dropped in that spot. 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. 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

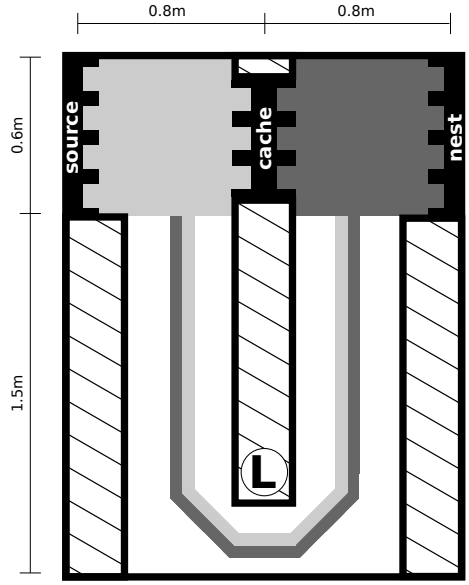

Figure 6: The environment used in the simulation experiments. Nest and source arrays are composed of four booths, the cache array is composed of three booths for each side. The color of the ground is used by the robots as a clue to recognize their location in the arena. The light source, marked with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

nest have their LEDs turned off (Figure 5a). In this configuration the cache is empty. Once a robot has dropped an object in the cache, by first entering and then 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 (Figure 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 first entering and then exiting the booth on the nest side (Figure 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 drop spot is available again (Figure 5d) [3].

5.3 Environment

The environment in which the robots have to perform the foraging task is shown in Figure 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

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

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

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 choose them. The goal of the second set of experiments is to assess the performance of the method. The goal of the third set 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 100,000 simulation steps of length 0.1 s each. This amounts to a simulated time of approximately two hours and forty-five minutes. The swarm is composed of fourteen simulated e-pucks except in the scalability experiments, in which we study the performance of swarms of various size. At the beginning of each run the robots are randomly positioned in the nest area. The speed of the robots is set to $3.5\,\mathrm{cm/s}$, 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 towards a booth. To avoid bias in the robots' behavior, the estimates of the times $t_{\varphi_{harvest}}$, $t_{\varphi_{store}}$, and $t_{\Phi_{forage}}$ are randomly initialized. The initial values of the weighted average times $\hat{t}_{\varphi_{harvest}}$ and $\hat{t}_{\varphi_{store}}$ have been uniformly sampled in $[500, 1000]$ and $\hat{t}_{\Phi_{forage}}$ in $[1000, 2000]$. For each experimental condition we run 50 randomly seeded simulations.

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 have better insights 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 the analysis and the possible values they can assume. For each combination of the parameters values, we run 15 randomly seeded simulations, with a swarm composed of fourteen robots. What follows is a summary of the results of ANOVA, for the complete results we refer the reader to Pini et al. (2011).

We perform four analyses, one including D as factor, and other three fixing D to a value and using the remaining parameters as factors. The four analyses agree on using high values for the parameter α, independently of the value of the delay D. This is due to the fact that for lower values of α, the swarm is not able to quickly estimate the costs of each strategy, and this leads to poor performance. The effect of the steepness S depends on the value of D: the higher the cache delay D, the lower should be S, and vice-versa. Therefore the value of S should be chosen depending on the specific

Table 1: Parameter space used for the analysis of variance. The values define a grid of points in the parameter space, ANOVA performs then a regression using these points.

Parameter	**Values**
Cache delay D (seconds)	0, 100, 200
α	0.2, 0.6, 1.0
Steepness S	-1, -3, -5
Steepness K	-0.5, -1.5, -2.5
Offset O	-1, -5, -9

environment. If the value of D is known, and it does not vary in time, S can be selected accordingly. On the other hand, if the value of the cache delay is unknown or can vary significantly over time, then S should be set to an intermediate value. An alternative would be to adaptively select the value of S on the basis of a measure of the value of D. 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 values for these parameters. To address this issue, ANOVA needs more data and higher order interactions should be studied in order to have deeper insights concerning how to select the two parameters values.

