UNIVERSITÉ LIBRE DE BRUXELLES
Ecole Polytechnique de Bruxelles
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Towards Autonomous Task Partitioning in Swarm Robotics

Experiments with Foraging Robots

Giovanni PINI

Promoteur de Thèse:
Prof. **Marco DORIGO**

Co-Promoteur de Thèse:
Prof. **Mauro BIRATTARI**

Thèse présentée en vue de l'obtention du titre de
Docteur en Sciences de l'Ingénieur

Année académique 2012-2013

Giovanni Pini

Towards Autonomous Task Partitioning in Swarm Robotics

Experiments with Foraging Robots

Université Libre de Bruxelles

This dissertation has been submitted in partial fulfillment of the requirements to obtain the doctoral degree at Université Libre de Bruxelles. This dissertation has not been previously submitted to Université Libre de Bruxelles nor any other university or organization. The work that is presented in this dissertation is based upon a number of publications of the author. Short quotations from the dissertation are permitted, provided that the original source is acknowledged.

To my family.

Acknowledgements

I joined IRIDIA in 2006, and since then I met people of what I consider to be three different generations of IRIDIANs. In time, people joined and left the lab, but IRIDIA never really changed: it is a big family where everyone is welcome since the first day. I am glad to be part of this family and in the future I will remember with pleasure the time I spent and the people I met at IRIDIA.

I would like to thank Prof. Marco Dorigo first. He is the one who gave me the opportunity to join IRIDIA and to make such an experience. I also thank him for his supervision and for being always available to give his help and advice. Thanks to Mauro, who supervised me in my everyday research and taught me a lot. I had several discussions with Mauro, many times my mood deteriorated after the discussion since they meant more work to be done, but often his suggestions were very helpful. I also thank Elio for his supervision during the first year I spent at IRIDIA. Together with Marco, Elio was the one who made possible my experience at IRIDIA.

Special thanks to Arne, since it was him who freed me from the hand-bot's grip. In a word, Arne is simply amazing; the work presented in this thesis could not have been carried out without his precious help. I want to thank Carlo for always being available to help people and for letting me complete the Ph.D. before him. Thanks to my office-mates Manu, Franco, and Tarik, for rendering my everyday work so funny. Thanks to "Kardesh" Ali and Eliseo, two of the nicest guys I have ever met. Thanks to Nithin for being always ready to listen and for his always-positive attitude. Thanks to Alessandro for his humor and for sharing his passion for Swiss ornithology with all of us. Thanks to Jeremie for keeping the cluster (almost) always up and running, which allowed me to run the many experiments presented in the thesis. Thanks to Prasanna for the great fun we always had together.

Thanks to the new IRIDIANs: Gabriele, Touraj, Lorenzo, Leslie, Giovanni, Dhananjay, Gianpiero, Anthony, Leonardo, Gaëtan, Roman. Us "oldies" are leaving IRIDIA in good hands.

Special (re-)mention for the heroes of Lausanne: Manu, Arne, Ali's evil twin, Alessandro, Eliseo, and Nithin. We managed to do an amazing job and to have some (sort of) fun despite the far-from-optimal conditions. Also thanks to Mr. O. and Mr. P. Faislosli for their logistic support during the aforementioned work.

Thanks to those who "volunteered" to proof read this thesis and (I hope) found all the remaining little errors: Arne, Dhananjay, Lorenzo, Manuele, Eliseo, Carlo, Giovanni, Gabriele, Gianpiero, Touraj, and Nithin. I hope I will soon return the favor to each of you. Thanks to Rachael for her tips on English grammar.

For reasons of space I cannot mention everybody, but I have fond memories of each person I met during my stay at IRIDIA (few exceptions); many many thanks to everybody.

Among the non-IRIDIANs I thank Lluis and Soschanne for being such nice friends and for the good time we always spend together.

Many thanks to my family for their love, help, and support. Without them I would never have come this far in my life.

Thanks to Silvia for her love and for always encouraging me when I most need it.

To conclude I also thank my friends back in Italy: Luca, Della, Teo, Mauri, Cippi, Davids, Dagno, Rudy. Each time I come back for a visit you make me feel as if I never left.

Abstract

In this thesis, we propose an approach to achieve autonomous task partitioning in swarms of robots. Task partitioning is the process by which tasks are decomposed into sub-tasks and it is often an advantageous way of organizing work in groups of individuals. Therefore, it is interesting to study its application to swarm robotics, in which groups of robots are deployed to collectively carry out a mission. The capability of partitioning tasks autonomously can enhance the flexibility of swarm robotics systems because the robots can adapt the way they decompose and perform their work depending on specific environmental conditions and goals. So far, few studies have been presented on the topic of task partitioning in the context of swarm robotics. Additionally, in all the existing studies, there is no separation between the task partitioning methods and the behavior of the robots and often task partitioning relies on characteristics of the environments in which the robots operate. This limits the applicability of these methods to the specific contexts for which they have been built. The work presented in this thesis represents the first steps towards a general framework for autonomous task partitioning in swarms of robots. We study task partitioning in foraging, since foraging abstracts practical real-world problems. The approach we propose in this thesis is therefore studied in experiments in which the goal is to achieve autonomous task partitioning in foraging. However, in the proposed approach, the task partitioning process relies upon general, task-independent concepts and we are therefore confident that it is applicable in other contexts. We identify two main capabilities that the robots should have: i) being capable of selecting whether to employ task partitioning and ii) defining the sub-tasks of a given task. We propose and study algorithms that endow a swarm of robots with these capabilities.

Contents

1 Introduction **1**
- 1.1 Contributions and Related Publications 3
- 1.2 Other Scientific Contributions 5
- 1.3 Structure of the Dissertation 8

2 Context and Related Work **11**
- 2.1 Swarm Intelligence and Swarm Robotics 11
  - 2.1.1 Swarm Intelligence and Self-organization 12
  - 2.1.2 Swarm Robotics 15
- 2.2 Task Partitioning 17
  - 2.2.1 The Benefits of Task Partitioning 17
  - 2.2.2 Task Partitioning in Artificial Systems 20
- 2.3 Task Allocation 23
- 2.4 Foraging 24

3 The Approach **27**
- 3.1 Tasks, Sub-tasks, and Task Partitioning 28
- 3.2 Sub-tasks, Amount of Work, and Interfaces 31
- 3.3 Strategy 36
- 3.4 Autonomous Task Partitioning in Swarms of Robots . . 39
- 3.5 Application to Other Domains 42
  - 3.5.1 A Construction Scenario 42
  - 3.5.2 A Tasty Scenario 43
- 3.6 Summary 44

