Multi-armed Bandit Formulation of the Task Partitioning Problem in Swarm Robotics

Giovanni Pini, Arne Brutschy, Gianpiero Francesca,
Marco Dorigo, and Mauro Birattari

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
{gpini,arne.brutschy,gianpiero.francesca,mdorigo,mbiro}@ulb.ac.be

Abstract. Task partitioning is a way of organizing work consisting in the decomposition of a task into smaller sub-tasks that can be tackled separately. Task partitioning can be beneficial in terms of reduction of physical interference, increase of efficiency, higher parallelism, and exploitation of specialization. However, task partitioning also entails costs in terms of coordination efforts and overheads that can reduce its benefits. It is therefore important to decide when to make use of task partitioning. In this paper we show that such a decision can be formulated as a multi-armed bandit problem. This is advantageous since the theoretical properties of the multi-armed bandit problem are well understood and several algorithms have been proposed for tackling it. We carry out our study in simulation, using a swarm robotics foraging scenario as a testbed. We test an ad-hoc algorithm and two algorithms proposed in the literature for multi-armed bandit problems. The results confirm that the problem of selecting whether to partition a task can be formulated as a multi-armed bandit problem and tackled with existing algorithms.

1 Introduction

Task partitioning refers to the act of dividing a task into a sequence of sub-tasks that can be tackled separately [9]. Many social insects, such as ants, bees, and wasps employ task partitioning for organizing their work. The benefits that insects draw from task partitioning are many: decrease of physical interference between individuals, higher exploitation of specialization, higher parallelism and efficiency in performing tasks [17]. Swarms of robots could benefit from task partitioning in the same ways. However, task partitioning also entails costs that are mainly a consequence of the coordination required to link different sub-tasks one to another. Therefore, task partitioning should be employed only when the benefits overcome the costs. In the rest of the paper, we will refer to the problem of selecting whether to employ task partitioning as the *task partitioning problem.*

In a previous work, we proposed a method that allows the robots to choose when to employ task partitioning, on the basis of the costs involved [16]. In this paper, we extend the work by reformulating the task partitioning problem as a multi-armed bandit problem [3]. The multi-armed bandit problem consists in repeatedly selecting actions to be performed in order to maximize a reward that depends on the actions taken. In terms of the multi-armed bandit, the task partitioning problem can be reformulated as the problem of choosing between partitioning the overall task and performing it as an unpartitioned task, with the goal of minimizing the resulting costs. Each robot tackles the multi-armed bandit problem independently of the other robots: it selects its actions on the basis of its individual estimate of the costs. The advantage of formulating the task partitioning problem as a multi-armed bandit stems from the fact that the latter is widely studied in statistics. Consequently, its theoretical properties are well understood and, most importantly, one can select among several existing algorithms, without the need of implementing ad-hoc solutions every time. The approach presented in this paper can be used to solve the task partitioning problem in situations in which the robots can measure or estimate the costs associated to employing task partitioning.

Multi-armed bandit problems are characterized by a tradeoff between *exploitation* and *exploration*. A balance has to be found between "exploring the environment to find profitable actions" [2] and "taking the empirically best action as often as possible" [2]. Also in the task partitioning problem there is such a tradeoff. Task partitioning should be exploited as much as possible, if the expected resulting costs are low. However, changes in the environment can affect costs. Therefore, the option of using task partitioning should be reconsidered in time, in order to detect such changes.

The rest of the paper is organized as follows. In Section 2 we review the existing work on task partitioning in swarm robotics. In Section 3 we describe the specific problem tackled in this work and present the three algorithms that we consider for tackling the problem. In Section 4 we briefly describe the experimental setup and the tools used to carry out the research. In Section 5 we present and comment the results of the experiments. Finally, in Section 6 we summarize the contribution of the work and we describe directions for future research.