Please notice 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. The results of ANOVA have been used to guide us in a trial and error selection of the method's parameters, to use in our specific experimental setup. The parameters settings are the following: the two steepness factors are set to $S = -2.5$ and $K = -0.6$, the offset O is set to -5, α is set to 0.8. 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 one of two reference methods, referred to as *never-partition* and *always-partition* methods. The never-partition method is always making use of the corridor ($P_p = 0$), while the always-partition method is always making use of the cache ($P_p = 1$) without giving up ($Pg_{\varphi_{store}} = Pg_{\varphi_{harvest}} = 0$). In case of the always-partition method, the swarm is manually divided into two groups of the same size. One group is positioned in the nest area and its members work exclusively on the store sub-task. The other group is positioned in the source area and its members work exclusively on the harvest sub-task. We tested 9 different settings, each corresponding to a different value of the cache delay D. The delay D varies from a minimum of 0 s to a maximum of 200 s, by steps of 25 s (simulated time). The value 200 s was chosen to be considerably higher than the average time needed by the robots to travel along the corridor. Which of the two strategies is more advantageous depends on the value of D. The graph in Figure 7 shows the results of the second set of experiments. The graph plots the average performance of the three

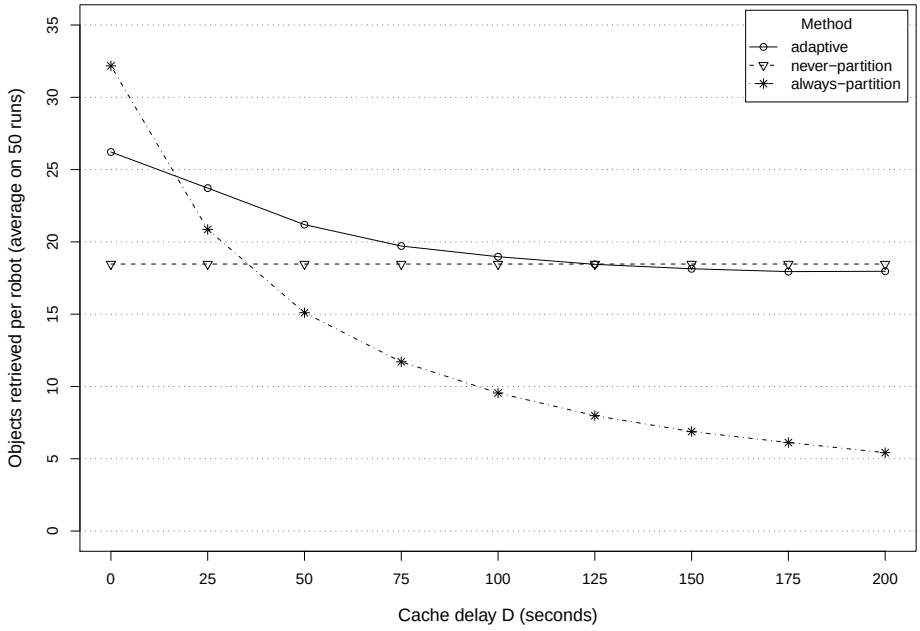

Figure 7: Average amount of objects retrieved per robot at the end of the experiment for different values of the cache delay D. 95% confidence intervals have been omitted for clarity, as they are very close to the mean.

methods for different values of the cache delay D. As expected, each reference method performs well only for a subset of values of D. The always-partition method performs well when the cache delay 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 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 D. The adaptive method is able to perform well in both cases, showing good performance on the whole spectrum of the parameter D.

Figure 8 provides a summary of the strategy employed by the swarm when using the adaptive method, for the different values of cache delay D. Each bar reports the percentage of selection of the two strategies, using the probability function of Equation 1. As mentioned, choices are made by the robots after storing an object in the nest and after harvesting an object from the source. The graph confirms that the robots select the corridor with a higher frequency for increasing values of the cache delay D. For small values of D, the preferred choice is the cache.