4 Tools **47**
- 4.1 The e-puck and the TAM 47
- 4.2 The MarXbot 49
- 4.3 ARGoS 53

5 Deciding Whether to Use a Fixed Interface **57**
- 5.1 Description of the Problem 58
- 5.2 Experimental Setup 61

5.3 Application of the Proposed Approach 65
5.4 The Ad Hoc Algorithm 68
5.4.1 The Algorithm 68
5.4.2 Experiments and Results 69
Performance Evaluation 70
Adaptivity to Changes 74
Scalability 77
5.5 Task Partitioning as a Bandit Problem 81
5.5.1 Studied Algorithms 82
5.5.2 Experiments and Results 85
Stationary Environmental Conditions 87
Non-stationary Environmental Conditions . . . 88
5.6 The Use of Communication 91
5.6.1 The Communication Protocol 92
5.6.2 Experiments and Results 92
The Effect of Communication 93
Algorithms with ε-exploration 95
5.7 Summary 96

6 Amount of Work Contributed by a Sub-task 99
6.1 Localization 100
6.2 Problem Description 102
6.3 Application of the Proposed Approach 105
The Model of the Cost Function 106
Estimation of the Costs 109
6.3.1 Task Partitioning Algorithms 112
The Cost-based Partitioning Algorithm 112
The Fixed Algorithms 112
The Random Initialization Algorithm 112
6.4 Experimental Setup 113
6.4.1 Experimental Environment 113
6.4.2 Behavior and Characteristics of the MarXbot . 115
6.4.3 Simulation of the System Using ARGoS 117
6.5 Validation of the System 121
6.6 Simulation Experiments and Results 126
6.6.1 Basic Properties 127
6.6.2 Size of the Environment 135
6.6.3 Distance to the Source 136
6.6.4 Heterogeneity in the Robot Swarm 138
6.6.5 Adaptivity to Variable Conditions 140
6.7 Summary 145

7 Conclusions **147**
7.1 Contributions 147
7.2 Future Work 148

Annexes **151**

A ARGoS 1 Versus ARGoS 2 **153**
A.1 Implementation Differences 153
A.2 Published Experiments and Results 155
A.2.1 Performance Evaluation 156
A.2.2 Adaptivity to Changes 159
A.2.3 Scalability 160

B Supplementary Material **163**

Bibliography **165**

Chapter 1

Introduction

Robots are machines built to replace human work in a variety of situations. There exist several types of robots; in this dissertation, we focus on systems composed of a large number of autonomous mobile robots. As the name suggests, mobile robots are robots that can freely navigate in the environment.

The nature of the work to be performed and of the environment in which it must be performed defines which type of robotic system should be employed and what its properties should be. For example, assembling a car requires a well-defined series of operations to be executed in a structured environment. In this situation, industrial manipulators are an optimal choice. These robots are not required to cope with uncertainty nor to be particularly flexible in their behavior, but rather to be fast and accurate in their movements.

Mobile robots, on the other hand, are suited for other types of tasks, such as exploration, search and rescue, cleaning, mine clearance, and collection of materials. In these contexts, using a multitude of robots is usually preferable to using a unique robot that performs all the work, since parallelism and fault tolerance increase. Additionally, the tasks mentioned are typically carried out in non-structured and potentially unknown environments which may change in time. This usually requires the robots to be (at least partially) autonomous, so that their behavior is flexible and it can dynamically adapt to a variety of environmental conditions.

Designing and implementing a robotic system composed of a large number of autonomous robots is a complex problem. Typically, in non-trivial situations, the designer of the system decomposes the work that the robots have to accomplish into a set of separate units. He implements solutions so that the robots can execute these units. The designer then composes these solutions to obtain a system capable of

performing the overall work. Isolating units of work that must be performed by the robots eases the job of the designer, since he can devise solutions for smaller and more manageable problems. Moreover, in the case of multi-robot systems, the separation into units of work usually results in a better exploitation of the parallel nature of such systems. However, there is a disadvantage associated with this approach: the way the overall work is decomposed, and consequently executed by the robots, is a choice made at design time and the system lacks flexibility with respect to this choice.

The research presented in this dissertation is motivated by the pursuit of flexibility and autonomy at the level of the definition of the units of work performed by the robots. Our vision is that the role of the designer of a robotic system should be to equip the robots with a set of minimal capabilities, while the decomposition of the overall work into smaller units should be a process carried out by the robots autonomously. The process by which work is decomposed into smaller units to be tackled separately is called *task partitioning*. Robotics systems capable of performing task partitioning autonomously would be extremely flexible with respect to unforeseen situations and dynamically changing environments: not only their behavior, but also the way the work is organized could be adapted to specific conditions. The research presented in this dissertation represents a step towards robotics systems with such a capability. We propose an approach that can be applied in the context of swarm robotics to let the robots of the swarm autonomously partition tasks into sequences of sub-tasks.

Other research work exists on the topic of task partitioning in swarm robotics. The task partitioning mechanisms and algorithms proposed so far have two characteristics in common. First, the task partitioning decisions are implemented upon specific behavioral traits of the robots and rely on known characteristics of the environment in which the robots operate. Second, task partitioning explicitly aims to reduce physical interference between the robots. We believe these characteristics limit the reusability of the proposed task partitioning mechanisms and result in a poor flexibility of the robots behavior. The approach presented in this dissertation aims to overcome these limitations. Within our approach, the task partitioning decisions do not depend on specific behaviors of the robots nor characteristics of the environment. In addition, task partitioning does not explicitly aim to reduce physical interference. These aspects render our approach more general with respect to what proposed so far.

We design our approach to obtain task partitioning in the specific context of foraging, since it is an abstraction of many practical prob-

lems, such as, for example, search and rescue, mine clearance, and cleaning. However, in our approach, task partitioning is built upon general, context-independent concepts and we are therefore confident that the approach can be directly applied to contexts other than foraging.

The contents of this dissertation are based upon a number of research articles that we published. In Section 1.1, we list these articles and highlight their contributions to the contents of this dissertation. In Section 1.2, we present other scientific contributions of ours that are not strictly related to the topic of task partitioning and therefore do not directly contribute to this dissertation.

1.1 Contributions and Related Publications

In our study of task partitioning, we identified two situations the robots may encounter. In one situation, due to constraints of the environment or the task, the robots are free to decide whether or not to partition the given task, but they are not free to autonomously decide what these sub-tasks consist in. In the other situation, the robots are free to define the sub-tasks of the given task. In our research, we study the two situations and propose algorithms and methods that enable the robots to tackle them.