2 Related Work

The biology literature is rich in studies devoted to task partitioning. In particular, task partitioning has been observed in social insects in the organization of tasks such as material transportation, nest excavation, and waste removal [17]. Swarm robotics draws inspiration from the world of social insects in the implementation of robotic systems composed of a large number of relatively simple cooperating robots [5]. The tasks performed by swarms of robots have often a counterpart in the world of social insects. As social insects draw benefits from task partitioning, it is interesting to study the application of task partitioning to swarms of robots performing similar tasks.

While in biology the body of literature on task partitioning is large, few works in swarm robotics have been devoted to this topic. In the majority of the works, the focus is on the use of task partitioning as a means for reducing physical interference. In [7], a foraging task is partitioned into sub-tasks developing in separate areas, each one assigned a-priori to a different robot. A similar work is presented in [14], with the difference that in this case the areas are not assigned statically and several robots can share the same working area. In [18], a swarm of robots has to forage for objects. Objects are progressively moved towards the nest by different robots, each working in an area of a given radius. The study shows that the higher the number of robots, the smaller the working area radius should be. The work as been extended to allow for a dynamical regulation of the working areas size [10] and to relocate the working areas depending on the objects distribution in the environment [11]. In [12], a swarm of robots has to perform foraging in an environment composed of several corridors. The authors show that task partitioning improves performance when the corridors are too narrow for two robots traveling in opposite directions to pass at the same time.

In a previous work we studied task partitioning in a foraging task and proposed a simple method that allows a swarm of robots to tackle the task partitioning problem [8]. In a follow up research the method has been extended to explicitly take into account costs linked to task-partitioning [16].

3 Problem Description and Methodology

We study the task partitioning problem: how to choose whether to partition a given task, or to perform it as a whole, unpartitioned task. When task partitioning is employed, the given task is partitioned into a sequence of sub-tasks. In this paper, we focus on the case in which there are two sub-tasks. The sub-tasks are linked by an interface of finite capacity. The output of the first sub-task can be stored at the interface and be subsequently used as the input for the second sub-task. In this paper we use a swarm robotics foraging scenario as testbed.

Figure 1 provides a schematic representation of the environment and the problem we study in this work. In the foraging scenario, the robots repeat an object retrieval task: harvesting an object from the *source*, and storing it at the *nest*. The environment is composed of two areas, one containing the source and the other containing the nest, separated by a *cache*. The robots cannot cross the cache, but they can use it to transfer objects from one area to the other. A *corridor* links the two areas and allows the robots to reach one from the other.

In the setup described, using the cache allows the robots to partition the object retrieval task into two sub-tasks: the first consists in harvesting an object from the source and drop it in the cache, the second in picking up an object from the cache and storing it at the nest. Therefore, the cache acts as an interface between sub-tasks. Conversely, the use of the corridor allows the robots to perform object retrieval as an unpartitioned task: a robot can directly reach the source from the nest and the other way around, harvesting and storing objects.

Each robot chooses whether to employ task partitioning in two situations, represented by a question mark in Fig. 1. First, after taking an object from the source, a robot decides whether to use the cache to DROP the object, or to use the corridor and STORE the object at the nest. Second, after storing an object in the nest, a robot decides whether to PICK UP an object from the cache, or to use the corridor and HARVEST an object from the source.

