Task partitioning in a robot swarm: a study on the effect of communication

**Giovanni Pini · Matteo Gagliolo · Arne Brutschy ·
Marco Dorigo · Mauro Birattari**

Abstract Task partitioning consists in dividing a task into sub-tasks that can be tackled separately. Partitioning a task might have both positive and negative effects: On the one hand, partitioning might reduce physical interference between workers, enhance exploitation of specialization, and increase efficiency. On the other hand, partitioning may introduce overheads due to coordination requirements. As a result, whether partitioning is advantageous or not has to be evaluated on a case-by-case basis. In this paper we consider the case in which a swarm of robots must decide whether to complete a given task as an unpartitioned task, or utilize task partitioning and tackle it as a sequence of two sub-tasks. We show that the problem of selecting between the two options can be formulated as a multi-armed bandit problem and tackled with algorithms that have been proposed in the reinforcement learning literature. Additionally, we study the implications of using explicit communication between the robots to tackle the studied task partitioning problem. We consider a foraging scenario as a testbed and we perform simulation-based experiments to evaluate the behavior of the system. The results confirm that existing multi-armed bandit algorithms can be employed in the context of task partitioning. The use of communication can result in better performance, but in may also hinder the flexibility of the system.

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

G. Pini (✉) · A. Brutschy · M. Dorigo · M. Birattari
IRIDIA, Université Libre de Bruxelles, Bruxelles, 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. Gagliolo
GERME and MLG, Université Libre de Bruxelles, Bruxelles, Belgium
e-mail: mgagliol@ulb.ac.be

1 Introduction

In this work, we present a study on task partitioning in a swarm robotics context. The term *task partitioning* comes from biology and it refers to situations in which a given task is divided into two or more sub-tasks that can be tackled separately (Jeanne 1986). The main body of research on the topic has been carried out in the field of entomology. Many studies have been published that report examples of task partitioning for the organization of work in colonies of ants, bees, and wasps (Ratnieks and Anderson 1999). Task partitioning has been mostly observed in activities that involve the transportation of material such as foraging (Fowler and Robinson 1979), nest excavation (Anderson and Ratnieks 2000), and waste removal (Hart and Ratnieks 2001). However, examples of task partitioning have been observed in other activities, such as hunting (Theraulaz et al. 2002).

The fact that several instances of task partitioning can be observed in different species of animals indicates that partitioning a task into sub-tasks is an advantageous way of organizing work. Indeed the benefits that insects draw from task partitioning are many. In certain situations, task partitioning enhances the exploitation of specialization, either morphological or behavioral. For example, in the seed-harvesting ant *Messor bouvieri* small workers transfer harvested seeds to larger workers, that transport them to the nest (Arnan et al. 2011). Task partitioning allows each category of workers to perform the activity for which they are best suited: smaller workers are better (i.e., faster) at finding seeds, while larger workers are better at transporting them.

Task partitioning can also increase the efficiency in performing a task. For example, Fowler and Robinson (1979) described the foraging strategy employed by the leaf-cutting ant *Atta sexdens*: some workers climb a tree and cut leaves that are dropped to the ground. There, other ants collect the leaves and transport them to the nest. In this case, partitioning the foraging task reduces the energy requirements for the swarm since only a few ants need to climb the tree and they do it only once.

Finally, task partitioning allows for the separation of individuals working on different sub-tasks. In some situations, the separation of workers may be beneficial. Think for example of the *Atta cephalotes* ant: the nest of this ant has an internal garbage heap, which is managed by garbage heap workers confined in the heap location. Waste management is carried out using task partitioning: garbage is delivered to the entrance of the heap, where it is collected by garbage heap workers and stored in the heap. This behavior allows the ants to isolate garbage heap workers, which are potentially contaminated by pathogens, from the rest of the colony (Hart and Ratnieks 2001).

Task partitioning requires the individuals to coordinate their work, and therefore entails overhead costs. For example, honeybee foragers transfer nectar loads to receivers located near the nest entrance. In order to transfer nectar, a forager and a receiver must search for each other, introducing a delay in foraging (Anderson and Ratnieks 1999). Task partitioning can also entail inefficiency in terms of material losses. Returning to the example of the foraging activity of the *Atta sexdens* ant, mentioned previously, Hubbell et al. (1980) pointed out that many of the leaves dropped to the ground are lost and therefore they cannot be utilized by the swarm.

In this paper, we present a research work in the context of swarm robotics, a branch of robotics that studies the implementation and control of large groups of autonomous robots, often by using solutions inspired by nature, in particular by social insect colonies. Robot swarms can obtain the same benefit as social insects from task partitioning: better exploitation of specialization, increased efficiency, and physical separation of the robots. Analogously to social insects, task partitioning also entails costs due to overheads, which can

hinder these benefits. Task partitioning should therefore be used only when the benefits overcome the costs.

We concentrate here on what we call the *task partitioning problem*: selecting between partitioning a given task into sub-tasks or performing it as a single piece of work. We study the task partitioning problem in a foraging scenario. In our previous research, we proposed distributed methods that enable a robot swarm to tackle the task partitioning problem autonomously (Frison et al. 2010; Pini et al. 2011). Recently, we have shown that the task partitioning problem can be formulated as a multi-armed bandit problem and tackled with existing algorithms proposed in the reinforcement learning literature (Pini et al. 2012a). This article further extends our previous work by introducing a *social* component: the robots explicitly communicate with each other. We perform simulation-based experiments to study the effect of communication. The results suggest that communication affects the system in a different way, depending on the algorithm being employed by the robots to take decisions. In some cases, communication enhances the performance of the swarm while in others the increased inflow of information can overwhelm the decision process of the robots and renders the behavior of the swarm inflexible with respect to changing environmental conditions.

This article is organized as follows. In Sect. 2, we review related work. In Sect. 3, we present the task partitioning problem studied in this paper and illustrate the approach we use to tackle it. In Sect. 4, we describe the tools that we used to carry out the experiments. In Sect. 5, we describe the simulation-based experiments that we carried out to study the system and comment on the results obtained. In Sect. 6, we summarize the contributions of this work and propose directions for further research.

2 Related work

As mentioned in the introduction, research on the topic of task partitioning has been carried out mainly by entomologists. For a review of task partitioning in social insects, we refer the reader to the work of Ratnieks and Anderson (1999). In Sect. 2.1, we focus instead on the related work on task partitioning in swarm robotics.

The research presented in this article utilizes concepts and ideas that come from reinforcement learning. The literature on the topic is vast and its review is beyond the scope of this work. In Sect. 2.2, we mention the related research work that is the most relevant to our study.

2.1 Task partitioning in swarm robotics

In the domain of swarm robotics, task partitioning has been used mainly in foraging scenarios as a mean for reducing physical interference between robots. Physical interference results from the fact that robots share the same space at the same time. Interference has a strong negative impact on the performance and the scalability of a robotics system (Lerman and Galstyan 2002). Task partitioning can mitigate interference in foraging through a physical separation of robots.

The first authors that used task partitioning to reduce physical interference in robotics were Drogoul and Ferber (1992). They studied a foraging scenario and reported the formation of "traffic jams" in the environment. The authors showed that allowing the robots to pass objects one to another leads to the formation of a chain, along which objects are passed till they reach the nest. This reduces interference and increases the foraging performance of the swarm.

Østergaard et al. (2001) studied the benefits of task partitioning in a setup in which the robots perform foraging in a maze. The authors concluded that task partitioning is beneficial in cluttered environments, in which the width of the corridors is such that it is hard for two robots traveling in opposite directions to pass at the same time.

Fontan and Matarić (1996) divided the environment in which the robots perform foraging into non-overlapping working areas, each assigned to one robot. Each robot transports pucks found in its working area to an adjacent working area, in the direction of the nest. A puck eventually reaches the nest by crossing several working areas. As the range in which each robot operates is limited, physical interference is diminished and foraging performance is enhanced. Goldberg and Matarić (2002) used the same setting to study the design of robust behavior-based controllers. The authors observed that a solution based on task partitioning reduces interference.

Pini et al. (2009) also divided the environment in working areas and showed that task partitioning reduces competition for accessing a shared resource (the nest where the foraged objects are stored). In the work of Pini et al., differently from the one of Fontan and Matarić (1996), the working areas are not exclusive, nor a priori assigned: each robot selects its working area autonomously.

Shell and Matarić (2006) introduced a further novelty with respect to the work of Fontan and Matarić (1996). In the work of Fontan and Matarić (1996), the position of working areas are given with respect to a global coordinate system and are fixed in time; on the other hand, in the work of Shell and Matarić (2006), the position of the working areas are given in the local coordinate system of the associated robot and drift in time. Indeed, given that robots estimate the position of working areas using odometry, working areas drift in time due to estimation errors. Each robot transports the pucks that it finds in its working area towards the nest, without leaving its working area. The robot drops the pucks at the boundary of its working area, where they are eventually collected by another robot. The authors showed that the higher the density of robots, the smaller the optimal size of the working area.

Lein and Vaughan (2008, 2009) further extended the work of Shell and Matarić (2006). The first extension consists in the implementation of a simple algorithm that each robot employs to dynamically regulate the size of its working area, on the basis of the perceived interference (Lein and Vaughan 2008). The second extension is a mechanism that the robots use to relocate their working areas towards zones in which there is a high puck density (Lein and Vaughan 2009).

Pini et al. (2012b) studied the case in which a robot swarm forages objects from an initially unknown location in the environment. Each robot uses odometry to estimate its position relatively to the location where an object was found. The authors showed that task partitioning reduces the negative impact of odometry errors and improves the foraging performance. Additionally the authors used the implemented system to study the costs deriving from the direct transfer of foraged objects from robot to robot. Together with the works of Fontan and Matarić (1996) and Goldberg and Matarić (2002), the work of Pini et al. (2012b) is the only work on the topic of task partitioning that includes experiments performed with real robots.

The same experimental setup, proposed in the work of Pini et al. (2012b), was employed in the work of Pini et al. (2013). There, the authors proposed a methodology that allows a robot swarm to autonomously partition tasks into sub-tasks. Each robot autonomously decides the amount of work that it should contribute with the sub-task that it performs. The decision is made on the basis of cost estimates that the robot computes while performing the sub-tasks.