In Chapter 5 of the dissertation, we concentrate on the first situation. We propose algorithms that swarms of robots can use to autonomously decide whether to partition a given task into sub-tasks. The contents of Chapter 5 are based upon four articles, listed in the following. The first algorithm that we developed has been published in:

- Frison M., Tran N.-L., Baiboun N., Brutschy A., **Pini G.**, Roli A., Dorigo M., and Birattari M. Self-organized task partitioning in a swarm of robots. *Proceedings of the* 7^{th} *International Conference on Swarm Intelligence (ANTS 2010).* Volume 6234 of LNCS, pages 287-298. Springer, Berlin, Germany, 2010.

The same experimental setup was used to design an extended version of the algorithm so that decisions are made by the robots on the basis of cost estimates. We performed experiments to assess the capability of the algorithm to properly select whether to employ task partitioning and to adapt the choice to variations occurring in the environment. The results of the research work have been published in:

- **Pini G.**, Brutschy A., Frison M., Roli A., Dorigo M., and Birattari M. Task partitioning in swarms of robots: An adaptive

method for strategy selection. *Swarm Intelligence*, 5(3-4):283-304, 2011.

In a follow-up work, we have shown that the problem of deciding whether to employ task partitioning can be seen as a multi-armed bandit problem. This is advantageous since one can select among the many algorithms that have been proposed to tackle this problem. We compare two of such algorithms to the ad-hoc algorithm we proposed in our previous work and show that these algorithms can be successfully employed in the context of task partitioning. The results of the research have been published in:

- **Pini G.**, Brutschy A., Francesca G., Dorigo M., and Birattari M. Multi-armed bandit formulation of the task partitioning problem in swarm robotics. *Proceedings of the* 8^{th} *International Conference on Swarm Intelligence (ANTS 2012).* Volume 7461 of LNCS, pages 109-120. Springer, Berlin, Germany, 2012.

The work was further extended to study the effect of communication on the system. An article, presenting the results of the study, has been accepted for publication in an international journal:

- **Pini G.**, Gagliolo M., Brutschy A., Dorigo M., and Birattari M. Task partitioning in a robot swarm: A study on the effect of communication. *Swarm Intelligence*, 2013. In press.[1]

In Chapter 6 of the dissertation, we study the case in which the robots are free to autonomously define the sub-tasks of a given task. The contents of the chapter are based upon two research articles.

In a first research work, we study the cost of performing object transportation as a partitioned task, with the robots directly handing over objects one to another. The work has been submitted for publication to an international journal:

- **Pini G.**, Brutschy A., Scheidler A., Dorigo M., and Birattari M. Task partitioning in a robot swarm: Object retrieval as a sequence of sub-tasks with direct object transfer.[2]

A similar case-study was used to develop and test a method that enables a swarm of robots to autonomously partition the transportation of objects on the basis of the perceived costs. An article presenting this research work has been published in:

[1] An Online First version of the article is available. DOI: 10.1007/s11721-013-0078-7.
[2] The work is currently available as a technical report (Pini et al., 2012b).

- **Pini G.**, Brutschy A., Pinciroli C., Dorigo M., and Birattari M. Autonomous task partitioning in robot foraging: An approach based on cost estimation. *Adaptive Behavior*, 21(2):117-135, 2013.

In addition to the works mentioned so far, we performed further research on the topic of task partitioning whose results are not included in this dissertation. The goal of this research was to study the use of task partitioning to reduce physical interference between robots. This research has been published in:

- **Pini G.**, Brutschy A., Birattari M., and Dorigo M. Interference reduction through task partitioning in a robotic swarm. *Proceedings of the* 6^{th} *International Conference on Informatics in Control, Automation and Robotics (ICINCO 2009)*, CD-ROM, 2009.

and, as an extended version, in:

- **Pini G.**, Brutschy A., Birattari M., and Dorigo M. Task partitioning in swarms of robots: Reducing performance losses due to interference at shared resources. *Lecture Notes in Electrical Engineering (LNEE)*, 85, 217-228, 2011.

1.2 Other Scientific Contributions

In this section, we present scientific contributions of ours that are not related to task partitioning. A first set of published contributions is within the topic of task allocation. Task allocation is the problem of assigning a number of tasks to individuals of a group. Task allocation complements task partitioning: after a task has been partitioned into sub-tasks, individuals must be assigned to the sub-tasks.

A first subject that we studied within the context of task allocation is behavioral specialization. Behavioral specialization occurs when "*individuals adapt their behavior so that they predominantly work on a subset of the available task types*" (Brutschy et al., 2012c). Behavioral specialization entails benefits, for example due to learning, but it may also be costly, since individuals may have to spend time searching for tasks in which they are specialized. Our research work on specialization studied its costs and benefits in the case in which the robots are subject to learning. The research work has been published in:

- Brutschy A., Tran N.-L., Baiboun N., Frison M., **Pini G.**, Roli A., Dorigo M., and Birattari M. Costs and benefits of behavioral specialization. *Proceedings of the* 12^{th} *Conference Towards Autonomous Robotic Systems (TAROS 2011)*, 90-101. Springer, Berlin, Germany, 2011.

- Brutschy A., Tran N.-L., Baiboun N., Frison M., **Pini G.**, Roli A., Dorigo M., and Birattari M. Costs and benefits of behavioral specialization. *Robotics and Autonomous Systems*, 60(11):1408-1420, 2012.

In the context of task allocation, we also proposed a self-organized method for allocating robots to tasks that exhibit a sequential interdependency, that is, tasks that must be executed in a given order. This type of dependency is often found in sub-tasks resulting from partitioning a task. The method is decentralized and it is based on measures of the delay that each robot experiences when waiting for the input needed to execute a task. We performed experiments that demonstrate that the method allocates the robots in a near-optimal fashion and that it is able to adapt the allocation to changes occurring in the environment. The research work has been accepted for publication in an international journal:

- Brutschy A., **Pini G.**, Pinciroli C., Birattari M., and Dorigo M. Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems*, 2012. [3]

Another research topic that we studied is evolutionary robotics. Evolutionary robotics is an automated approach to the implementation of robotic systems. The approach consists in controlling the robots with neural networks that are automatically selected through an evolutionary process by a genetic algorithm. For more information about evolutionary robotics refer to Nolfi and Floreano (2000).

Our research on this topic focused on the evolution of social learning. Social learning is the form of learning that occurs in an individual due to interactions with other individuals, for example via imitation. We studied a setup in which the goal of the robots is to learn whether to approach a light source or move away from a it (i.e., phototaxis or antiphototaxis). We synthesized a controller that enables a robot to learn both from environmental cues and socially, from the interaction with another robot. The results of the research have been published in:

- **Pini G.**, Tuci, E, and Dorigo, M. Evolution of social and individual learning in autonomous robots. *Proceedings of the* 1^{st} *Workshop on Social Learning in Embodied Agents (SLEA)*, CD-ROM, 2007.