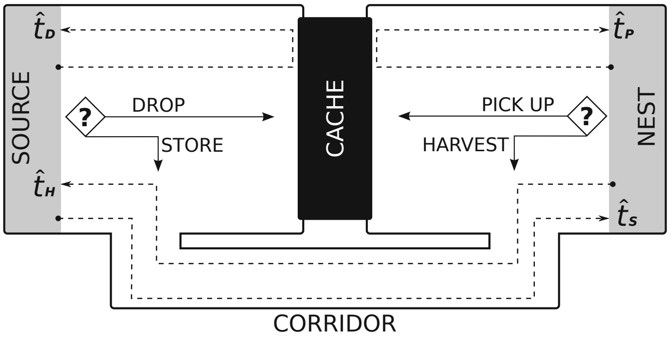

Fig. 1. Representation of the studied foraging problem. Foraging consists in harvesting objects from the source and storing them at the nest. Robots choose between using task partitioning (i.e., use the cache) or not (i.e., use the corridor) in two cases, marked with "?" in the figure. After taking an object from the source, a robot chooses between STORE it at the nest or DROP it at the cache. Upon storing an object in the nest, a robot chooses between PICK UP the next at the cache or HARVEST one from the source. The dashed arrows represent cost estimates $\hat{t}_i$ that the robot associates to each action.

In this work, we show that the task partitioning problem can be formulated as a multi-armed bandit problem. In the multi-armed bandit problem, the goal is to maximize a reward. If the dual problem of minimizing costs is tackled, algorithms and techniques for the bandit problem can be employed for the task partitioning problem as well. The nature of the costs depends on the specific task and on the characteristics of the environment. Typically costs are represented by resources needed to perform the task. Examples are: energy, time required to complete a task, or materials employed. In the foraging scenario studied in this paper, the goal is to maximize the number of objects delivered to the nest. This can be done by maximizing the throughput; we therefore express costs in terms of time.

Each robot keeps a cost estimate for each of the possible four actions: i) HARVEST an object from the source (using the corridor), ii) PICK UP an object from the cache, iii) DROP an object in the cache, and iv) STORE an object in the nest (using the corridor). Each estimate $\hat{t}_i$ is computed as:

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

where $\alpha \in (0,1]$ is a weight factor. t_M is the measure of the time taken by the last action performed by the robot, its meaning depending on the specific estimate being updated (refer to the dashed arrows in Fig. 1). When estimating the cost $\hat{t}_H$ of HARVESTING an object from the source, t_M measures the time from the moment an object is stored in the nest till the moment a new object is harvested from the source. Analogously, when estimating the cost $\hat{t}_S$ of STORING an object in the nest, t_M denotes the time measured from the moment an object is taken from the source to the moment it is deposited in the nest. When estimating the cost $\hat{t}_D$ of DROPPING objects in the cache, t_M measures the time from the moment an object is taken from the source, to the moment the following one is taken from the source, after dropping the first in the cache. Analogously, for the cost $\hat{t}_P$ of PICKING UP, t_M accounts for the time between two objects being stored in the nest, with the second one taken from the cache. These estimates are used by the robots to decide between using the cache or the corridor.

In this work we compare three algorithms, used by the robots to make this decision. The first is an *ad-hoc* algorithm that we proposed in a previous work [16]. Using the ad-hoc algorithm, after taking an object from the source, a robot has a probability P_p of DROPPING it in the cache:

$$P_p = \begin{cases} \left[1 + e^{-S\left(\left(\hat{t}_H + \hat{t}_S\right)/\left(\hat{t}_P + \hat{t}_D\right) - 1\right)}\right]^{-1}, & \text{if } \hat{t}_H + \hat{t}_S > (\hat{t}_P + \hat{t}_D) \\ \left[1 + e^{-S\left(1 - \left(\hat{t}_P + \hat{t}_D\right)/\left(\hat{t}_H + \hat{t}_S\right)\right)}\right]^{-1}, & \text{if } \hat{t}_H + \hat{t}_S \leq (\hat{t}_P + \hat{t}_D) \end{cases}, \tag{2}$$

where S is a steepness factor. The higher its value, the higher the degree of exploitation of the algorithm. Analogously, after delivering an object to the nest, a robot has the same probability P_p of PICKING UP the following one from the cache. Thus, the object retrieval task is performed as a partitioned task with a probability of P_p and performed as an unpartitioned task with a probability $1 - P_p$.