Figure 9 reports the average time needed by the robots to use the corridor and the cache employing the three different methods, when the cache delay is 25 s (left), 50 s (center), or 75 s (right). The time for going through the corridor ($t_{\Phi_{forage}}$) is measured from the moment a robot decides to use the corridor till the moment it exits on the other side. The time needed for dropping an object ($t_{\varphi_{harvest}}$) is measured from the moment

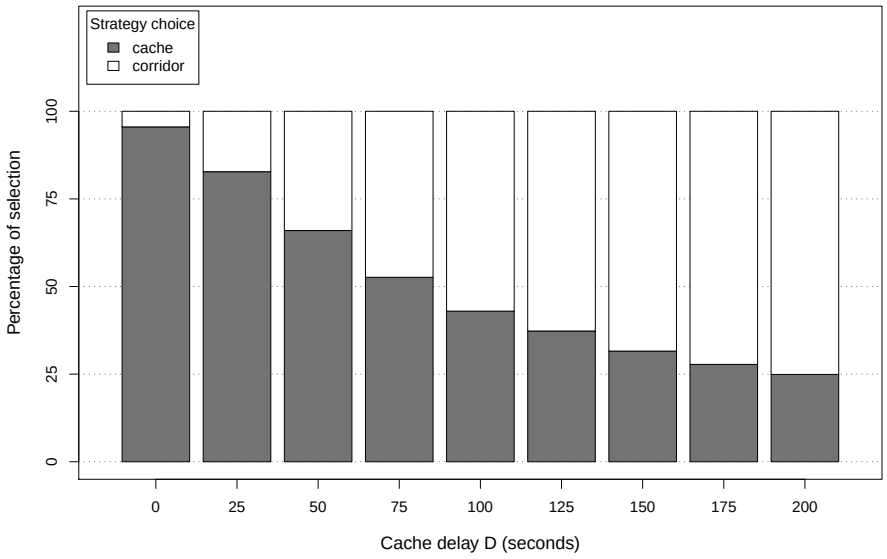

Figure 8: Strategy chosen by the robots, using the adaptive method, for different values of the cache delay D. Each bar reports, for each value of D, the percentage of selection of the partition strategy (gray) and non-partition strategy (white). The values are averages computed over 50 experimental runs.

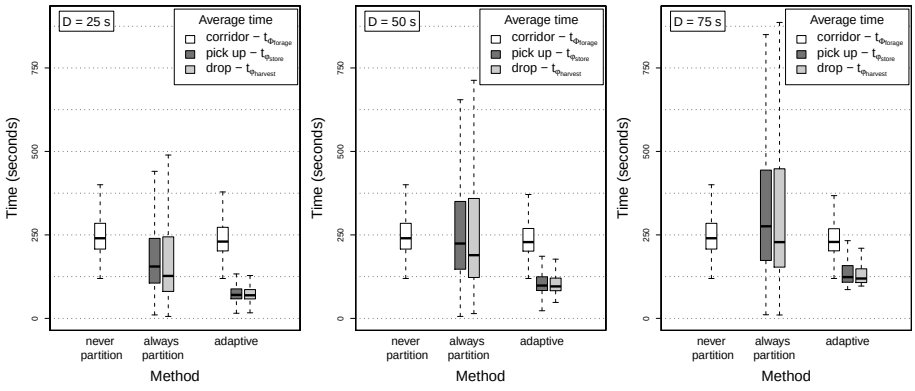

Figure 9: Average time needed for using the corridor and the cache for the three different methods. The graphs report the data for $D = 25$ s (left), $D = 50$ s (center), and $D = 75$ s (right).

a robot harvests an object from the source till the moment it leaves the cache after depositing the object. The time needed for picking an object up ($t_{\varphi_{store}}$) is measured

from the moment a robot stores an object in the nest till the moment it leaves the cache after picking an object up. Notice that the average measured pick up and drop times include the delay D.