[3] An Online First version of the article is available. DOI: 10.1007/s10458-012-9212-y.

- **Pini G.**, and Tuci, E. On the design of neuro-controllers for individual and social learning behaviour in autonomous robots: An evolutionary approach. *Connection Science Journal*, 20(2-3):211-230, 2008.

In addition to the research work on the topics of task allocation and evolutionary robotics, we published a series of contributions within the *Swarmanoid* project.[4] Swarmanoid was a Future and Emerging Technologies (FET) project funded by the European Union. The scientific goal of the project was "*the design, implementation and control of a novel distributed robotic system [...] made up of heterogeneous, dynamically connected, small autonomous robots.*" The project successfully ended in 2010.

The scientific goals of the project, the implementation of the Swarmanoid robots and simulation software, and the results of a search and retrieval demonstration scenario have been presented in:

- 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., Di Caro G., Ducatelle F., Ferrante E., Fröster A., Martinez Gonzales J., Guzzi J., Longchamp V., Magnenat S., Mathews N., Montes de Oca 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., and Vaussard F. Swarmanoid: A novel concept for the Study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*. In press.

Additionally, a video illustrating the search and retrieval scenario has been published in the AAAI 25^{th} Conference on Artificial Intelligence, winning the best video award of the AI video competition 2011:

- Dorigo M., Birattari M., O'Grady R., Gambardella L. M., Mondada F., Floreano D., Nolfi S., Baaboura T., Bonani M., Brambilla M., Brutschy A., Burnier D., Campo A., Christensen A. L., Decugnière A., Di Caro G., Ducatelle F., Ferrante E., Martinez Gonzales J., Guzzi J., Longchamp V., Magnenat S., Mathews N., Montes de Oca M., Pinciroli C., **Pini G.**, Rétornaz P., Rey F., Roberts J., Rochat F., Sperati V., Stirling T., Stranieri A., Stützle T., Trianni V., Tuci E., Turgut A. E., and Vaussard F. Swarmanoid, the movie. 25^{th} *Conference on Artificial Intelligence (AAAI-11)*, AI video competition 2011. San Francisco, CA. Winner of the best video award.

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

The project also led to the development of *ARGoS*, a simulation software that we utilized to perform the simulation-based experiments presented in this dissertation. Two articles, describing ARGoS and its features, have been published in:

- 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., and Dorigo M. ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. *Swarm Intelligence*, 6(4):271-295, 2012.
- Pinciroli C., Trianni V., O'Grady R., **Pini G.**, Brutschy A., Brambilla M., Mathews N., Ferrante E., Di Caro G., Ducatelle F., Stirling T., Gutiérrez A., Gambardella L. M., and Dorigo M. ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, 5027-5034. IEEE Computer Society Press, Los Alamitos, CA, 2011.

1.3 Structure of the Dissertation

The contents of this dissertation are organized as follows. In Chapter 2, we describe the principles and properties at the basis of swarm intelligence and swarm robotics. We introduce task partitioning, illustrate its benefits through examples taken from nature, and review related work on the topic, focusing on artificial systems.

In Chapter 3, we illustrate task partitioning as studied in this dissertation and present our approach for autonomous task partitioning. We provide general definitions and concepts that are used throughout this dissertation and we describe the core principles at the basis of our approach.

In Chapter 4, we give an overview of the hardware and software tools we used to perform the research work presented in this dissertation. The chapter is not meant to provide an exhaustive description of these tools, but rather to introduce the features of these tools that are relevant to our experiments.

In Chapter 5, we present a scenario in which the robots must decide whether to employ task partitioning or not. In the studied setup, the given task is pre-partitioned into a sequence of two sub-tasks, defined a priori, and the robots decide whether or not to perform the two subtasks separately. We propose and test a set of algorithms that can be employed by the robots to make the decision.

In Chapter 6, we illustrate a different scenario, in which the robots are free to autonomously define sub-tasks of the given task (i.e., the sub-tasks are not defined a priori). We present a method that the robots can use to autonomously partition tasks into sub-tasks and we show that the partitioning solution adopted by the robots suits the environmental conditions.

In Chapter 7, we summarize the contents and the contributions of this dissertation and we discuss possible directions for future research.

In Annex A, we report the original results that were published in Pini et al. (2011) and compare them to the results presented in Section 5.4.2. The results presented in Pini et al. (2011) were obtained with an older version of the simulator. For a matter of coherence of the dissertation, we repeated the same experiments with the simulator version that we used for the rest of the experiments presented in this dissertation.

A CD-ROM, containing supplementary material not included in this dissertation for reasons of space, is included as an annex. The same material is available online at the address `http://iridia.ulb.ac.be/supp/IridiaSupp2013-001`.

Chapter 2

Context and Related Work

In this chapter, we describe the research context of the work presented in this dissertation and discuss related studies. In Section 2.1 we present swarm intelligence and swarm robotics. We illustrate the principles and the properties that are the basis of both studies. In Section 2.2, we focus on task partitioning, the main subject of this dissertation: we highlight its benefits and review the state of the art on the topic. Task partitioning is strongly related to task allocation; in Section 2.3, we discuss differences and relations between the two topics and we provide pointers to relevant swarm robotics studies on task allocation. Finally, Section 2.4 is dedicated to foraging, since we developed and tested our approach for autonomous task partitioning in the context of foraging in swarms of robots.

2.1 Swarm Intelligence and Swarm Robotics

Progress in mechatronics over the last few decades, has led to an increase in applications of robotic systems. Robots replace humans in activities that are dangerous, repetitive, or that require high precision, strength, or a high execution speed. While there are many types of robots, our research focuses on autonomous mobile robots, which are machines that can move and perform actions in an environment without the need of direct human control. Recently, there has been a growing interest in multi-robot systems, in which a number of robots perform tasks cooperatively. In multi-robot systems, centralized control paradigms often hit some limits, typically computational or communicational and therefore other approaches must be employed. To tackle this issue, roboticists often resorted to swarm intelligence, giving birth to the field known as swarm robotics.

The rest of this section is organized as follows. Section 2.1.1 is

dedicated to swarm intelligence and self-organization. With the help of examples, we illustrate what swarm intelligence is and describe its core principles. In Section 2.1.2, we introduce swarm robotics. We also focus on the properties that make it appealing for the implementation and control of multi-robot systems and we discuss its potential applications.