In this article, the task partitioning problem that we study consists in selecting whether to employ task partitioning or not. In a previous research work, we proposed a simple method to

let a swarm of robots tackle autonomously the task partitioning problem (Frison et al. 2010). In follow up work, we extended the method so that the costs of performing the given task and each of the sub-tasks are taken into account by the robots when taking decisions (Pini et al. 2011). The research presented in this article further extends a previous study, in which we have shown that the task partitioning problem can be seen as a multi-armed bandit problem and can be tackled with existing algorithms proposed in the reinforcement learning literature (Pini et al. 2012a). We extend the work by adding a social element to the studied system: the robots are allowed to exchange information with each other and they use the received information to integrate their knowledge of the environment.

2.2 Multi-agent reinforcement learning

In the artificial intelligence community, the problem of multi-agent learning (Weiss 1999; Shoham et al. 2007) is receiving a growing attention. In this context, also simple forms of *social learning* are being considered, where agents can communicate with each other, in order to improve the overall performance of the group. Most existing multi-agent learning approaches are based on *reinforcement learning* (Sutton and Barto 1998; Busoniu et al. 2008): the agents are allowed to interact with each other and with the environment, choosing actions sequentially according to some policy, which is learned online. The only feedback available to guide learning is a (possibly delayed) *reward* signal. Only limited work in multi-agent learning deals with some form of social learning (see Panait and Luke 2005, for a review). In the work of Whitehead (1991), agents interact only with a common environment, and can exchange *learning episodes*, in the form of (*state*, *action*, *next state*) triplets. Tan (1993) allowed for interactions between agents, as well as for some forms of communication, such as sharing observed reward values and learned policies. In the work of Nunes and Oliveira (2008), each agent broadcasts the reward obtained during the last learning episode, and poorly performing agents can query the best ones for advice. Reinforcement learning in a stateless environment corresponds to what is called the *multi-armed bandit* problem (Sutton and Barto 1998; Cesa-Bianchi and Lugosi 2006): in a sequence of trials, the player chooses one of the arms of a multi-armed bandit and receives the corresponding reward. The aim of the agent is to minimize its *regret* of not having played the remaining arms, whose reward is not observed. Schlag (1998) studied social learning in a group of agents, all playing against the same multi-armed bandit. They focused on social learning strategies in which each agent observes the last pulled arm, and the corresponding reward, for itself and another individual, randomly selected.

3 Problem and methodology description

In this section, we present the task partitioning problem studied in this work and illustrate the approach we employ to tackle this problem. In Sect. 3.1, we describe in detail the task partitioning problem and introduce the terminology used in the rest of the paper. In Sect. 3.2, we provide an abstract description of the foraging scenario studied in this work and frame the task partitioning problem in the context of foraging. In Sect. 3.3, we illustrate the approach we use to tackle the task partitioning problem and present algorithms that we compare in our study.
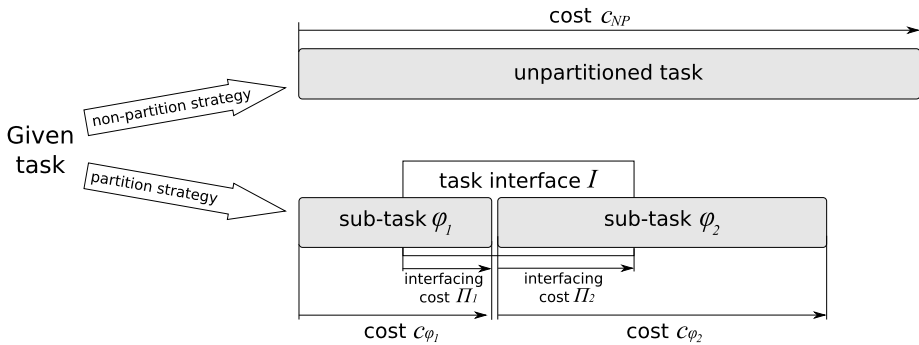
Fig. 1 Representation of the task partitioning problem considered in this study. A given task can be either completed as a whole, unpartitioned task (*top*) or partitioned into a sequence of two sub-tasks φ_1 and φ_2 connected by an interface I (*bottom*). Depending on the chosen option, the cost for performing the given task is different. If the given task is performed as an unpartitioned task, it entails a cost c_{NP} for its completion. If task partitioning is used, the cost for the completion of the given task is the sum of the costs c_{φ_1} and c_{φ_2} required for completing the sub-tasks. These costs include possible interfacing costs Π_1 and Π_2, required to access I. In general, c_{NP} differs from the sum of c_{φ_1} and c_{φ_2}, rendering task partitioning advantageous or not

3.1 The task partitioning problem

In this work, we tackle the problem represented in Fig. 1. A given task can be partitioned into the sequence of two sub-tasks φ_1 and φ_2, that interface with each other by means of an interface I. The interface allows to store the output of the sub-task φ_1, that can be subsequently used as input for φ_2. In this context, we tackle what we refer to as the *task partitioning problem*: selecting whether to perform the given task as overall, unpartitioned task (Fig. 1, top), or to employ task partitioning and execute the two sub-tasks separately (Fig. 1, bottom). Notice that we do not tackle the problem of defining what the sub-tasks φ_1 and φ_2 are. We consider the sub-tasks as defined a-priori and we tackle the problem of selecting whether to execute the given task with or without partitioning it into sub-tasks. For an example of a swarm that autonomously defines sub-tasks of a given task, we refer the reader to the study of Pini et al. (2013).

An example of the situation depicted in Fig. 1 is the leaf foraging activity of the *Atta* ant, described in the work of Fowler and Robinson (1979). In that case, the task consists in harvesting a leaf from a tree and transporting it to the nest. Ants partition this task into two sub-tasks. A group of ants cut leaves from the tree and drop them to the ground (sub-task φ_1). There, the leaves are collected by a second group of ants and transported to the nest (sub-task φ_2). The ground underneath the tree is the interface I, where the output of the first sub-task (a leaf that has been cut) is temporarily stored before it is used as the input of the sub-task that follows. In principle, the ants could perform the same task without partitioning it into sub-tasks by repeatedly climbing the tree to collect leaves.

Returning to the problem represented in Fig. 1, we use the term *strategy* to indicate the way the given task is performed. We indicate with *partition strategy* the case in which the given task is performed as two separate sub-tasks and with *non-partition strategy* the case in which the given task is performed as one unpartitioned task. Each strategy entails a *cost*; the way costs are measured depends on the specific context; examples are: energy, time, and resources needed for executing tasks and sub-tasks. Notice that, in general, the costs of the two strategies are different (i.e., $c_{NP} \neq c_{\varphi_1} + c_{\varphi_2}$). The cost c_{NP} of the non-partition strategy

may include a component deriving from competition in accessing a shared resource, such as space (Pini et al. 2009). On the one hand, the partition strategy can be used to reduce costs of this nature, by distributing individuals across sub-tasks. On the other hand the partition strategy usually entails overhead costs such as interfacing costs (Π_1 and Π_2). Which of the two strategies is the most advantageous depends on the context and on the nature of the costs involved.

In the case of leaf foraging in the *Atta* ant, the ants perform foraging using the partition strategy. The result, compared to the use of the non-partition strategy, is that the total energy spent by the swarm is lower due to the fact that the ants do not need to repeatedly climb the tree trunk. However, the partition strategy is costly in terms of foraging efficiency: many of the leaves that are dropped to the ground by an ant are not found by any other ant (Fowler and Robinson 1979).

3.2 The foraging scenario

Foraging consists in the repetition of an object transportation task: collecting an object from the environment and delivering it to a predefined location. Figure 2 represents our experimental environment, in which the robots forage for objects. We consider the case in which the objects are located in a unique place in the environment, referred to as the *source*. The objects that are collected from the source must be delivered by the robots to the *nest*. We assume that the source never depletes and that the robots know a priori the locations of the source and the nest. Source and nest are located in two separate areas, which are connected by a path referred to as the *corridor*. The corridor allows the robots to reach the source from the nest and the other way around. The two areas are separated by the *cache*, which cannot be crossed by the robots, but that can be used to transfer objects from one area to the other.

The foraging scenario illustrated in Fig. 2 is an instance of the task partitioning problem presented in Fig. 1. Performing the transportation task as one unpartitioned task corresponds to utilizing the corridor: a robot directly HARVESTS an object from the source and STORES it in the nest. The cache corresponds to the interface *I* and it allows the robots to partition the transportation task into a sequence of two sub-tasks: a robot DROPS in the cache an object
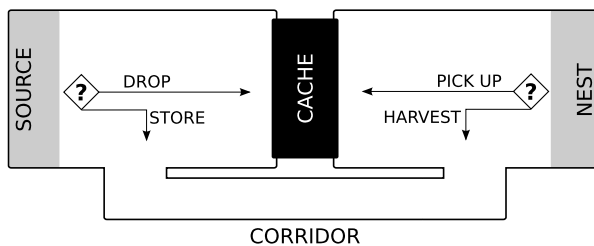

Fig. 2 Representation of the environment in which foraging is performed. The robots collect objects from the source, and deposit them in the nest. The areas containing the source and the nest are connected by a corridor, which can be utilized by the robots to reach one area from the other. The corridor allows the robots to perform transportation without employing task partitioning: a robot directly HARVESTS an object from the source and STORES it in the nest. The cache can be used to transfer objects from an area to the other and it allows the robots to partition transportation: an object is first DROPPED in the cache by a robot and subsequently it is PICKED UP by another robot on the other side. The robots decide whether to use the cache or the corridor in two cases, represented as question marks: after taking an object form the source (*left-hand side*) and after depositing an object in the nest (*right-hand side*)

that it collected from the source; the same object is subsequently PICKED UP by a second robot on the other side of the cache and delivered to the nest. Dropping an object in the cache corresponds to the sub-task labeled with φ_1 in Fig. 1; picking up an object from the cache to the sub-task labeled with φ_2. In the foraging context, we measure costs as time for completing the transportation task. The cost c_{NP} of the non-partitioning strategy depends on the length of the corridor, which defines the time required to navigate through it. The overhead cost of the partitioning strategy depends on the time required to use the cache on both its sides. The use of the cache imposes an interfacing time Π, which is required to drop or pick up an object from the cache: depending on the value of Π and the length of the corridor, task partitioning can be more or less advantageous.