When $D = 25\,\text{s}$ the always-partition method performs better than the never-partition method (see Figure 7). Figure 9 (left) shows that the average time needed for using the cache is lower than the one needed for crossing the corridor. Therefore a strategy that always partitions the task is preferable to one that never does. For $D = 50\,\text{s}$ and $D = 75\,\text{s}$, the never-partition method performs better than the always-partition one. The value of D is still small when compared to the time it takes to travel through the corridor, but a robot that wants to use the cache needs to wait longer when other robots are using it. This effect can be seen in the high variances of the pick up and drop times, when using the always-partition method. Due to this reason, the resulting pick up and drop times are often higher than the time needed to travel along the corridor (Figure 9, center and right).

The adaptive method performs better than the two reference methods for the three values of D considered here. The average times needed to access the cache on the two sides when employing the adaptive method are lower than the ones of the always-partition method. This is probably due to the fact that the robots eventually give up using the cache when their waiting time becomes too high. The fact that some robot choose the corridor also helps reducing the average cache usage time, as there is less concurrent access to the cache.

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 delay D at the cache when the experiment reaches half of its duration. We consider two cases: one in which D changes from 0 s to 200 s and another in which it changes from 200 s to 0 s. In the first case it is expected that the corridor is selected more frequently by the swarm in the initial phase and, once D changes to zero, the selection 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 Figure 10 (top) refers to the first case described (decrease of D); the graph of Figure 10 (bottom) refers to the second case (increase of D). In the graphs of Figure 10 the total time frame of the experiment has been divided into windows of 15 minutes. Each point of the plot reports, for each strategy, the percentage of selection in the 15 minutes time window preceding the value indicated on the X axis. In both cases, at half the experiment time, one of the two strategies is selected 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 both cases there is a swap of the strategy being selected the most, as the change in the value of D makes it more advantageous to use the other strategy after the experiment has reached halftime.