2.1.1 Swarm Intelligence and Self-organization

The term "swarm intelligence" was first introduced in Beni and Wang (1989) to refer to

> "systems of non-intelligent robots exhibiting collectively intelligent behaviour".

The initial definition was specific to robotic systems, but it was later generalized to systems composed of other types of artificial agents or living beings. Each of the individuals is depicted as "non-intelligent", but, as mentioned later on, swarm intelligence can be observed also in intelligent individuals, such as human beings (see examples in Helbing et al. (2001)). The core characteristic of swarm intelligence is that the intelligence of the collectivity exceeds the one of the individuals, producing *"results that are "improbable" so are in some way surprising or unexpected"* (Beni and Wang, 1989) considered the (relative) simplicity of each individual.

In nature, swarm intelligence can be observed in a variety of systems (see Figure 2.1).[1] The coordinated motion of flocks of birds or schools of fishes, for example, is a form of swarm intelligence. Each individual bird or fish acts on its own and, most importantly, has no control over the others nor is aware of what the group as a whole is doing. Despite this, the motion of the group and its cohesion are such that one might think of it as a single entity, controlled by a unique brain.

The most astonishing examples of swarm intelligence systems are found in social insects, such as ants, termites, bees, and wasps. A termite nest, for instance, is one of the most complex structures built by an animal. A single nest can be taller than a man and house millions of individuals. The nest is composed of several compartments, that serve as food storage areas, fungus gardens, queen chambers, and nurseries. In addition, ventilation shafts maintain the internal temperature constant regardless the climatic conditions. Termites build such complex

[1] Sources: (Top-left) CC BY-NC-ND 2.0 - monkeywithastungun (flickr username) - http://www.flickr.com/photos/23645016@N00/4363399006 (Top-right) CC BY-NC-SA 2.0 - Lance McCord - http://www.flickr.com/photos/mccord/41614967 (Bottom-left) CC BY-NC-SA 2.0 - Stephen Lowe - http://www.flickr.com/photos/stephenlowe/289197489 (Bottom-right) CC BY - James Cridland (flickr username) - http://www.flickr.com/photos/jamescridland/613445810

Figure 2.1: Examples of swarm intelligence systems. Top-left: A flock of birds. Top-right: A school of fishes. Bottom-left: A termite nest. Bottom-right: A crowd.

structures without following a specific plan nor having an architect leading the work. Each worker makes decisions on its own and yet the colony is able to create a structure that is functional to its survival.

Crowds can also exhibit swarm intelligence behaviors. Examples can be observed in the movements of pedestrians. Normally, each individual moves freely towards its destination, but in conditions of high-density, movements are constrained and patterns often emerge. For example, when pedestrians in a crowd move in opposing directions, different lanes of people walking in a common direction form spontaneously (Helbing et al., 2001). This happens without the need of explicit negotiation or coordination and without predefined rules. This example highlights that not only non-intelligent individuals, but also individuals with high cognitive skills, such as humans, may resort to swarm intelligence.

What these examples have in common is that none of the individuals is aware of the overall behavior of the group, nor there is a leader or supervisor that decides for the whole group. Instead, each individual decides autonomously and acts to pursue its own goals. The actions of locally interacting individuals lead to collective behaviors that seem to

be driven by a unique will and controlled by a single entity. This form of collective behaviors are called *self-organized* behaviors. Bonabeau et al. (1999) define self-organization as a *"set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components"*. These interactions are *"executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system"*. Garnier et al. (2007) identify four basic elements of self-organization:

1. **Positive feedback**: results from the application of simple rules and promotes the creation of structures;
2. **Negative feedback**: balances positive feedback and contributes to the stabilization of the system;
3. **Fluctuations**: random behaviors and errors that allow the creation of structures and render the system more flexible;
4. **Multiple interactions**: lead to the appearance of global patterns.

These four elements can be observed, for instance, in the foraging activity of ants. Ants are capable of collectively selecting the shortest path that leads to a food source (Goss et al., 1989). When presented with different paths to reach a destination, the ants prefer those with higher concentrations of pheromone. While it is following a path, an ant may lay pheromone and reinforce an existing pheromone trail on the same path. This has a positive feedback effect: the pheromone trail is strengthened and more ants will follow it, resulting in an amplification effect. Negative feedback occurs in terms of evaporation of the pheromone, interference, and exhaustion of the food source. Fluctuations are due to ants deviating from a selected path. These ants may discover a better food source and establish a pheromone trail that leads to it. In ant foraging, interactions are mediated by the environment: the ants can influence the behavior of each other through the pheromone-laying mechanism. The mix of the four components results in a self-organized foraging behavior that allows the ants to exploit efficiently the available food sources and to discover new ones.

The interest in swarm intelligence is not only towards the study of its theoretical properties, but also towards its application to real-world problems (Dorigo and Birattari, 2007). Usually, systems built following swarm intelligence principles provide suboptimal but acceptable solutions for problems in which optimal solutions are not known or cannot

be applied, for example due to limited resources such as computational time.

Implementing artificial swam intelligence systems is a challenging task, since the designer has to focus on the implementation and control of individual agents[2] with the ultimate goal of obtaining a certain behavior at the collective level. This collective behavior is the result of many local and often stochastic interactions between the agents and between the agents and their environment. These interactions and their effects are difficult to predict, specially if the environment is not fully known.

Typically, swarm intelligence systems are built by trial and error or by taking inspiration from natural systems. Examples of the latter approach are the ant colony optimization technique, inspired by the foraging behavior of ant colonies (Dorigo and Stützle, 2004), clustering algorithms inspired by the cemetery construction and brood sorting in species of ants (Handl et al., 2006; Lumer and Faieta, 1994), the PSO[3] optimization technique inspired by bird flocking and fish schooling behaviors (Kennedy and Eberhart, 1995), and task allocation methods based on models of division of labor in social insects (Campos et al., 2000).

2.1.2 Swarm Robotics

In many situations, a system composed of a multitude of cooperating robots is preferable to one in which a single robot performs all the tasks. This is the case, for example, when the tasks are too complex for a single robot to accomplish, or when producing a number of simple robots is cheaper than producing a single robot capable of performing all the tasks (Cao et al., 1997). However, as the number of robots increases, also the complexity of the system does. Relying on centralized control and global communication becomes unfeasible and one has to resort to other approaches.

An approach that has been applied is to implement and control robotic systems by taking inspiration from natural swarm intelligence systems. The application of swarm intelligence principles to collective robotics gave birth to *swarm robotics*. There is no precise definition to identify a swarm robotics system in the more general context of collective robotics systems. Dorigo and Şahin (2004) propose four criteria to measure *"the degree to which the term swarm robotics might apply"*:

[2]The term "agent" refers in this context to any entity (physical or virtual) that can perform actions in an environment.