The robots decide whether to use the cache or the corridor (i.e., whether to partition object transportation or not) in two occasions, represented with a question mark in Fig. 2. After taking an object from the source, a robot must decide whether to traverse the corridor and directly store the object in the nest or to use the cache to drop the object. After delivering an object to the nest, a robot must decide whether to traverse the corridor and harvest the following one from the source or to reach the cache to pick up an object there. As each robot takes decisions autonomously, there is not, in general, a unique decision made by all the robots: some robots may employ the cache while others may employ the corridor.

The robots can abandon the decision to use the cache to pick up or drop an object (details are given in Sect. 3.3.2). This protects the swarm against deadlocks, which may occur if all the robots try to pick up objects at the same time, and the cache is empty or, dually, if all the robots try to drop objects at the same time and the cache is full.

3.3 The approach

In this section we describe the approach we employ for tackling the task partitioning problem presented in Sect. 3.1. In our approach, each robot associates a cost to the possible actions and uses these costs to select which action to perform. In Sect. 3.3.1, we describe how the robots compute the cost estimates. In Sect. 3.3.2, we describe the abandon mechanism that prevents potential deadlocks in the system. In Sect. 3.3.3, we present algorithms that the robots can employ to select which action to perform. In Sect. 3.3.4, we illustrate how the robots communicate with each other and use the information received.

3.3.1 Computation of the cost estimates

As mentioned above, the robots use cost estimates to select the action they should perform. In the foraging scenario studied here, possible actions are: (i) HARVEST an object from the source, (ii) STORE an object in the nest, (iii) DROP an object in the cache, and (iv) PICK UP an object from the cache. Robots aim at maximizing the number of objects delivered to the nest in a given time, which is equivalent to minimizing the time needed to deliver each object. Therefore, in this study costs are expressed as time. Each robot computes four cost (time) estimates $\hat{t}_i$, each corresponding to the robot's estimate of the duration of the (sub-)task i. The estimates $\hat{t}_i$ are computed as a recency-weighted average of the observed costs:

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

where $\alpha \in (0, 1]$ is a weight factor and t_M is the measured time required to perform the corresponding action i.[1] We decided to use a recency-weighted average since it gives more

---

[1]In the experiments, α is set to 0.5, refer to Sect. 5 for details.

weight to recent observations, resulting in a more reactive behavior in non-stationary environments compared to a simple average (Sutton and Barto 1998).

Each robot initializes the estimates $\hat{t}_i$ with a random value (refer to Sect. 5 for details). The way t_M is measured depends on the estimate being updated. In all the cases, t_M measures the time taken between two sub-sequent decisions, made by the robot (see question marks in Fig. 2). When the robot estimates the time $\hat{t}_H$ for HARVESTING an object from the source, t_M measures the time from the moment an object was deposited in the nest to the moment the following object is harvested from the source, after navigating through the corridor. In case the estimate being updated is the time $\hat{t}_S$ required to STORE an object in the nest, t_M measures the time from the moment the robot takes an object from the source to the moment it deposits it in the nest, after navigating through the corridor. The estimate $\hat{t}_P$ of the time required to PICK UP an object from the cache is updated with the time t_M measured from the moment an object is deposited in the nest, to the moment the next object, picked up from the cache, is deposited in the nest. Analogously, a robot updates its estimate $\hat{t}_D$ of the time required to DROP an object in the cache with the time t_M measured from the moment an object is taken from the source to the moment the following object is taken from the source, after the first has been dropped in the cache.

3.3.2 Abandon mechanism

The abandon mechanism allows a robot to cancel the decision of using the cache. Abandoning is based upon a timeout mechanism: the robot measures the time t_M spent trying to access the cache and abandons the decision (i.e., it takes the corridor) if t_M is greater than a certain threshold. In our previous work, we proposed different methods to compute this threshold (see Pini et al. 2012a). We report here the two methods that are utilized in the experiments presented in this work.[2]

The first method consists in computing two thresholds, τ_P and τ_D. The threshold τ_P is used when a robot is trying to pick up an object from the cache, τ_D when a robot is trying to drop an object in the cache. The two thresholds are computed as follows:

$$\tau_P = 3\hat{t}_P, \qquad \tau_D = 3\hat{t}_D. \tag{2}$$

The second method uses a single threshold τ, that is derived using a formula analogous to Eq. 1. Each time a robot utilizes the cache to pick up or to drop an object, the measured time t_M required to use the cache updates a value $\tilde{t}$:

$$\tilde{t} = (1 - \alpha)\tilde{t} + \alpha t_M, \tag{3}$$

α is the same weight factor used in Eq. 1 for computing the estimates $\hat{t}_i$. $\tilde{t}$ is initialized to the average of $\hat{t}_P$ and $\hat{t}_D$. The timeout threshold τ is then computed as

$$\tau = 3\tilde{t}. \tag{4}$$

Notice that, in case a robot abandons the decision of using the cache, its current value t_M updates the estimates $\hat{t}_P$ or $\hat{t}_D$ using Eq. 1 (and $\tilde{t}$ in case the timeout threshold is computed using Eq. 3).

---

[2]In our previous work, we selected for each algorithm one method for computing the threshold. In this study we use for each algorithm the corresponding method that was selected in the previous work (refer to Sect. 5).

3.3.3 Studied algorithms

In this work, we study three algorithms that the robots can use to take decisions on the basis of the cost estimates. These algorithms are compared to a set of four reference algorithms, in which the decisions do not depend on the value of the estimates. In the following we describe each algorithm in detail.

The AdHoc algorithm   The *AdHoc* algorithm was originally proposed in one of our previous works (Pini et al. 2011). A robot employing the AdHoc algorithm selects actions stochastically. After collecting an object from the source, a robot DROPS it in the cache with a probability P defined as

$$P = \begin{cases} [1 + e^{-S((\hat{t}_H + \hat{t}_S)/(\hat{t}_P + \hat{t}_D) - 1)}]^{-1}, & \text{if } \hat{t}_H + \hat{t}_S > (\hat{t}_P + \hat{t}_D), \\ [1 + e^{-S(1 - (\hat{t}_P + \hat{t}_D)/(\hat{t}_H + \hat{t}_S))}]^{-1}, & \text{if } \hat{t}_H + \hat{t}_S \le (\hat{t}_P + \hat{t}_D), \end{cases} \tag{5}$$

where S is a parameter that regulates the degree of exploration of the algorithm. Exploration consists in sampling less-advantageous solutions in order to detect variations in the environment that possibly made these solutions more advantageous. In the case of the AdHoc algorithm, the lower the value of S, the higher the exploration. In other words, the lower the value of S, the higher the ratio between $\hat{t}_H + \hat{t}_S$ (i.e., the estimated cost of using the corridor) and $\hat{t}_P + \hat{t}_D$ (i.e., the estimated cost of using the cache) must be in order to obtain a given partition probability P. After delivering an object to the nest, a robot PICKS UP the next one from the cache with the same probability P. Therefore, a robot employing the AdHoc algorithm partitions the transportation task with a probability P and performs transportation as one unpartitioned task with a probability $1 - P$.

The ε-Greedy algorithm   The *ε-Greedy* (Sutton and Barto 1998) is a simple stochastic algorithm that has been applied in many contexts. The robots employing the ε-Greedy select with a probability $1 - \varepsilon$ the action i with the lowest associated cost $\hat{t}_i$ and with probability ε a random action. ε is the only parameter of the algorithm and defines the degree of exploration: the higher ε, the higher the exploration.

The UCB algorithm   The *UCB* is a heuristic adaptation of the UCB1 policy presented in the work of Auer et al. (2002), that in turn was derived from the index-based policy proposed by Agrawal (1995). UCB1 is characterized by a rapid convergence because it was originally designed for stationary problems (Auer et al. 2002). The robots employing the UCB algorithm deterministically select the action to perform. For example, after depositing an object in the nest, a robot PICKS UP the following one from the cache if:

$$\hat{t}_P - \gamma \sqrt{\frac{2\ln(n_P + n_H)}{n_P}} < \hat{t}_H - \gamma \sqrt{\frac{2\ln(n_P + n_H)}{n_H}}, \tag{6}$$

otherwise the robot uses the corridor to HARVEST an object from the source. n_P and n_H are, respectively, the number of times the robot used the cache to PICK UP an object and the number of times the robot HARVESTED an object from the source (i.e., going through the corridor). γ is a parameter that allows tuning the degree of exploration: higher values of γ correspond to a higher exploration. A formula analogous to Eq. 6 is used by the robots to decide whether to DROP and object in the cache or STORE it in the nest.

The Exp3 algorithm Exp3 (Cesa-Bianchi and Lugosi 2006) is a well-known MAB solver designed to keep bounds on regret in a variety of settings, including the pessimistic *adversarial* setting, where the problem is designed to be deceptive for the player. Using Exp3 the robots stochastically select between the cache and the corridor. After taking the ith object from the source, a robot DROPS it in the cache with a probability $P_{D,i}$ defined as

$$P_{D,i} = (1 - \gamma) \frac{e^{-\eta L_{D,i}}}{e^{-\eta L_{D,i}} + e^{-\eta L_{S,i}}} + \frac{\gamma}{2}, \tag{7}$$

where $L_{D,i} = \sum_{j=1}^{i-1} l_{D,j}$ and the values $l_{D,j}$ are computed as

$$l_{D,j} = \begin{cases} \frac{\tilde{t}_{MD,j}}{P_{D,j}} & \text{if the } j\text{th object was dropped in the cache,} \\ 0 & \text{if the } j\text{th object was stored in the nest via the corridor,} \end{cases} \tag{8}$$

$L_{S,i}$ is computed in an analogous fashion, substituting D for DROP with S for *STORE*. $\tilde{t}_{MD,j}$ in Eq. 8 is analogous to t_M in Eq. 1: it is the measured time required to drop the jth object in the cache. In this case the measure is normalized, using a maximum and minimum values set a priori (refer to Sect. 5 for details).