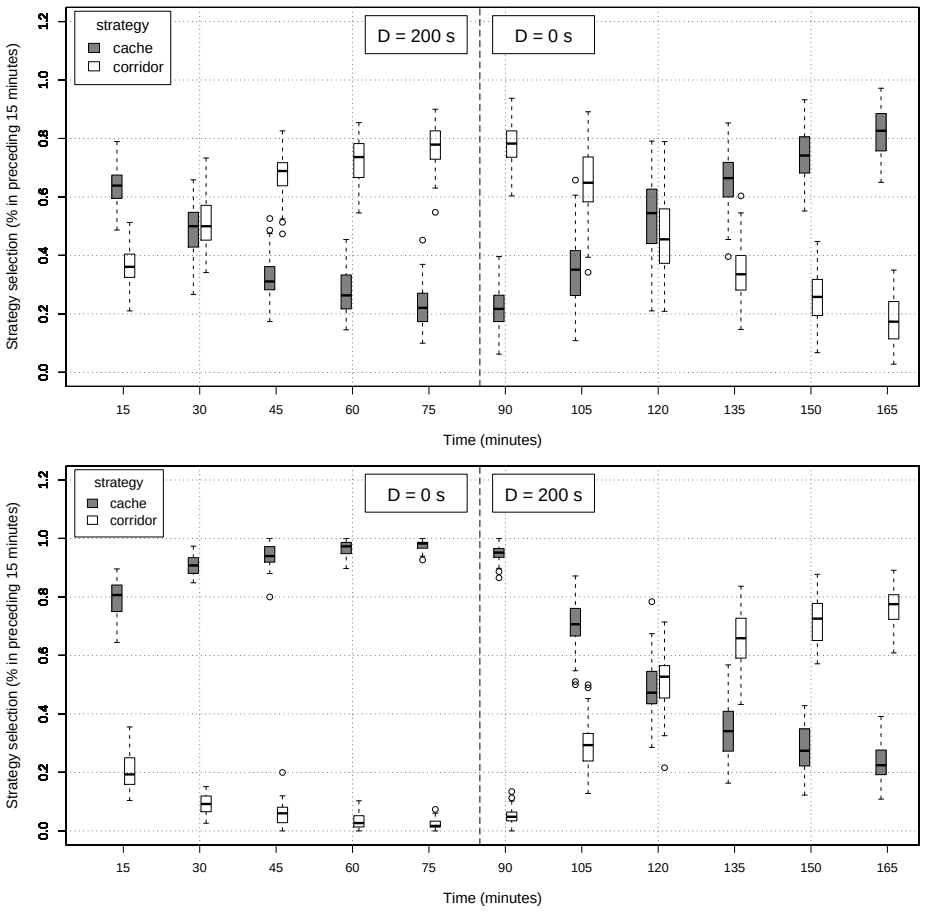

Figure 10: Strategy selection in time when the delay at the cache D is varied from 200 s to 0 s (top) and from 0 s to 200 s (bottom). The vertical dashed line marks the instant in which D is changed (time $t = 83$ minutes). The total length of the experiment is divided into windows of 15 minutes, each box reports the percentage of selection of the two strategies in the time window preceding the value reported on the X axis.

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 $D = 25$ s and $D = 50$ s. We vary the swarm size from a minimum of 6 to a maximum of 30 robots, by steps of 4. 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 $D = 2.5\,\mathrm{m}^2$ and the radius of an e-puck is 70 mm, the area coverage varies from a minimum of 3.69%, when the swarm is composed of 6 robots, to a maximum

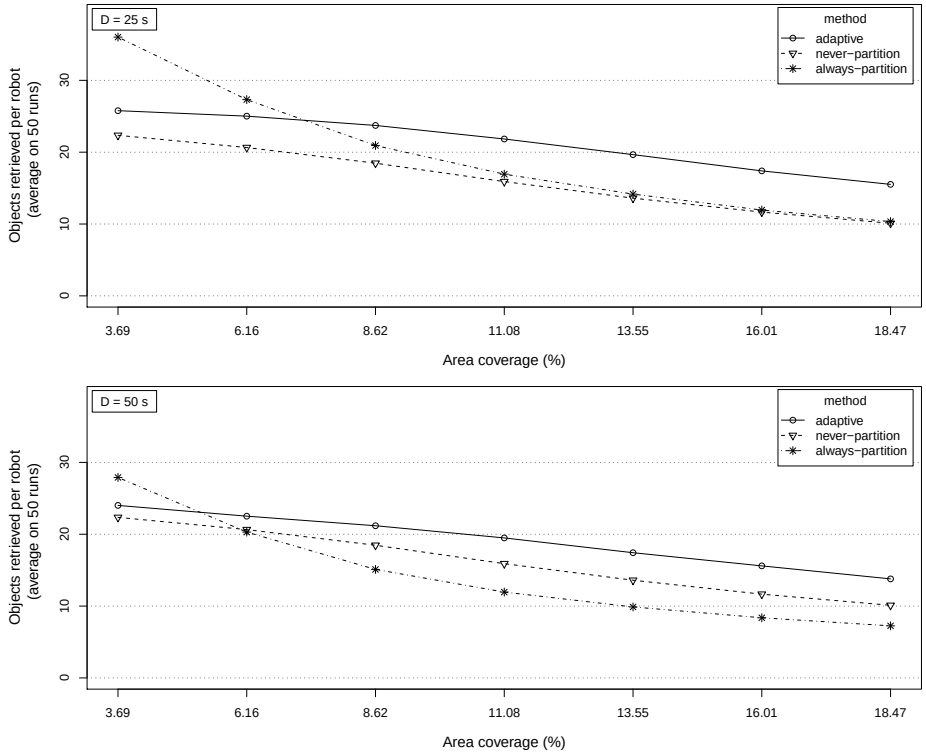

Figure 11: Results of the scalability experiment for $D = 25\,\text{s}$ (top) and $D = 50\,\text{s}$ (bottom). The plots report, for each of the three methods, the average individual performance for different area coverages.

of 18.47%, when the swarm is composed of 30 robots.[4] Figure 11 reports the results of the scalability experiments for $D = 25\,\text{s}$ (top) and $D = 50\,\text{s}$ (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 the always-partition method is the one that is affected the most by overcrowding, with steep drops in performance for increasing robots densities. The never-partition and the adaptive methods scale well for the different densities, but the performance of the adaptive method is always superior to the one of the never-partition method. The reason for which the adaptive method performs better than the two reference methods for a wide range of robots density is that the adaptive method can balance the number of robots trying to use the cache and the corridor. By doing so the robots can exploit the low cache cost without incurring in overcrowding problems, as some robots go through the corridor from time to time.