We compare the ad-hoc algorithm with two other algorithms that have been previously proposed in the literature to tackle multi-armed bandit problems. The first of the two, which we will refer to as *UCB*, is a modified version of the UCB1 policy presented in [2] that, in turn, is derived from the index-based policy described in [1]. Using UCB, after taking an object from the source, a robot DROPS it in the cache if:

$$\hat{t}_D - \gamma\sqrt{\frac{2\ln(n_D + n_S)}{n_D}} < \hat{t}_S - \gamma\sqrt{\frac{2\ln(n_D + n_S)}{n_S}}, \tag{3}$$

otherwise it takes the corridor to STORE the object in the nest. n_D is the number of times that the robot selected the cache for DROPPING an object, n_S the number of times the robot used the corridor for STORING an object in the nest. γ is a parameter that allows to tune the degree of exploration of the algorithm: the higher the value, the higher the exploration. An analogous formula is used to choose between PICKING UP an object from the cache or HARVEST it from the source using the corridor.

The third algorithm studied in this work, is the *ε-greedy* algorithm, a simple algorithm widely employed in reinforcement learning [19]. With the ε-greedy algorithm, the action perceived as the less costly is selected with a probability $1 - \varepsilon$, otherwise a random action is selected. ε defines the degree of exploration of the algorithm: the higher the value, the higher the exploration.

Notice the difference between the ad-hoc algorithm and the other two. In the former, no distinction is made between the two decision points: both at the nest and at the source there is the same probability P_p of employing task partitioning.

In the UCB and the ε-greedy algorithms, the robots discriminate between the two cases when making their choice.

For all the algorithms, a *give up* mechanism allows the robots to abandon the choice of using the cache. Without this mechanism, deadlocks could occur in two cases. The first case happens if all the robots are trying to drop objects in the cache and the cache is full. The second case happens if all the robots are trying to pick up objects from the cache, and the cache is empty. Giving up is implemented using a timeout: the robot measures the time it has been trying to access the cache, and abandons its choice when the time reaches a given threshold. Details about how the threshold is computed can be found in the online supplementary material [15]. When a robot gives up, its current waiting time updates the respective estimate $\hat{t}_D$ or $\hat{t}_P$ using Equation 1.

4 Experimental Setup

This section briefly describes the experimental tools and the environment in which we run the experiments presented in the paper. A more detailed description can be found in [16], of which the research presented here is a follow up.

All the experiments presented in this work have been carried out in simulation using *ARGoS* [13], a simulator developed within the *Swarmanoid*[1] project [6]. We simulate the *e-puck*[2], a small wheeled robot that has been used in many studies in swarm robotics. As the e-puck does not have the capability of grasping objects, we abstract this process by using a device called *Task Allocation Module (TAM)* [4]. Each TAM is a small booth in which an e-puck can enter. In the experiments, we simulate the TAM and its basic functionalities: two RGB LEDs that can be perceived by the e-pucks and a light barrier that can detect the presence of a robot within the TAM. In the experiments presented in this article, we implement the source, the nest, and the cache using TAMs.

The experiments take place in the environment represented in Fig. 2. The source is located at the top-left, the nest at the top-right, and the cache between the source and the nest. The nest and the source are implemented using four TAMs on one side; the cache is implemented with eight TAMs, four on each side, organized in pairs facing opposite directions[3]. The corridor links the areas containing the source and the nest.

We perform the experiments in two environments, called short-corridor and long-corridor environments. In both environments, the source is 1.5 m away from the nest. The environments differ in terms of the total length of the corridor: 5.0 m in the short-corridor and 7.5 m in the long-corridor environment. We also impose a cache processing time Π that the robots have to spend in the cache when dropping or picking up an object. The length of the corridor and the value of Π determine whether it is more advantageous to perform the object retrieval

[1] `http://www.swarmanoid.org`
[2] `http://www.e-puck.org`
[3] A video showing the behavior of the cache can be found in the online supplementary material at the following url: `http://iridia.ulb.ac.be/supp/IridiaSupp2012-005/`