[3]Particle swarm optimization.

1. The implemented system should allow the coordination of a large number of robots;
2. The system should be composed of relatively few homogeneous groups of robots (i.e., heterogeneous systems are less "swarm robotics");
3. The robots of the system have difficulties in carrying out tasks on their own, collaboration is needed to improve performance;
4. The robots should have limited sensing and communication capabilities.

Therefore, a swarm robotics system is one in which a large number of relatively simple robots cooperate to solve tasks that cannot be tackled efficiently by single individuals. Each robot acts autonomously on the basis of its local perception of the surrounding environment. Interactions and communication between robots are also local and are often mediated by the environment.

Swarm robotics systems have many desirable properties. The large number of robots provides redundancy, which protects the robotic system against faults: if some robots fail to perform their tasks or get damaged, the rest of the swarm can continue to perform the tasks. Decentralization also contributes to enhance fault tolerance: contrary to a centralized system, in a swarm there is no single element that, in case of failure, would compromise the entire system. The use of local information, interactions and communication usually renders swarm robotics systems scalable: robots can be added to the swarm without the need of redesigning the control and communication strategies. Finally, since the robots are relatively simple, the production costs are lower compared to the production costs of complex robots and this renders the robots more expendable.

To date, swarm robotics systems have been implemented only in research laboratories and no real-world application is based on swarm robotics. However, the field of autonomous robotics is growing and technological progress is making robots pervade our lives; examples are vacuum cleaning robots (Rooks, 2001), lawn mowers (Hicks and Hall, 2000), and the Kiva robotics warehouse system.[4] Due to their properties, swarm robotics systems are suited for applications that require redundancy and/or scalability, that entail danger, and that are performed in unstructured and dynamical environments or in large spaces (Şahin, 2005; Brambilla et al., 2013). When the technology matures, swarm

[4] `http://www.kivasystems.com`

robotics will eventually be applied to domains with these characteristics, such as exploration, surveillance, mine clearance, cleaning, search and rescue, and military applications (Brambilla et al., 2013).

2.2 Task Partitioning

Task partitioning is a technique for organizing the work of groups of individuals that consists in decomposing a task into a number of sub-tasks (Jeanne, 1986). Decomposition allows tackling different sub-tasks in parallel, or in different moments in time. Task partitioning is a technique that can be applied to a multitude of tasks and in different contexts. There are several potential benefits and therefore task partitioning is an option to consider when organizing the execution of tasks.

Task partitioning has been studied mainly by entomologists and the biology literature is rich with research articles describing task partitioning in social insects. In Section 2.2.1 we refer to some of these studies and use them to illustrate the benefits of task partitioning. A complete review of the biology literature is beyond the scope of this dissertation. Comprehensive reviews can be found in the works of Ratnieks and Anderson (1999) and Hart et al. (2002). In the rest of this dissertation however, we sometimes borrow from biology to provide examples and to illustrate general principles and concepts related to task partitioning. Contrary to biology, there are few works in the literature describing task partitioning applied to artificial domains. In Section 2.2.2 we review such works, focusing on studies in swarm robotics.

2.2.1 The Benefits of Task Partitioning

Several examples of task partitioning can be observed in nature. The most evident example are humans, that exploit the advantages of task partitioning both at the individual and at the societal level. At the level of the individual, a natural way of tackling a non-trivial and lengthy task is to decompose it into less complex sub-tasks. At the level of the society, many activities and interactions are so complex that in order to be manageable, they must be simplified by partitioning them. For example, large companies are organized in divisions that are managed separately.

Task partitioning can also be observed in simpler forms of life, such as social insects. Examples of task partitioning are reported in the foraging activity of species of wasps (Jeanne, 1991), bees (Seeley, 1995), and ants (Hubbell et al., 1980). Other activities in which task parti-

tioning is observed are hunting, waste removal, and nest excavation (see Ratnieks and Anderson, 1999, for details). Swarms of robots are very similar to colonies of social insects. Often swarm robotics systems are built drawing inspiration from the behavior and the morphology of social insects. This is largely due to the fact that the tasks performed by swarms of robots frequently have a counterpart in the world of social insects. Examples are: transportation of objects, exploration, formation of structures, construction. Since, in many cases, insects benefit from task partitioning when performing their activities, it is likely that swarms of robots performing similar activities can draw the very same benefits. These benefits are several and they are best illustrated through examples.

Reduction of physical interference. In many cases, task partitioning is beneficial to reduce physical interference between workers. Physical interference results from the competition for space when two or more individuals are at the same place at the same time (Goldberg, 2001). As the density of workers grows, the per-individual performance decreases due to physical interference (Lerman and Galstyan, 2002). Often partitioning a task generates a number of sub-tasks developing in physically separated areas. When this happens, workers can be separated as well, thus reducing physical interference.

For example, some species of *Atta* ants employ task partitioning when foraging for leaves as a response to an increased physical interference. Hart and Ratnieks (2000) observe that when the leaves are not processed fast enough at the nest, a blockage of leaves and foragers form at the entrance of the nest. Foragers react to this blockage depositing their leaves in piles outside the nest. Other ants fetch leaves from the piles and store them inside the nest. Task partitioning allows the foragers to return to their activity, while the bottleneck at the nest is relieved by other ants.

Physical interference is an issue also in swarm robotics, since it negatively affects the performance and the scalability of a system. With the exception of our contributions, all the research works on the topic of task partitioning in swarm robotics study its application as a mean to reduce physical interference (see Section 2.2.2).

Efficiency gains. There are situations in which partitioning a task into sub-tasks increases efficiency. For example, Fowler and Robinson (1979) describe how the *Atta sexdens* ant forages leaves on trees. This species partitions the foraging task into sub-tasks. Some individuals work at the top of the tree, cutting leaves and dropping them to the

ground. Other individuals collect the leaves from the ground and cut them into pieces that are transported to the nest. Here, the advantage of task partitioning stems from the fact that the ants do not need to repeatedly climb the tree. As a result, the energy efficiency of the swarm is increased.

Other examples in which task partitioning enhances efficiency are reported in the works of Hubbell et al. (1980) and Lopes et al. (2003), both studying foraging in ants. The authors report that the ants discovering new food sources do not transport food all the way back to the nest, but transfer it to other workers along the way and return to the food source. In this way, the ants can quickly establish a pheromone trail that leads to the food source and that can be followed by other ants. This results in a fast and efficient exploitation of the food source.