The parameter γ in Eq. 7 is computed as

$$\gamma = C \left(\frac{4 \ln 2}{n_D + n_H} \right)^{1/3}, \tag{9}$$

and the parameter η as

$$\eta = \frac{1}{C} \left(\frac{\ln 2}{2(n_D + n_H)} \right)^{2/3}. \tag{10}$$

In Eqs. 9 and 10, n_D and n_H are the total number of times the robot DROPPED an object in the cache and the number of times the robot STORED an object in the nest (i.e., traversing the corridor), respectively. In other words, $n_D + n_H$ is the total number of decisions that the robot made concerning where to deposit an object taken from the source (also refer to Fig. 2), therefore corresponding to the value of i in Eq. 7.[3] C is a constant value, computed as

$$C = (\sqrt{e} - 2)^{2/3}. \tag{11}$$

Analogous formulas define the probability $P_{P,i}$ of PICKING UP an object from the cache after the robot deposited its ith object in the nest (substitute DROP with PICK UP and STORE with HARVEST in all the formulas).

The reference algorithms The reference algorithms are used as a yardstick to evaluate the algorithms presented above. The first reference algorithm is the *never-partition* algorithm, which consists in always using the corridor to harvest and store objects. Analogously, the *always-partition* algorithm consists in always using the cache, to pick up and drop objects. In case the robots use the always-partition algorithm, they are prevented from abandoning (see Sect. 3.3.2). The third reference algorithm is the *random* algorithm. Using this algorithm, the robots select the actions to perform stochastically with equal probability. Finally, the

---

[3]Notice that γ and η are not constants, due to their dependency on $n_D + n_H$.

informed algorithm consists in always selecting the cache or the corridor on the basis of an external oracle information provided to the robots. The performance of the informed algorithm is therefore an upper bound for the performance of the studied algorithms. Notice that both the random and the informed algorithms allow the robots to abandon the decision of using the cache.

3.3.4 Communication

The robots are equipped with a communication device that allows them to exchange information. Each robot communicates the time t_A associated to the last action A it performed. The robots integrate their own cost estimates with the information received. Each robot uses received information as if it were its own observation—i.e., as if the robot itself performed the action A and measured the time t_A. Equation 1 is utilized to update the estimate using the information received (A identifies the index i and t_A replaces t_M).

The communication device of the robots imposes many limitations. First, the communication range is limited to roughly 0.75 m, and robots can only exchange messages when they are in line of sight. Second, a robot can receive one message per control step at most. Third, each message has a payload limited to 16 bits.

Communication is implemented as explained in the following. As mentioned, a robot communicates the measured time t_A associated to the last performed action A. The first 2 of the 16 payload bits are used to identify the action A: HARVEST, STORE, DROP, or PICK UP. The remaining 14 bits directly express the value t_A, measured in control cycles.[4] Each robot broadcasts a message every control cycle.

At most one message per control cycle can be received by a robot. To avoid using the same piece of information more than once, each robot memorizes the last 10 received messages (i.e., 16-bits numbers). Each time a robot receives a new message, it checks it against the contents of the buffer. If the same message is already present, the robot discards it; otherwise, the received information integrates the cost estimates of the robot, and the message is inserted in the buffer. The message buffer is managed using a FIFO[5] policy: if a new message must be added to the buffer and the buffer is full, the oldest message in the buffer is discarded. Notice that, in principle, it is possible that two different robots send the same message (i.e., they performed the same action A and measured the same time t_A). A robot receiving a message from both robots would therefore discard the second message received, assuming that the two messages came from the same sender.

4 Experimental setup

In this section, we present the tools that we employed to carry out the experiments described in Sect. 5. All the experiments presented in this work have been carried out in simulation. In Sect. 4.1 we describe the e-puck, which is the robot that we simulate in the experiments. In the same section we present the TAM, a device that is used in the experiments to abstract object handling. In Sect. 4.2, we illustrate the environment in which the robots perform foraging. In Sect. 4.3, we describe ARGoS, which is the simulator employed to carry out our study.

---

[4]Each control cycle lasts 0.1 seconds.
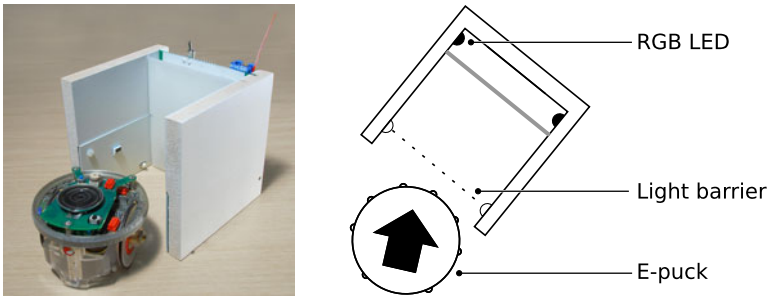
[5]Acronym for *first in, first out*.

Fig. 3 An e-puck entering a TAM: picture (*left*) and schematic representation (*right*)

4.1 Robots and objects

In this work, we perform foraging experiments using a simulation of the *e-puck*.[6] The e-puck is a small wheeled robot that has been developed by Mondada et al. (2009) as an open tool for university students. Each e-puck features a minimal set of sensors and actuators that allow the robot to navigate and interact with other robots. The e-puck has a modular structure that permits to add extension boards that enhance the basic capabilities of the robot. In this work, we employ the infrared ground sensors board and the infrared range and bearing board (for the latter refer to Gutiérrez et al. 2009). The infrared range and bearing board allows line of sight communication between the e-pucks.

The e-puck does not have the capability of grasping and manipulating objects. To overcome this limitation, in the experiment we simulate a device developed by Brutschy et al. (2010), the *TAM* (acronym for *task abstraction module*).

Figure 3 reports a picture of an actual TAM (left) and a schematic representation of an e-puck entering a TAM (right). Each TAM is a small booth that can be entered by one e-puck at a time. The TAM features two RGB LEDs that can be perceived by the e-puck by means of the front camera. The TAM is also equipped with an infrared barrier that can detect the presence of a robot inside the TAM. The user can program the TAM and define its behavior, that is, the way LEDs are actuated on the basis of the event detected (e.g., a robot entering or exiting the TAM).

In this work, we use TAMs to abstract object handling. In the experiments, a TAM whose LEDs are green represents an object that can be collected by an e-puck by entering that TAM. Conversely, a TAM whose LEDs are blue represents a location where an object can be deposited. In both cases, the TAM temporarily turns the LEDs to red when a robot is inside. This serves as an acknowledgement mechanism for the robot, to confirm its presence inside the TAM. The robots themselves keep track of whether they are carrying objects or not and behave accordingly.

In the foraging experiments, the source, the nest, and the cache are implemented with arrays of TAMs. The TAMs at the source are always green, to represent an object source that never depletes. The TAMs at the nest are always blue, to represent unlimited storage space at the nest. The cache is implemented using an array of paired TAMs: the opening of one TAM is oriented towards the source and the opening of the other towards the nest.
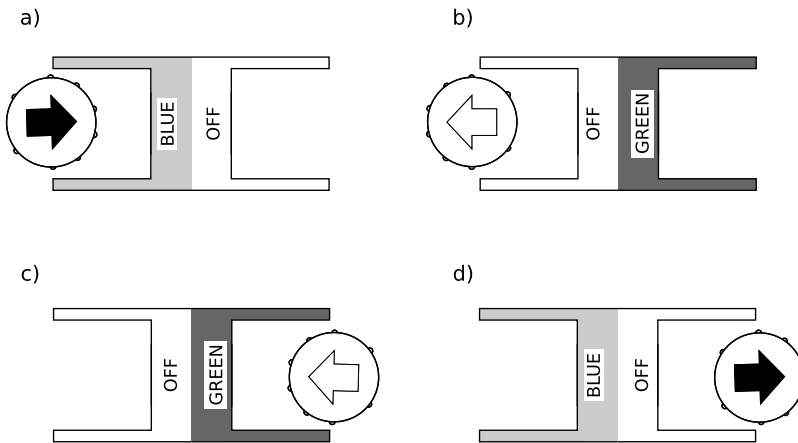
---

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

Fig. 4 Implementation of a cache slot using two paired TAMs. The source (not represented) is located on the left-hand side, the nest (not represented) on the right-hand side. Robots carrying an object are marked with a *black arrow*, robots not carrying an object with a *white arrow*. (**a**) Initial configuration: the cache slot is empty, a robot can enter the TAM oriented towards the source (lit up in *blue*) to drop an object. (**b**) An object has been deposited by a robot; the TAM oriented towards the source turns the LEDs off and the paired TAM turns on the LEDs in *green*. (**c**) The object is available at the TAM oriented towards the nest and another robot enters the TAM to pick up the object. (**d**) When this robot leaves with the object, the two TAMs return to their initial configuration
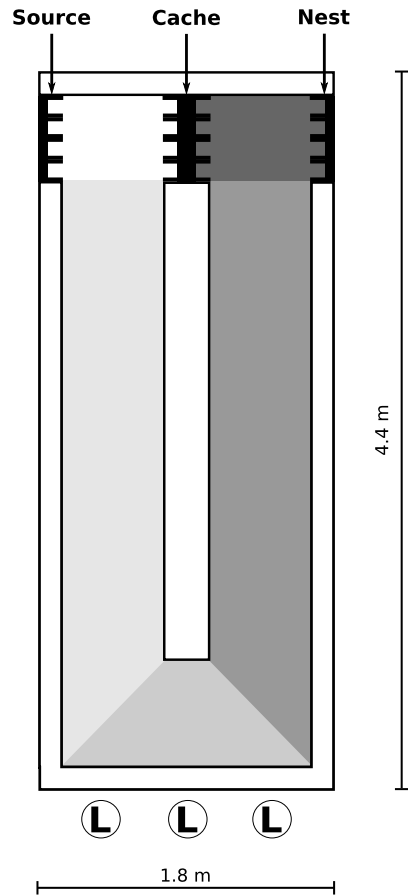
Figure 4 depicts the behavior of two paired TAMs that implement a slot in the cache.[7] In the figure, a robot carrying an object is marked with a black arrow, a robot without an object with a white arrow. Figure 4(a) shows an e-puck carrying an object that enters an empty cache slot. The cache TAM facing the source (left-hand side) is blue, the paired TAM facing the nest is off. In Fig. 4(b) the robot is leaving the TAM, after depositing the object previously carried. The object becomes available on the other side of the cache slot: the TAM facing the nest (right-hand side) is now green and therefore an object can be picked up there. The paired TAM facing the source side is switched off, to avoid additional objects to be delivered there. In Fig. 4(c) a second robot enters the cache slot on the nest side, to pick up the object. When the robot leaves the TAM with the object (Fig. 4(d)), the cache slot returns to its initial configuration: an object can be dropped again in the cache at the source side. A video that illustrates the behavior of the cache can be found in Pini et al. (2012c). The interfacing time Π is implemented as a delay between the two phases represented in Fig. 4(a) and (b). After the robot entered the TAM oriented towards the nest, the TAM turns its LEDs to red (not shown in figure) to acknowledge the robot presence, the LEDs do not turn off (i.e., the robot cannot leave the TAM) until a time equal to Π has passed. Analogously, the transition between the phases represented in Fig. 4(c) and (d), requires a time Π.