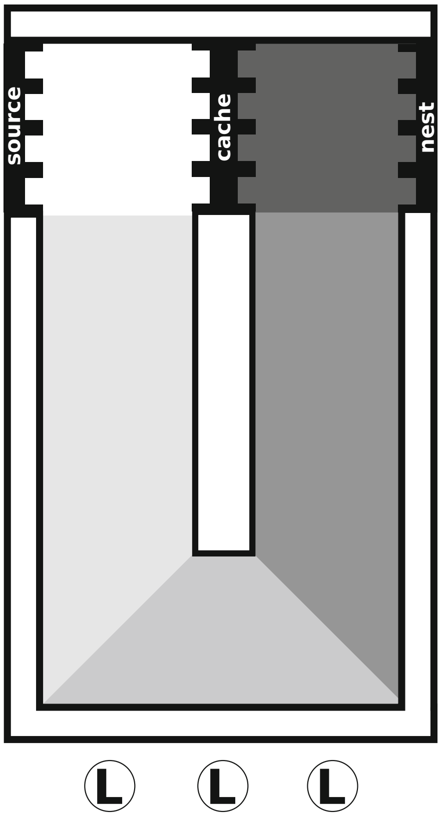

Fig. 2. Representation of the experimental environment. Nest, source and cache are implemented using TAMs. The different ground colors are used by the robots for localization in the arena. Light sources, marked with "L", provide directional information. The short-corridor and long-corridor environments differ in the total length of the corridor.

task as unpartitioned task, or to partition it into two sub-tasks. By changing the value of Π, we can tune the relation between the performance obtained by using the cache and the one obtained by using the corridor. Consequently, we can also define how advantageous it is to employ task partitioning.

5 Experiments and Results

We run all the experiments in both the short-corridor and long-corridor environments, with swarms of 10 and 20 robots. Every experiment lasts 10 simulated hours. At the beginning of each experiment, half of the swarm is positioned in the area containing the source and the other half in the area containing the nest. The value of α, used for computing the time estimates in Equation 1, has been set to 0.5. Notice that, in order to reduce the parameter space, we do not tune the value of α with systematic experiments. We select this value of α since low values are likely to render the algorithms poor in reacting to changes, while high values increase sensitivity to noise. The values of the cost estimates are initialized randomly: $\hat{t}_H$ and $\hat{t}_S$ are uniformly sampled in the interval $[40, 80]$, $\hat{t}_P$ and $\hat{t}_D$ in $[20, 40]$.

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

Algorithm	parameter	exploiting version	exploring version
ad-hoc	S	6.0	1.0
UCB	γ	100	1000
ε-greedy	ε	0.01	0.11

We run two sets of experiments. The goal of the first set of experiments is to select the parameters for each algorithm. Details about these experiments and the complete results can be found in the online supplementary material. Following these experiments, we selected two parameters settings for each algorithm, one corresponding to an *exploring* version and one to an *exploiting* version of the algorithm.

In the second set of experiments, the goal is to test whether, by employing the different algorithms, the robots are able to choose properly when to use task partitioning and when not to. Additionally, we test if the choice made by the robot adapts to variations occurring in the environment. In each experiment, the value of the cache interfacing time Π is initialized to 0, it switches to 160 s after 2.5 hours, and then it switches back to 0 when the experiment reaches half of its duration. The robots are expected to choose between the cache and the corridor and to adapt their choice in time.

Figure 3 reports the results of the experiments, for a swarm composed of 20 robots, in the long-corridor environment. Plots on the same row refer to the same algorithm (from top to bottom: ad-hoc, UCB, ε-greedy). The plots in the left column report the results for the exploiting version of the corresponding algorithm, the ones on the right for the exploring version. Each box reports the percentage of usage of the cache in the 30 minutes preceding the time reported on the X axis. The grey horizontal lines report the optimal cache usage, that changes depending on the value of Π. The grey slanted lines report percentages of cache usage that lead to a performance of at least 95% of the optimal. Performance is measured as average total number of objects retrieved at the end of the experiment. To determine the optimal way of using the cache, we performed experiments in which some of the robots were forced to always use the cache. For each value of Π and the two swarm sizes, we exhaustively tested all the possible values of the number of robots forced to use the cache and recorded the corresponding performance. The performance of the different algorithms is reported in Fig. 4. The optimal performance and the performance of an algorithm randomly selecting between cache and corridor are also reported for reference.