Task partitioning can result in efficiency gains also in swarms of robots. In Chapter 6 we describe an actual example of swarm robotics system in which the swarm efficiency is increased by the usage of task partitioning.

Exploitation of specialization. Task partitioning may also enhance the exploitation of specialization. This happens when different parts of a task require specific skills in order to be performed optimally. In some cases it is possible to isolate these parts into separate sub-tasks. Workers can then be allocated to sub-tasks on the basis of their capabilities. The workers can be more or less suited for certain sub-tasks because of innate differences (e.g., morphological) or because of learning.

For example in the leaf-cutting ant *Atta laevigata* large ants climb the plant stems, cut leaves at their petioles, and drop them to the ground. Smaller ants cut the lamina of the dropped leaves and transport leaf fragments to the nest. Partitioning the leaf foraging task into sub-tasks allows this ant species to allocate individuals to sub-tasks on the basis of their size. The size-dependent task allocation reflects the fact that petioles are tougher than lamina and can be more easily cut by larger individuals (Vasconcelos and Cherrett, 1996).

An analogous example, reported by Arnan et al. (2011), describes the behavior of the seed-harvesting ant *Messor bouvieri*: small individuals transfer harvested seeds to larger ones, that transport them to the nest. The authors show that smaller individuals are better (i.e., faster) at finding seeds, while larger individuals are better at transporting them. Partitioning the seeds foraging task allows each category of ant to perform the activity for which it is best suited.

One can easily imagine the very same situation occurring in a swarm

of heterogeneous robots, in which large robots with high load capacities are dedicated to the transport of items collected by smaller and more agile robots.

The benefits of task partitioning do not come for free: task partitioning requires additional coordination between individuals and therefore overheads are an inevitable cost to pay. The nature of these overheads is discussed in Chapter 3. However, the fact that many examples of task partitioning can be observed in nature suggests that the benefits of using task partitioning often overcome its costs.

2.2.2 Task Partitioning in Artificial Systems

In the previous section, we highlighted the benefits of task partitioning, providing examples mainly taken from the world of social insects. Task partitioning can also be beneficial if applied in artificial contexts. In this section, we describe artificial systems that utilize task partitioning and provide a complete review of the state of the art in the field of swarm robotics.

Perhaps the most evident example of task partitioning applied in an artificial context is an assembly line. In an assembly line the production of a good is executed as a sequence of separate steps often performed by different workers or machines. Here, benefits of task partitioning such as efficiency and exploitation of specialization are pushed to the extreme.[5]

The task partitioning principle is also applied in computer science. In computer systems with multiple processors, programs to be executed can be divided into separate units that are subsequently executed in parallel by different processors (Ennals et al., 2005). In a similar way, an operating system executes processes by allocating each process to a CPU for a limited time (Bovet and Cesati, 2005). The result is that processes are executed concurrently, each as a sequence of separate steps. Task partitioning is also the idea at the basis of recursive algorithms that implement the *divide and conquer* paradigm: a problem is recursively decomposed into smaller instances, whose solutions are then recombined (Cormen et al., 2001).

To date, in the context of swarm robotics, task partitioning has been exclusively applied in foraging. Foraging involves searching and collecting items in an environment. Many studies consider *central place foraging* (Orians and Pearson, 1979), in which the items to be collected must be delivered to a unique location, usually called *nest*. Physical

[5] Usually, the boredom of the workers is pushed to the extreme as well.

interference is a common problem in foraging: the robots share the same space and must avoid each other while moving in the environment. All the existing studies of task partitioning applied to swarm robotics systems use it as a means for reducing physical interference.

Drogoul and Ferber (1992) were the first to propose a solution based on task partitioning to deal with "traffic jams" forming in certain locations of the environment. In their study, the robots are allowed to pick up objects carried by other robots. The result is that chains of robots form in the environment and objects are transferred along these chains till they reach the nest. The goal of the work is to highlight how the global behavior of the swarm can be affected by variations in the rules governing the behavior of each individual. The role of task partitioning is only marginal and the authors ignore some important aspects of the problem, in particular how object transfer is actually implemented and what are the costs involved.

Fontan and Matarić (1996) tackle physical interference using an approach based on territoriality: the space is divided into exclusive areas (territories) each assigned a priori to a robot. Each area is associated to a sub-task to be performed: transporting the objects located in that area towards the nest. When a robot moving towards the nest exits its working area, it releases the object and it returns within its area boundaries. Proceeding in this way, the objects are progressively transported from area to area till they eventually reach the nest.

Goldberg and Matarić (2002) use a modified version of the same robotics system to study the design of behavior-based controllers that are robust with respect to failures and easy to modify. The authors propose different control strategies to deal with physical interference at the nest. One of these strategies uses task partitioning: one of the robots works in an exclusive area in proximity of the nest and is in charge of storing objects delivered nearby by the other robots.

The studies of Fontan and Matarić (1996) and of Goldberg and Matarić (2002), together with the work presented in Chapter 6, are the only works on task partitioning that include experiments performed with real robotic platforms.

Shell and Matarić (2006) introduce a novelty with respect to the work of Fontan and Matarić (1996). In the work of Fontan and Matarić (1996), the position of the 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 the robots estimate the position of the working areas using odometry, the working areas drift in time due

to estimation errors. Each robot transports the objects that it finds in its working area towards the nest, without leaving its working area. The robot releases objects at the boundary of its working area, where they are eventually collected by another robot. The authors show that the higher the density of robots, the smaller the optimal size of the working area.

Lein and Vaughan (2008) extend the system of Shell and Matarić (2006) with a mechanism that dynamically adapts the size of the working areas. Each robot constantly increases the size of its working area and decreases it when another robot is perceived nearby. This simple behavior improves performance over a method with static working area sizes, since it is adaptive with respect to the density of robots in the environment.

In a follow-up work, the same authors point out that the foraging method based on task partitioning is sensitive to the distribution of objects in the environment (Lein and Vaughan, 2009). In particular, if the objects are grouped into clusters, the method based on task partitioning performs worse than a non-partitioning method. Therefore, the authors further extended the algorithm proposed in Lein and Vaughan (2008) with a mechanism that relocates the working areas towards clusters of objects.

In the work of Østergaard et al. (2001), a group of robots forages in a maze-like environment. The authors compare an algorithm that uses task partitioning to a non-partitioning algorithm, in environments that differ in the width of the corridors. The authors conclude that task partitioning performs better in spatially constrained environments. It is worth pointing out that the authors mention that the driving factor that motivates the use of the task partitioning algorithm is not physical interference, but the type of environment. However, the results of their study confirm the findings of other studies: for an increasing robot density, task partitioning becomes preferable. This indicates that physical interference among the robots plays an important role in selecting whether or not to use task partitioning. The different environments simply determine the level of physical interference experienced by the robots.