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

7 Conclusions

Our research 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 studied the environmental conditions under which it is, or it is not, advantageous to use a partition strategy. We considered environmental settings that differ in terms of the cost ratio of these two strategies. We proposed an adaptive method that allows a swarm of robots to decide whether to employ task partitioning 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 a foraging task 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 here presented is a first step towards a more general method for tackling a general task partitioning problem, in which the sub-tasks are more than two. 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 here presented 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, but 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 concerns adding 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 we propose 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, founded 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 here presented.

References

Aho, A. (1983). *Data Structures and Algorithms*. Addison-Wesley, Boston, MA.

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

Anderson, C., Franks, N. R., and McShea, D. W. (2001). The complexity and hierarchical structure of tasks in insect societies. *Animal Behaviour*, 62(4):643–651.

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

Anderson, C. and 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.

Beni, G. (2005). From swarm intelligence to swarm robotics. In Şahin, E. and Spears, W. M., editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 1–9. Springer, Berlin, Germany.

Brutschy, A., Pini, G., Baiboun, N., Decugnière, A., and 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. and Reid, N. (2000). *The Theory of the Design of Experiments*. Chapman and Hall/CRC, London.

Fontan, M. S. and Matarić, M. J. (1996). A study of territoriality: The role of critical mass in adaptive task division. In Maes, P., Matarić, M. J., Meyer, J.-A., Pollack, J., and Wilson, S., editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference of Simulation of Adaptive Behavior*, pages 553–561. MIT Press, Cambridge, MA.

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

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

Hart, A. G. and 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. and 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. and 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.

Hubbell, S. P., Johnson, L. K., Stanislav, E., Wilson, B., and 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 Kikuchi, T., Azuma, N., and Higashi, S., editors, *Proceedings of the 14th Congress of the IUSSI*, pages 81–130. Hokkaido University Press, Sapporo, Japan.

Labella, T. H., Dorigo, M., and 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. and Vaughan, R. (2008). Adaptive multi-robot bucket brigade foraging. In Bullock, S., Noble, J., Watson, R., and Bedau, M. A., editors, *Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 337–342. MIT Press, Cambridge, MA.

Lopes, J. F., Forti, L. C., Camargo, R. S., Matos, C. A. O., and 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., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In Gonçalves, P. J. S., Torres, P. J. D., and Alves, C. M. O., editors, *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65. IPCB: Instituto Politècnico de Castelo Branco, Castelo Branco, Portugal.

Østergaard, E. H., Sukhatme, G. S., and 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 Fifth International Conference on Autonomous Agents*, pages 29–30. ACM Press, New York.

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

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., and Dorigo, M. (2011). ARGoS: a Pluggable, Multi-Physics Engine Simulator for Heterogeneous Swarm Robotics. Technical Report TR/IRIDIA/2011-026, IRIDIA, ULB.

Pini, G., Brutschy, A., Birattari, M., and Dorigo, M. (2009). Interference reduction through task partitioning in a robotic swarm. In Filipe, J., Andrade-Cetto, J., and Ferrier, J. L., editors, *Sixth International Conference on Informatics in Control, Automation and Robotics – ICINCO 2009*, pages 52–59. INSTICC Press, Setùbal, Portugal.

Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., and Birattari, M. (2011). Task partitioning in swarms of robots: An adaptive method for strategy selection – Online supplementary material. `http://iridia/supp/IridiaSupp2011-003/`.

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

Reyes, J. L. and 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 Şahin, E. and Spears, W. M., editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, Berlin, Germany.

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. and 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)*, pages 2717–2723, Pitscataway, NJ. IEEE Press.

Theraulaz, G., Bonabeau, E., Solé, R. V., Schatz, B., and 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 Asama, H., Kurokawa, H., Ota, J., and Sekiyama, K., editors, *Distributed Autonomous Robotic Systems 8*, pages 185–192. Springer, Berlin, Germany.