A comparison between the two versions of each algorithm highlights the trade-off between exploration and exploitation, typical in multi-armed bandit problems. When the exploiting versions of the algorithms are employed, the robots can select when to use the cache and when to use the corridor, but are unable to detect changes occurring in the environment. This can be seen in the near

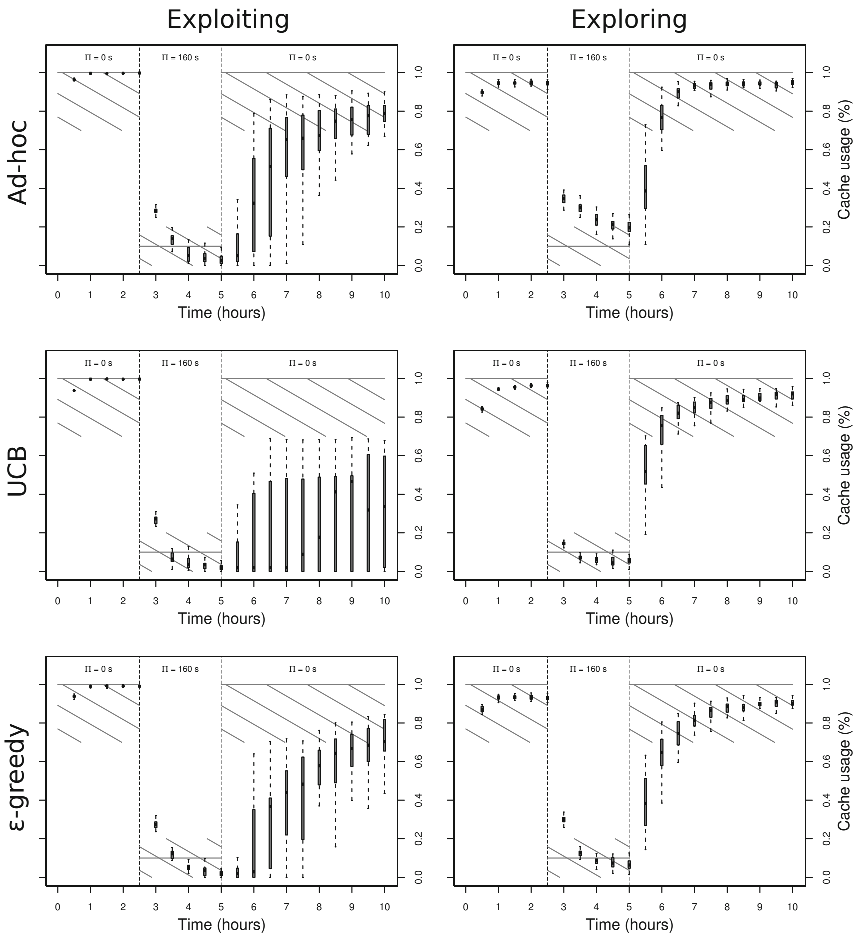

Fig. 3. Percentage of usage of the cache for the ad-hoc (first row), the UCB (second row), and the ε-greedy (third row) algorithm. In each row, the plot on the left reports the results for the exploiting version, the one on the right for the exploring version of the corresponding algorithm. We report the results obtained in the long-corridor environment, with a swarm composed of 20 robots. The cache interfacing time Π is initialized to 0. After 2.5 hours of experiment, the value is changed to 160 seconds, and returns to 0 at half experiment. Vertical dashed lines mark the moments in which the value of Π changes. Each box reports the percentage of usage (over 25 experimental runs) of the cache in the 30 minutes preceding the time reported on the X axis. The grey horizontal line reports the cache usage that maximizes the number of objects retrieved, which varies with the value of Π. The grey slanted lines report percentages of cache usage that lead to a number of objects retrieved that is at least 95% of the maximum.