4.2 Implementation of the foraging environment

The environment in which we carry out the experiments is represented in Fig. 5. It is a 1.8 m by 4.4 m rectangular arena surrounded by walls. The source and the nest are implemented

---

[7]The TAMs can communicate via WiFi. The behavior of a cache slot, that is, the coordination between two paired TAMs, can be easily implemented in the real-world by interfacing the TAMs with a PC.

Fig. 5 The environment in
which the robots perform
foraging. The source, the nest,
and the cache are implemented
using arrays of TAMs. Different
areas in the arena are marked
with a specific ground color, that
can be recognized by the robots.
A row composed of three light
sources (each marked with "L")
is used by the robots as a
directional clue for navigation

using four TAMs organized in arrays. The cache consists of two arrays of four TAMs each, one facing the source and one facing the nest. A row of three light sources, located at the bottom of the arena, is used by the robots as a directional cue for navigation. The robots estimate the direction of the lights using the eight on-board ambient light sensors. The directional information allows the robots to traverse the corridor and to estimate their heading with respect to source, cache, and nest, which is necessary to navigate towards the TAMs. The color of the ground, different for every location in the arena, is recognized by the robots by means of the infrared ground sensors. This information is used by the robots to determine their location in the arena. While navigating the environment, the robots perform obstacle avoidance using the eight infrared proximity sensors as bumpers.

4.3 ARGoS

All the experiments presented in this paper have been carried out using *ARGoS*, a physics-based simulation software developed within the *Swarmanoid*[8] project (Dorigo et al. 2013). ARGoS allows the real-time simulation of thousands of robots and can be customized by

---

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

the user (Pinciroli et al. 2012). In the experiments presented in this work, the e-puck and the TAM are simulated using the 2D-dynamics physics engine available in ARGoS. The simulation proceeds at discrete steps. At each simulation step, a uniformly distributed random value between -5 % and 5 % of the reading is added to the ambient light, proximity, and ground sensor readings. The maximum communication distance of the simulated range and bearing board is set to 0.75 m; each robot has a 30 % probability per control step of not receiving any message (even the ones that were sent within communication range).

5 Experiments and results

We carry out all the simulation-based experiments with a swarm of 20 e-pucks. At the beginning of each experimental run, 10 robots are placed in the area containing the source and 10 in the area containing the nest. The initial position and orientation of each robot are selected randomly. We chose this initial configuration because the always-partition algorithm requires an equal number of robots to be positioned on both sides of the cache, in order to use it optimally. To allow for a fair comparison of the algorithms, we use the same initialization also for the other algorithms. The cost estimates are initialized stochastically: we uniformly sample $\hat{t}_P$ and $\hat{t}_D$ from the interval $[20, 40]$ and $\hat{t}_H$ and $\hat{t}_S$ from the interval $[40, 80]$. Each experimental run amounts to a time of 10 simulated hours. For each experimental condition we perform 100 runs, varying the seed of the pseudo-random number generator.

For each algorithm presented in Sect. 3.3.3, we test two versions that differ in the amount of exploration, controlled by setting the corresponding parameter (S for the AdHoc, γ for UCB, and ε for ε-Greedy), and in the method utilized to compute the timeout threshold, used by the abandon mechanism. In our previous work, we carried out experiments targeted at determining the values of the parameters of the algorithms (refer to Pini et al. 2012a, for details). In this work, we use the values previously determined, which are summarized in Table 1. The value of the parameter α, which is used to compute estimates according to Eq. 1, is set to 0.5 for all the algorithms. Note that Exp3 has no parameters and therefore there is only one version of the algorithm. We tested Exp3 with both methods to compute the abandon threshold presented in Sect. 3.3.2. The results obtained with the two are comparable and therefore we chose to report in the paper only the ones obtained with Eq. 2. Complete results are available in Pini et al. (2012c). As mentioned in Sect. 3.3.3, Exp3 requires the time measures of the robots to be normalized (Eq. 8). The values used for normalization are 12 s for the minimum and 300 s for the maximum, determined as follows. The minimum

Table 1 Selected parameters for the exploiting and exploring versions of the studied algorithms

Algorithm	Version	Parameter	Abandon
AdHoc	exploiting	$S = 6.0$	Eq. 4
	exploring	$S = 1.0$	Eq. 4
UCB	exploiting	$\gamma = 100$	Eq. 4
	exploring	$\gamma = 1000$	Eq. 2
ε-Greedy	exploiting	$\varepsilon = 0.01$	Eq. 2
	exploring	$\varepsilon = 0.11$	Eq. 2
Exp3	does not apply	None	Eq. 2

corresponds to the minimum time observed by a robot to access the cache for $\Pi = 0$ s. The maximum corresponds to the maximum time observed by a robot to travel along the corridor (one direction only). Both values have been determined in experiments in which the robots employ the random algorithm.

The main goal of the experiments is to determine the effect of communication on the studied system: we compare two versions of each algorithm, with and without communication, labeled *social* and *non-social*, respectively.

The rest of this section is organized as follows. In Sect. 5.1, we present experiments in which we study the behavior of the non-social version of the algorithms. In Sect. 5.2, we describe experiments in which we study the social versions. In Sect. 5.3, we propose a modification to the AdHoc and UCB algorithms, which we made to tackle an issue that emerged in the experiments performed with the social version of the algorithms.

5.1 Non-social algorithms: experiments and results

In this section, we present the experiments carried out with the non-social version of the studied algorithms. In a first group of experiments, we test the capability of the swarm to decide whether to use the cache or the corridor for different values of the cache interfacing time Π. We test two cases: in one, we set Π to 0 s, which means that the cache is preferable over the corridor; in the other, we set Π to 160 s, which means that the corridor is preferable.

Figure 6 reports the performance of the studied algorithms, for $\Pi = 0$ s (top) and $\Pi = 160$ s (bottom). We measure the performance of an algorithm as the total number of objects delivered to the nest by the swarm, when that algorithm is employed. The results confirm that, for $\Pi = 0$ s, the cache is advantageous and the best performing algorithm is the always-partition algorithm. Dually, for $\Pi = 160$ s, the corridor is advantageous and the never-partition algorithm is the best performing. With the exception of Exp3, the studied algorithms perform well in both environments. The results obtained by the ε-Greedy and UCB algorithms confirm that existing algorithms that are used in the literature for tackling the multi-armed bandit problem can be successfully applied to the task partitioning problem. The poor performance of Exp3 can likely be ascribed to an excessive exploration due to its pessimistic assumptions: in an adversarial setting, one can never be too careful in exploiting.

Given that the problem is stationary (i.e., Π does not vary in time), the exploiting version of each algorithm performs better than the corresponding exploring version. In fact, once the robots have determined whether the cache is advantageous over the corridor or not, they do not need to keep exploring and the exploiting version of the algorithms is more efficient.

To test a more challenging situation for the robots, we study a non-stationary environment in which we vary Π during the course of the experimental run. Varying the value of the cache interfacing time allows us to model situations in which the environmental conditions change and render task partitioning more or less advantageous.

We test two cases; in one case the interfacing time is initialized to $\Pi = 0$ s and, at time $t = 2.5$ hours (i.e., one quarter of experiment), we set it to 160 s. Therefore, the cache is initially preferable (for $t < 2.5$ hours). After Π changes, the cache becomes costly and the robots should utilize the corridor. In the other, dual case, we initialize Π to 160 s and we set its value to s at $t = 2.5$ hours. When the robots use the informed algorithm their behavior is hard-coded. While Π is low, the robots always use the cache. Conversely, when Π is high, they always traverse the corridor. As mentioned in Sect. 3.3.3, the informed algorithm is an upper bound for the other algorithms.
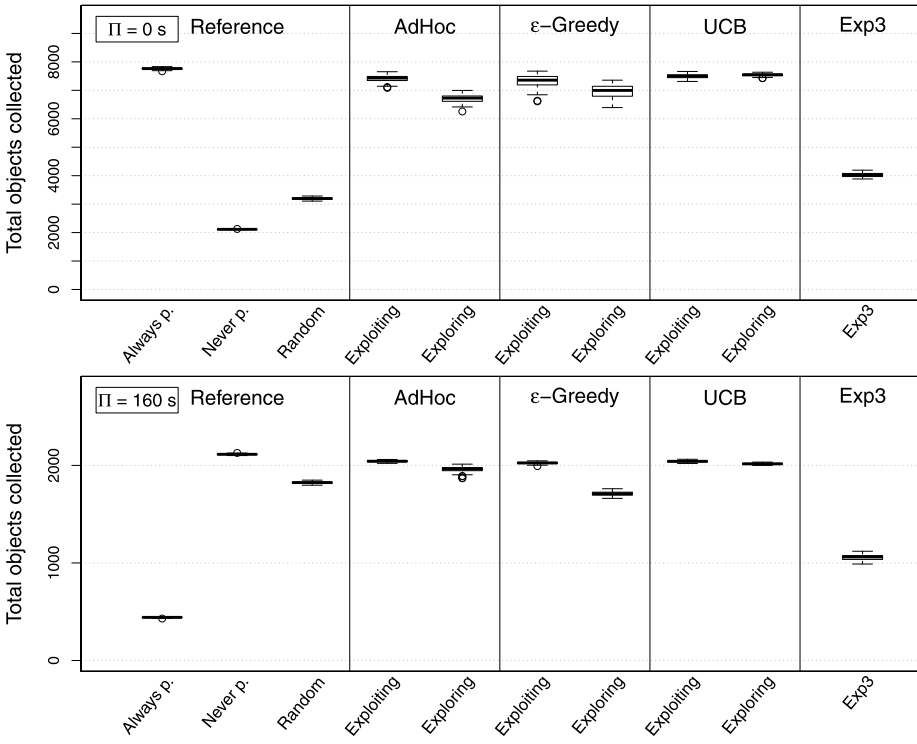
Fig. 6 Performance of the studied algorithms, measured as objects delivered to the nest by the swarm. The *top plot* reports the results obtained for $\Pi = 0$ s, the *bottom plot* for $\Pi = 160$ s
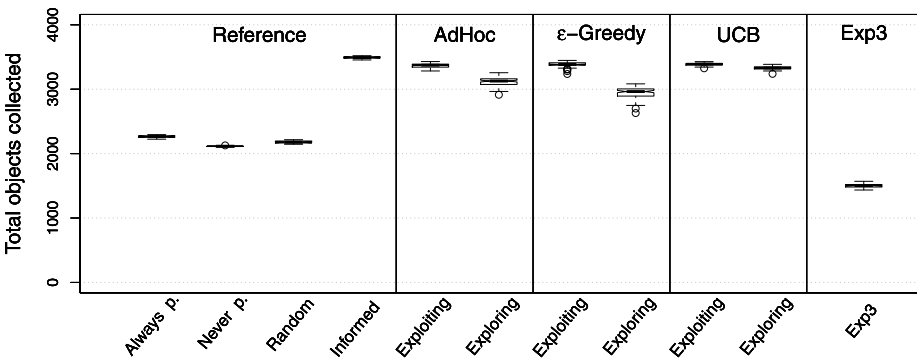