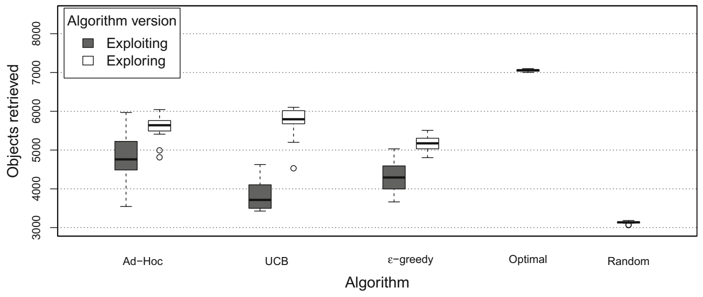

Fig. 4. Total number of objects retrieved by a swarm of 20 robots in the long-corridor environment

optimal behavior in the first half of the experiment, which degrades in the second half. Notice that the first change in the value of Π is detected also when employing the exploiting version of the algorithms. The reason is that initially the cache is selected often by the robots. Consequently, they can detect changes in Π independently of the version of the algorithm being employed. Detecting the opposite transition in the value of Π is harder and it only happens when the exploring version of the algorithms is employed.

The overall results (see online supplementary material) indicate that, in general, the algorithms perform better, more consistently, and with higher reactivity to changes, when the swarm is larger. This highlights that cooperation is required in order to render task partitioning effective. When the robots are many, it is more likely that robots are present on both sides of the cache, which is critical in order to exploit the cache properly. The length of the corridor seems to have little effect on the behavior of algorithms.

The results confirm that the task partitioning problem can be formulated as a multi-armed bandit problem. General algorithms for tackling bandit problems, such as the UCB and the ε-greedy, can be successfully employed to tackle the task partitioning problem, with results comparable with those of an ad-hoc algorithm. In particular the ε-greedy is a suitable candidate, since it is simple and its only parameter is easy to understand and tune manually.

6 Conclusions

In this paper, we studied the problem of choosing whether to tackle a task as a whole, or to partition it into a sequence of two sub-tasks. We show that the problem can be formulated as a multi-armed bandit problem. This is advantageous since the problem is well studied and understood, and its theoretical properties are known. Most importantly, several algorithms have been proposed in the literature for tackling the problem. This allows one to select an algorithm knowing its strengths and weaknesses and apply it to task partitioning problems without the need of designing ad-hoc solutions each time. The approach can be applied to situations in which costs can be measured or estimated by the robots. We pointed out that the tradeoff between exploration and exploitation, typical of multi-armed bandit problems, also arises in the task partitioning problem. This tradeoff has to be taken into account when choosing an algorithm and its parameters. Directions for future work aim at investigating more complex cases with more than two sub-tasks, as well as cases in which the location of the sub-tasks interface is not predefined, but must be decided by the robots autonomously. Additionally, in this work each robot tackles the task partitioning problem individually. As future work, we also plan to enhance the system with explicit communication. The robots could exchange information about the environment and compute cost estimates also on the basis of the information received.

Acknowledgements. The research leading to the results presented in this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 246939. Marco Dorigo, Mauro Birattari, and Arne Brutschy acknowledge support from the Belgian F.R.S.–FNRS. Giovanni Pini acknowledges support from Université Libre de Bruxelles through the "Fonds David & Alice Van Buuren".