Fig. 7 Performance, measured as objects delivered to the nest by the swarm, for a sub-set of the algorithms. The figure reports the results obtained for the experimental setup in which Π is initialized to 0 s and set to 160 s when the experimental run reaches a quarter of its duration

Figure 7 reports the performance of the two versions of each algorithm and the four reference algorithms, for the case in which Π is changed from 0 s to 160 s. The results reported in the figure show that the exploiting version of each algorithm performs better than the corresponding exploring version and reaches performance levels close to the ones of the
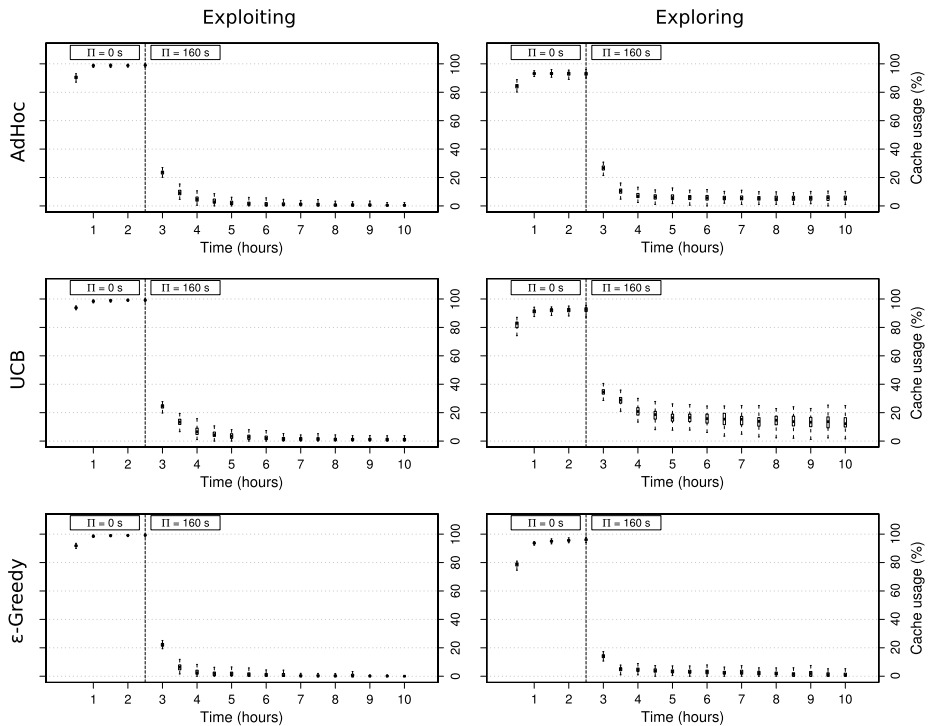
Fig. 8 Percentage of usage of the cache in time, for the different versions of the algorithms. The *plots* report the results of the experiment in which Π is initialized to 0 s. Each *box* reports the overall data collected over 100 experimental runs, for the period of 30 minutes that precedes the value reported on the *X* axis. The vertical *dashed line* marks the instant in which the value of Π is changed

informed algorithm. As in the previous case, Exp3 performs badly in the tested experimental conditions.

Figure 8 reports the percentage of usage of the cache in time for the studied algorithms. The plots in the same row refer to the same algorithm, from top to bottom: AdHoc, UCB, and ε-Greedy. [9] The left-hand side column of plots reports the results of the exploiting version of the algorithms, the right-hand side column of plots the results obtained with the exploring version of the algorithms. Each box reports the percentage of usage of the cache in the 30 minutes that precede the time reported on the *X* axis. The plots show that, in all the cases, the robots initially identify the cache as the best choice and utilize it most of the times. When Π changes from 0 s to 160 s, the robots switch to the corridor, which becomes advantageous over the cache. Since in the first part of the experiment the robots mostly use the cache, they directly perceive the variation of Π and react to the change. The difference between the exploring and exploiting version of the algorithms is that the exploring version samples the (perceived) less advantageous option with a higher frequency.

This results in a loss of performance, as observable in Fig. 7. Therefore, in case Π varies from a low to a high value, the swarm does not benefit from exploration: the variation directly impacts the cost of the choice selected the most by the robots (i.e., using the cache)

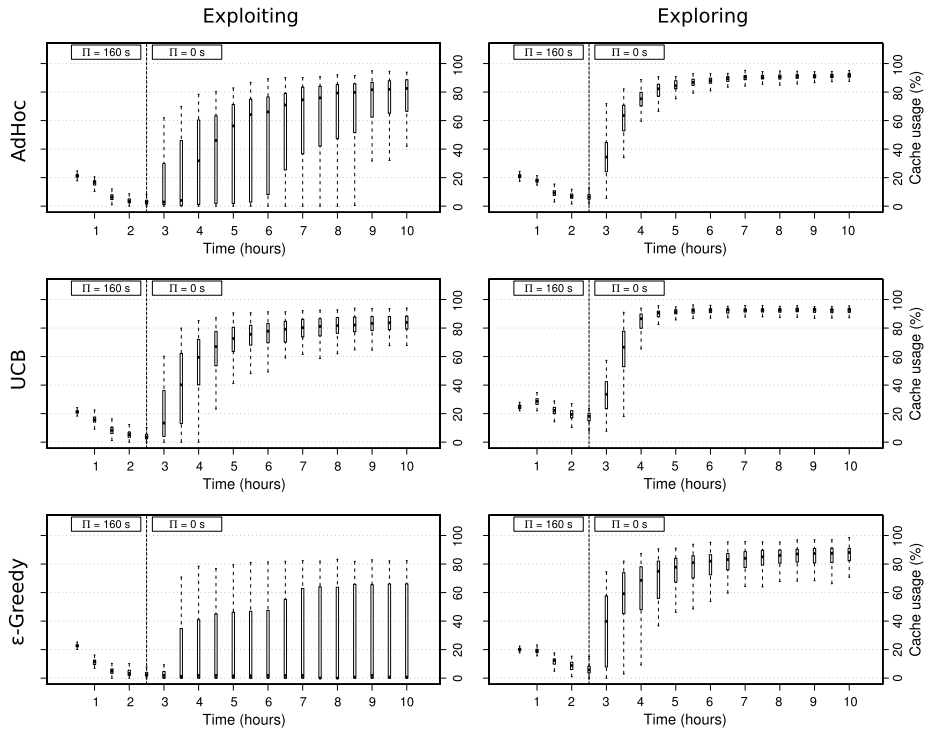[9]An analogous plot for Exp3 is available in Pini et al. (2012c).

Fig. 9 Percentage of usage of the cache in time, for the different versions of the algorithms. The *plots* report the results of the experiment in which Π is initialized to 160 s. Each *box* reports the overall data collected over 100 experimental runs, for the period of 30 minutes that precedes the value reported on the X axis. The vertical *dashed line* marks the instant in which the value of Π is changed

and therefore the change can be perceived directly by the swarm. The dual case in which Π varies from a high to a low value presents a different challenge to the robots. Figure 9 reports, analogously to Fig. 8, the percentage of usage of the cache in time, for the two versions of the studied algorithms. In this case, the initial value of Π is high and the robots initially select the corridor more frequently than the cache. This implies that, differently from the previous case, the variation occurring at the cache cannot be directly detected by all the robots in the swarm. The plots on the left-hand side of Fig. 9 show that the exploiting versions of the algorithms struggle to detect the variation of Π. The exploring versions of the algorithms, on the other hand, allow the swarm to adapt their choice to the new value of Π. This indicates that, in this case, exploration is beneficial.