References

1. Agrawal, R.: Sample mean based index policies with $O(\log n)$ regret for the multi-armed bandit problem. Advances in Applied Probability 27, 1054–1078 (1995)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2), 235–256 (2002)
3. Berry, D.A., Fristedt, B.: Bandit problems: Sequential allocation of experiments. Chapman & Hall, London (1985)
4. Brutschy, A., Pini, G., Baiboun, N., Decugnière, A., Birattari, M.: The IRIDIA TAM: A device for task abstraction for the e-puck robot. Tech. Rep. TR/IRIDIA/2010-015, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2010)
5. Dorigo, M., Şahin, E.: Guest editorial. Special Issue: Swarm robotics. Autonomous Robots 17(2-3), 111–113 (2004)
6. 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.: Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. IEEE Robotics & Automation Magazine (in press, 2012)
7. Fontan, M.S., Matarić, M.J.: A study of territoriality: The role of critical mass in adaptive task division. In: Maes, P., Matarić, M.J., Meyer, J.A., Pollack, J., Wilson, S. (eds.) From Animals to Animats 4: Proceedings of the Fourth International Conference of Simulation of Adaptive Behavior, pp. 553–561. MIT Press, Cambridge (1996)

8. Frison, M., Tran, N.-L., Baiboun, N., Brutschy, A., Pini, G., Roli, A., Dorigo, M., Birattari, M.: Self-organized Task Partitioning in a Swarm of Robots. In: Dorigo, M., Birattari, M., Di Caro, G.A., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L.M., Groß, R., Şahin, E., Sayama, H., Stützle, T. (eds.) ANTS 2010. LNCS, vol. 6234, pp. 287–298. Springer, Heidelberg (2010)
9. Jeanne, R.L.: The evolution of the organization of work in social insects. Monitore Zoologico Italiano 20, 119–133 (1986)
10. Lein, A., Vaughan, R.: Adaptive multi-robot bucket brigade foraging. In: Bullock, S., Noble, J., Watson, R., Bedau, M.A. (eds.) Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, pp. 337–342. MIT Press, Cambridge (2008)
11. Lein, A., Vaughan, R.T.: 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 2009), pp. 601–606. IEEE Press, Piscataway (2009)
12. Østergaard, E.H., Sukhatme, G.S., Matarić, M.J.: Emergent bucket brigading: A simple mechanisms for improving performance in multi-robot constrained-space foraging tasks. In: AGENTS 2001: Proceedings of the Fifth International Conference on Autonomous Agents, pp. 29–30. ACM Press, New York (2001)
13. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G.A., Ducatelle, F., Stirling, T., Gutiérrez, A., Gambardella, L.M., Dorigo, M.: ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In: Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011), pp. 5027–5034. IEEE Computer Society Press, Los Alamitos (2011)
14. Pini, G., Brutschy, A., Birattari, M., Dorigo, M.: Task Partitioning in Swarms of Robots: Reducing Performance Losses Due to Interference at Shared Resources. In: Cetto, J.A., Filipe, J., Ferrier, J.-L. (eds.) Informatics in Control Automation and Robotics. LNEE, vol. 85, pp. 217–228. Springer, Heidelberg (2011)
15. Pini, G., Brutschy, A., Francesca, G., Dorigo, M., Birattari, M.: Multi-armed bandit formulation of the task partitioning problem in swarm robotics – Online supplementary material (2012), `http://iridia.ulb.ac.be/supp/IridiaSupp2012-005/`
16. Pini, G., Brutschy, A., Frison, M., Roli, A., Birattari, M., Dorigo, M.: Task partitioning in swarms of robots: An adaptive method for strategy selection. Swarm Intelligence 5(3–4), 283–304 (2011)
17. Ratnieks, F.L.W., Anderson, C.: Task partitioning in insect societies. Insectes Sociaux 46(2), 95–108 (1999)
18. Shell, D.J., Matarić, M.J.: 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. IEEE Press, Pitscataway (2006)
19. Sutton, R., Barto, A.: Reinforcement learning, an introduction. MIT Press, Cambridge (1998)