Figure 10 confirms that indeed exploration entails benefits. In this case, the performance of the exploring version of the algorithms is higher than the performance of the corresponding exploiting versions. The results highlight that, as in the multi-armed bandit problem, also in the task partitioning problem a tradeoff exists between exploiting the cumulated knowledge and exploring the environment to detect changes and react to them. Compared to the previous case (Fig. 7) Exp3 performs relatively better, but its performance is still far from the one of the best algorithms.
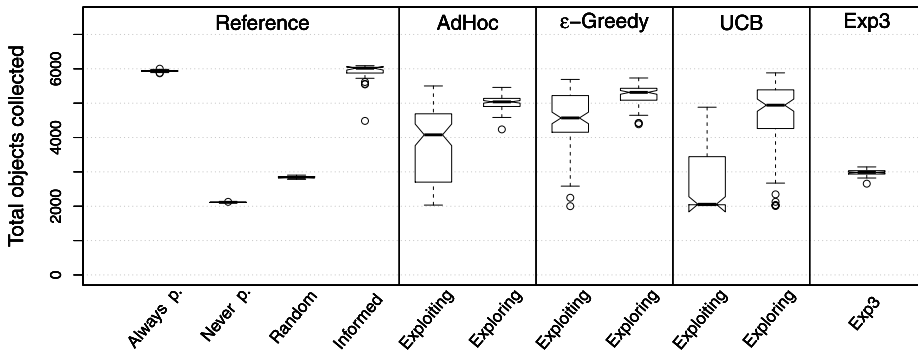
Fig. 10 Performance of the algorithms for the case in which Π is initialized to 160 s and set to 0 s when the experimental run reaches a quarter of its duration
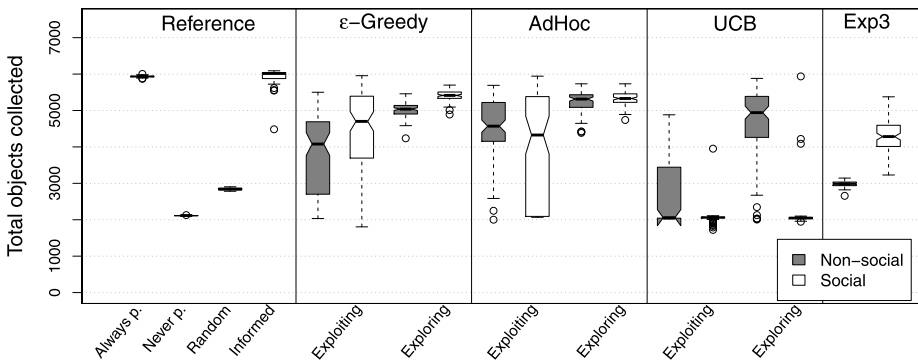

Fig. 11 Performance of the studied algorithms for the case in which Π is initialized to 160 s and set to 0 s when the experimental run reaches a quarter of its duration. For each algorithm (excluding the reference algorithms) we report the results obtained with the non-social (*gray boxes*) and social version (*white boxes*)

5.2 Social algorithms: experiments and results

In this section, we present the experiments that we carried out to test the influence of communication on the behavior of the system. We focus on the case in which the cache interfacing time is initialized to 160 s and set to 0 s at $t = 2.5$ hours. We study the behavior of the social and non-social versions of the AdHoc, ε-Greedy, and UCB algorithms.

Figure 11 reports the performance of the algorithms. Each plot reports the performance of the four reference algorithms and the four different versions of each algorithm (exploring/exploiting and social/non-social). In the plot, the gray and white boxes report the results of the non-social and social versions of the algorithms, respectively. The data reported in the figure show that communication affects the swarm differently, depending on the algorithm being utilized by the robots. Communication affects positively the ε-Greedy algorithm: the performance increases independently of the setting of the parameter ε. Communication also improves the performance of Exp3. Conversely, communication has a strong negative effect on the UCB algorithm, independently of the value of γ. The effect of communication on the AdHoc algorithm depends on its version. The performance of the exploring version in-
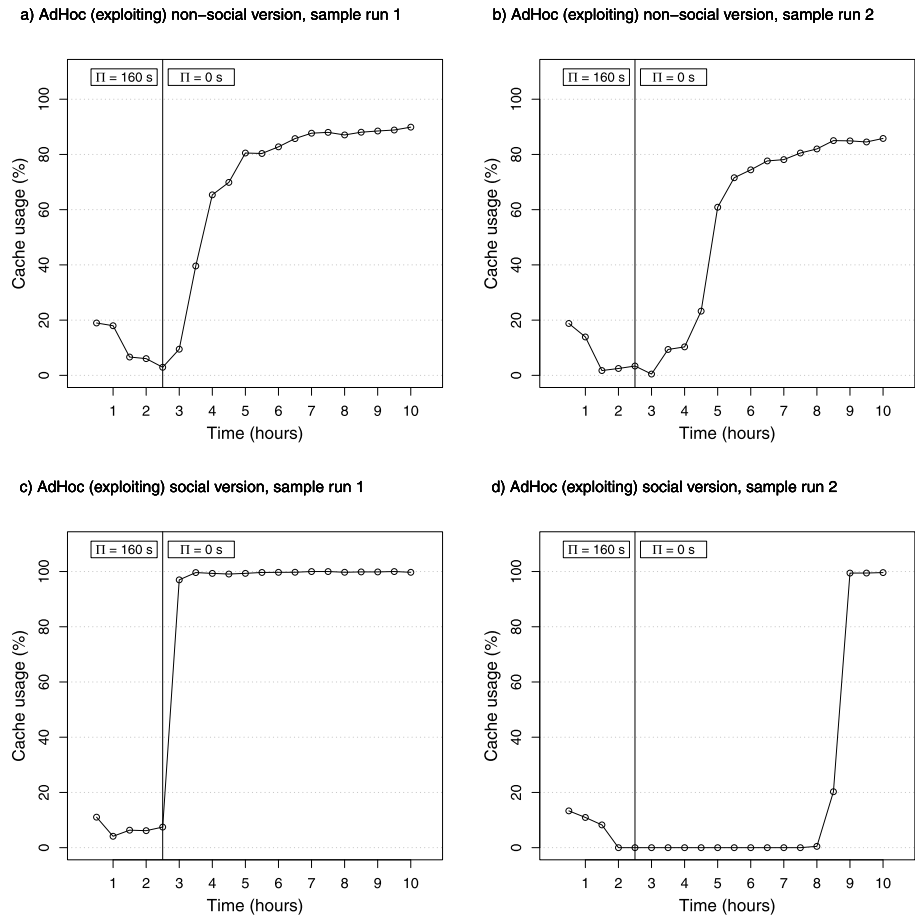
Fig. 12 Cache usage in time, as observed in four selected experimental runs. *Top*: non-social version of the AdHoc algorithm (exploiting). *Bottom*: social version of the AdHoc algorithm (exploiting)

creases slightly, while there is an increase in the variability of the results obtained with the exploiting version.

To understand the effect of communication on the exploiting version of the AdHoc algorithm, we study the behavior of the robots across single experimental runs. In the plots of Fig. 12, we report the percentage of use of the cache in time, for four experimental runs. The plots in the first row report the results obtained with the non-social version of the AdHoc algorithm, the plots in the second row for the case in which the social version is employed.

The runs reported here are examples that allow us to illustrate general trends that we observe in the experiments. Analogous plots for each experimental run are available in (Pini et al. 2012c). In general, communication renders the choice made by the robots of the swarm more uniform. This affects the system in two ways. On the one hand, as shown in Fig. 12(c) and Fig. 12(d), the transitions are sharper. If a few robots detect that the cache became advantageous, information spreads within the swarm and all the robots rapidly switch to using the cache. The sooner this happens, the more the robots can exploit the benefits of

using the cache. In case the robots do not communicate (Figs. 12(a) and 12(b)), the transition is slower, because every robot has to detect by itself that the cache became advantageous.

Besides rendering the transitions quicker, the fact that the choice made by the robots is strongly biased towards one of the two options also entails the risk that the change occurring at the cache goes undetected, as in the case reported in Fig. 12(d), where the robots detect the variation of Π only at the end of the experiment.

In the case of UCB, the situation is pushed to the extreme: in all the runs, the robots converge to the usage of the corridor and cache is never sampled again. Therefore, the robots are not able to detect that the cache became advantageous and the resulting performance of the swarm is low (see Fig. 11). This is likely due to the fact that, as mentioned in Sect. 3.3.3, the algorithm on which our UCB heuristic is based was originally designed for stationary problems (Auer et al. 2002) and it is therefore characterized by a rapid convergence, which is here further accelerated by communication.

Differently from the other algorithms, ε-Greedy algorithm does not suffer the mentioned problem. This is because the probability of selecting the action perceived as the worst does not change with the estimated action cost, being always ε. Therefore, the ε-Greedy draws only benefits from the increased flow of information.

5.3 Algorithms with ε-exploration: experiments and results

To face the problems we encountered with the AdHoc and UCB algorithms, we modify the two algorithms to include a term of ε-exploration, as in ε-Greedy: with probability $1 - \varepsilon$, the robots make a choice according to the corresponding algorithm and with probability ε the robots choose randomly. We select the value $\varepsilon = 0.01$, which is the same utilized in the exploiting version of the ε-Greedy algorithm. We refer to the modified versions of the AdHoc and UCB algorithms as the ε-AdHoc and ε-UCB algorithms, respectively.

Figure 13 highlights the effect of ε-exploration on the performance of the different versions of the UCB (top) and AdHoc (bottom) algorithms. The top plot shows that, in general, the UCB benefits from the ε-exploration, the only exception being the exploiting version that uses communication.

The social version of the AdHoc algorithm also benefits from the added ε-exploration. The performance of the ε-AdHoc (in its exploiting version) is higher than the one of the AdHoc and the variability in the results is reduced. The ε-exploration, together with communication, allows the AdHoc algorithm to have a portion of exploitation, which allows the swarm to use efficiently the cache (or the corridor) and, at the same time, to remain flexible with respect to changes.

6 Conclusions

Task partitioning, the process by which tasks are decomposed into sub-tasks, can be beneficial for the organization of work in groups of individuals. Examples in nature demonstrate that social insects benefit from task partitioning in terms of an increased exploitation of specialization, enhanced efficiency, and physical separation of the workers. Swarms of robots share many similarities with colonies of social insects and can draw the very same benefits from partitioning their tasks. Task partitioning also entails overhead costs, that are due to the increased coordination required within the members of the swarm. Depending on the specific situation and on the costs involved, task partitioning may or may not be advantageous. It is therefore important that a swarm properly selects whether to employ task partitioning or not.
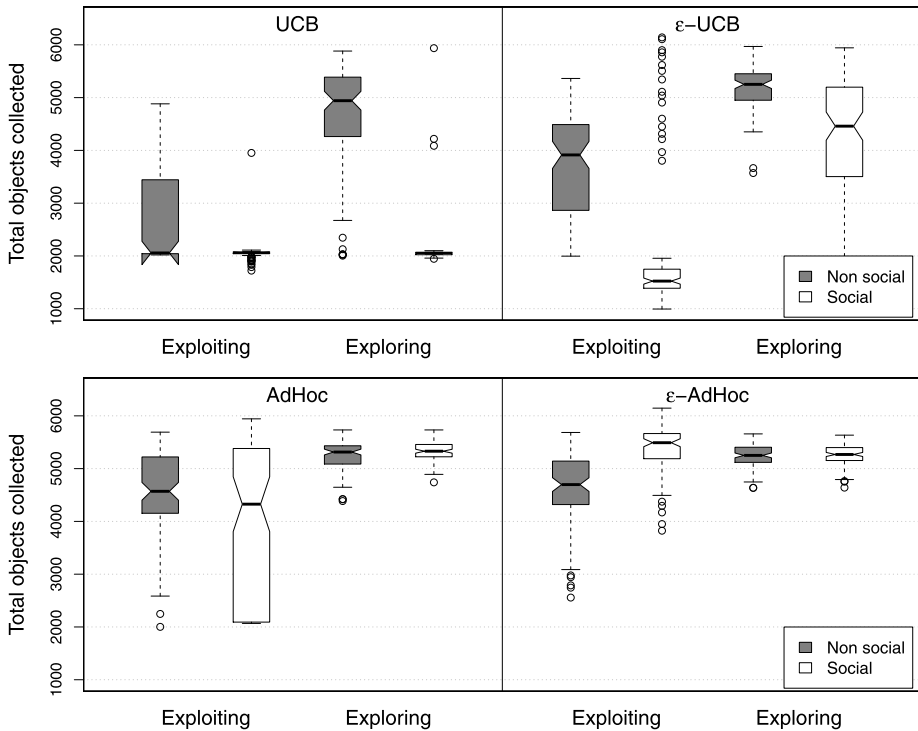
Fig. 13 Performance of the UCB (*top*) and AdHoc (*bottom*) algorithms, with and without ε-exploration. The *gray boxes* report the data for the case in which communication is not used, the *white boxes* for the case in which communication is used

In previous research works, we proposed distributed methods to tackle the task partitioning problem: selecting whether to perform a given task as a whole, or perform it as a sequence of sub-tasks. The research work presented in this article builds upon our previous research. As in the previous works, the robots take decisions individually, on the basis of the estimated cost of task partitioning. We extend the system by adding a social component within the swarm: the robots exchange information about the environment by communicating explicitly. Each robot communicates the measured cost of the last action it performed and uses the received information to integrate the cost estimates used to take decisions.

We evaluate the system in simulation-based experiments, using a foraging scenario as a testbed. We study three different algorithms, that the robots can utilize to solve the task partitioning problem: an AdHoc algorithm, that we explicitly designed to tackle the task partitioning problem and three algorithms taken from the multi-armed bandit literature, ε-Greedy, UCB, and Exp3. We compare the algorithms in their *social* and *non-social* versions, the former using communication.

In the experiments, we compare situations that differ in terms of the cost of using task partitioning. The results of the experiments confirm that existing algorithms for the multi-armed bandit problem can be successfully employed in the context of swarm robotics to perform task partitioning. We point out that, as in the multi-armed bandit problem, also in the task partitioning problem a tradeoff exists between exploration and exploitation. The

results obtained with the social version of the algorithms indicate that communication leads to conformism within the swarm and strongly biases the choice made by the robots. On the one hand, this renders the swarm quicker in taking decisions, but on the other hand it may prevent the swarm from being able to detect variations occurring in the environment. We propose a modification to the ad hoc algorithm, in terms of built-in exploration, which improves the capability of the algorithm to react to changes occurring in the environment. We point out that the same modification does not improve the performance of UCB for the exploiting version of the algorithm that uses communication. This indicates that, contrary to what one might expect, communication does not always lead to benefits for the swarm.

Future research will aim at investigating more deeply the effects of communication in the studied system as well as studying different communication strategies and ways of using the information received. For example, the robots could directly exchange information about their internal states, rather than the last observation made in the environment. Another option is that the robots associate a confidence level to the information they are communicating. On the reception side, a robot would then use this confidence measure to weight the information it received. Finally, more complex communication strategies could be used to achieve explicit coordination among the robots of the swarm. A number of works in the literature of multi-agent systems has shown that coordination improves performance. Decentralized coordination is a well studied research domain and many solutions proposed in the literature can be applied. However, coordination approaches are in general much more complex, in terms of both communication and computation. The communication strategy used in our study, on the other hand, is minimalistic and it can be used by agents with limited communication capabilities.

References

Agrawal, R. (1995). Sample mean based index policies with $O(\log n)$ regret for the multi-armed bandit problem. *Advances in Applied Probability*, *27*, 1054–1078.

Anderson, C., & Ratnieks, F. L. W. (1999). Task partitioning in insect societies I: effect of colony size on queueing delay and colony ergonomic efficiency. *The American Naturalist*, *154*(5), 521–535.

Anderson, C., & Ratnieks, F. L. W. (2000). Task partitioning in insect societies: novel situations. *Insectes Sociaux*, *47*, 198–199.

Arnan, X., Ferrandiz-Rovira, M., Pladevall, C., & Rodrigo, A. (2011). Worker size-related task partitioning in the foraging strategy of a seed-harvesting ant species. *Behavioral Ecology and Sociobiology*, *65*(10), 1881–1890.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2), 235–256.

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, ULB.

Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, *38*(2), 156–172.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge: Cambridge University Press.

Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A. L., Decugnière, A., Caro, G. D., Ducatelle, F., Ferrante, E., Förster, A., Gonzales, J. M., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., de Oca, M. M., O'Grady, R., Pinciroli, C., Pini, G., Rétornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stützle, T., Trianni, V., Tuci, E., Turgut, A. E., & Vaussard, F. (2013, in press). Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*.

Drogoul, A., & Ferber, J. (1992). From tom thumb to the dockers: some experiments with foraging robots. In J.-A. Meyer, L. R. Herbert, & W. W. Stewart (Eds.), *Proceedings of the second international conference on simulation of adaptive behavior* (pp. 451–459). Cambridge: MIT Press.

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 fourth 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.

Frison, M., Tran, N.-L., Baiboun, N., Brutschy, A., Pini, G., Roli, A., Dorigo, M., & Birattari, M. (2010). Self-organized task partitioning in a swarm of robots. In M. Dorigo, M. Birattari, G. A. Di Caro, R. Doursat, A. P. Engelbrecht, D. Floreano, L. M. Gambardella, R. Groß, E. Sahin, H. Sayama, & T. Stützle (Eds.), *LNCS: Vol. 6234. Swarm intelligence, 7th international conference, ANTS 2010* (pp. 287–298). Berlin/Heidelberg: Springer.

Goldberg, D., & Matarić, M. J. (2002). Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. In T. Balch & L. E. Parker (Eds.), *Robot teams: from diversity to polymorphism* (pp. 315–344). Natick: AK Peters/CRC Press.

Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., & Magdalena, L. (2009). Open E-puck range & bearing miniaturized board for local communication in swarm robotics. In *ICRA'09: proceedings of the 2009 IEEE international conference on robotics and automation* (pp. 3111–3116). Piscataway: IEEE Press.

Hart, A. G., & Ratnieks, F. L. W. (2001). 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., & 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.

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

Lein, A., & Vaughan, R. T. (2009). Adapting to non-uniform resource distributions in robotic swarm foraging through work-site relocation. In *2009 IEEE/RSJ international conference on intelligent robots and systems (IROS'09)* (pp. 601–606). Piscataway: IEEE Press.

Lerman, K., & Galstyan, A. (2002). Mathematical model of foraging in a group of robots: effect of interference. *Autonomous Robots*, *13*, 127–141.

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 *Proceedings of the 9th conference on autonomous robot systems and competitions* (pp. 59–65). IPCB-Instituto Politecnico de Castelo Branco.

Nunes, L., & Oliveira, E. (2008). Communication during learning in heterogeneous teams of learning agents. *Intelligent Decision Technologies*, *2*(3), 153–166.

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 E. Andre, S. Sen, C. Frasson, & P. M. Jörg (Eds.), *AGENTS'01: proceedings of the fifth international conference on autonomous agents* (pp. 29–30). New York: ACM Press.

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: the state of the art. *Autonomous Agents and Multi-Agent Systems*, *11*(3), 387–434.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., & Dorigo, M. (2012). ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, *6*(4), 271–295

Pini, G., Brutschy, A., Birattari, M., & Dorigo, M. (2009). Task partitioning in swarms of robots: reducing performance losses due to interference at shared resources. In J. A. Cetto, J. Filipe, & J.-L. Ferrier (Eds.), *LNEE: Vol. 85. Informatics in control, automation and robotics* (pp. 217–228). Berlin/Heidelberg: Springer.

Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., & Birattari, M. (2011). Task partitioning in swarms of robots: an adaptive method for strategy selection. *Swarm Intelligence*, *5*(3–4), 283–304.

Pini, G., Brutschy, A., Francesca, G., Dorigo, M., & Birattari, M. (2012a). Multi-armed bandit formulation of the task partitioning problem in swarm robotics. In M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. P. Engelbrecht, R. Groß, & T. Stützle (Eds.), *LNCS: Vol. 7461. Swarm intelligence, 8th international conference, ANTS 2012* (pp. 109–120). Berlin/Heidelberg: Springer.

Pini, G., Brutschy, A., Scheidler, A., Dorigo, M., & Birattari, M. (2012b). Task partitioning in a robot swarm: Retrieving objects by transferring them directly between sequential sub-tasks. Technical Report TR/IRIDIA/2012-010, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Pini, G., Gagliolo, M., Brutschy, A., Dorigo, M., & Birattari, M. (2012c). Task partitioning in a robot swarm: a study on the effect of communication—Online supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2012-0015/.

Pini, G., Brutschy, A., Pinciroli, C., Dorigo, M., & Birattari, M. (2013, in press). Autonomous task partitioning in robot foraging: an approach based on cost estimation. Adaptive behavior.

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

Schlag, K. (1998). Why imitate, and if so, how? A boundedly rational approach to multi-armed bandits. *Journal of Economic Theory*, *78*(1), 130–156.

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). Piscataway: IEEE Press.

Shoham, Y., Powers, R., & Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, *171*(7), 365–377.

Sutton, R., & Barto, A. (1998). *Reinforcement learning, an introduction*. Cambridge: MIT Press.

Tan, M. (1993). Multi-agent reinforcement learning: independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning* (pp. 330–337).

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.

Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge: MIT Press.

Whitehead, S. D. (1991). A complexity analysis of cooperative mechanisms in reinforcement learning. In T. L. Dean & K. McKeown (Eds.), *Proceedings of the 9th national conference on artificial intelligence* (Vol. 2, pp. 607–613). Menlo Park/Cambridge: AAAI Press/MIT Press.