Parker and Zhang (2010) study the case in which a group of robots must perform a sequence of mutually exclusive tasks: a task should begin only when the preceding one in the sequence is completed and no robot is working on it anymore. Analogous situations may occur when a task is partitioned into a sequence of sub-tasks. The focus of the study is on the decision making process that allows the robots to collectively estimate whether a sub-task is complete and the group can

start working on the following one.

2.3 Task Allocation

In this section, we illustrate relations and differences between task partitioning and task allocation. Additionally, we provide pointers to research work on task allocation that allow us to illustrate the main approaches to task allocation in swarm robotics.

Task allocation and *task partitioning* are both mechanisms that can be utilized for the organization of work, but they answer different needs and act at different levels in organizing work. On the one hand, task partitioning defines the work itself: by decomposing a task into sub-tasks it determines which actions contribute to which sub-task. Task allocation, on the other hand, is used to decide which individuals perform which task. Therefore, task allocation organizes the workforce and has no impact on what actions are part of the work to be performed. The relation between the two mechanisms stems from the fact that, once a task has been partitioned into sub-tasks, a task allocation problem arises: the sub-tasks must be executed and therefore the decision about which individuals execute which sub-task must be taken.

In the context of collective robotics, task allocation methods can be divided into explicit and implicit. In explicit methods, the allocation of robots to tasks is obtained through coordination and explicit communication. Implicit task allocation methods, on the other hand, are based on swarm intelligence principles and offer decentralized and often stochastic solutions to the task allocation problem. Implicit methods are more suited to swarm robotics, where the capabilities of the individuals are limited and information is local and often noisy. Kalra and Martinoli (2006) point out that explicit methods are computationally and communicationally expensive and when information is inaccurate, similar performance can be obtained, at a lower cost, with implicit methods.

Most task allocation methods proposed in the swarm robotics literature are based on the response threshold model, first introduced by Bonabeau et al. (1996) to model the division of labor observed in *Pheidole* ants. In these methods, a robot is "stimulated" by the perception of a task to be performed: the longer the task is perceived, the more the stimulus grows. The robot engages in the task depending on the current value of the stimulus and a threshold associated to that task. In simple cases, the thresholds are static, as in the work of Krieger and Billeter (2000); in other cases, the thresholds are modified in time, as in the works of Labella et al. (2006) and Campo and Dorigo

(2007).

Dahl et al. (2009) propose a different approach to implicit task allocation, inspired by vacancy chains, a mechanism by which resources are distributed to consumers. Task allocation is obtained through a distributed reinforcement learning mechanism: each robot estimates "utilities" associated to a task, computed on the basis of rewards and punishments. The utilities are used by the robots to select which task to perform. In a recent work, we propose an implicit method to allocate robots to tasks that have a sequential dependency (see Brutschy et al., 2012b). The method is based on the delays experienced by robots waiting for input to perform their task.

As mentioned, partitioning a task into sub-tasks introduces a task allocation problem: the individuals must be assigned to each sub-task. In general, there is a way of allocating individuals to sub-tasks that maximizes the overall performance. In the work presented in this dissertation, we do not explicitly tackle the task allocation problem associated to task partitioning. In our experiments, the allocation of robots to sub-task is implicitly defined within the mechanisms that implement task partitioning. This is likely to result in a suboptimal performance, which could be improved by explicitly tackling the task allocation problem with dedicated algorithms. Among the others, the task allocation method proposed in Brutschy et al. (2012b) is particularly relevant since it allocates individuals to tasks exhibiting a sequential dependency, which is the type of dependency that we consider in this dissertation.

2.4 Foraging

In this dissertation, we focus our study on the application of task partitioning in the context of foraging. Foraging is an activity consisting in searching and collecting items in an environment. There exist many typologies of foraging problems, that vary in relation to factors such as the characteristics of the robots, the number of source areas and collection points, and the type of items to be collected. In this dissertation, we consider cases in which the items must be delivered to a unique location, referred to as *nest*. This form of foraging is usually referred to as *central place foraging* (Orians and Pearson, 1979). For simplicity, in this dissertation we use the more general term *foraging* implying central place foraging. We study situations in which the items to be collected can be found by the robots in a unique location of the environment, referred to as *source*. Therefore, in the studied foraging problems, there is a unique collection point (the source) and unique

destination point (the nest).

Foraging has been widely studied in collective robotics mainly because it is an abstraction of real-world applications such as search and rescue, harvesting, exploration, cleaning, and mine clearance (Winfield, 2009). Indeed one of the reasons for which we study foraging is its relevance for such applications. Additionally, our interest in foraging is also motivated by the characteristics of the robots available in our laboratory. Among the types of robots available, three are mobile platforms: the e-puck (Mondada et al., 2009), the s-bot (Mondada et al., 2004), and the marXbot (Bonani et al., 2010). The s-bot and the marXbot additionally have the capability of grasping and transporting objects. The three platforms are suited for the real-world applications mentioned and therefore foraging scenarios are natural testbeds for our robots.

Given the fact that we focus on foraging, we developed our approach with such a context in mind. However, task partitioning entails general benefits, such as reduction of physical interference, increased parallelism, efficiency gains, and enhanced exploitation of specialization (see Section 2.2.1); these benefits do not depend on specific applications. Moreover, foraging shares a number of aspects with other swarm robotics applications; for example, the need of coordinating large groups of individuals and the use of robots with limited sensing, actuation, and computation capabilities. Due to these reasons, we built our task partitioning approach on general concepts so that it can be applied to other problem instances without radical modifications.

Chapter 3

The Approach

In this chapter, we present the approach towards achieving autonomous task partitioning in foraging. Definitions and concepts used in the rest of this dissertation are introduced and the general principles at the basis of our approach are stated.

In the existing work on foraging in swarm robotics, task partitioning is used to reduce physical interference. The mechanisms that implement task partitioning are often blended with the behavior of the robots and explicitly aim to reduce physical interference.

In the existing approaches we see two limitations. The first limitation is that there is no separation between the task partitioning process and the behavior of the robots. The task partitioning mechanisms are built within the behavior of the robots and they heavily depend upon characteristics of the environment and the tasks to be tackled. If any of these elements changes, the mechanisms upon which task partitioning is based are likely to require modifications.

The second limitation is that the task partitioning mechanisms are built with the explicit purpose of reducing physical interference among the robots. In Section 2.2.1 we point out that there are also other factors that motivate the use of task partitioning (e.g., the desire to increase the exploitation of specialization or efficiency). Additionally, task partitioning entails overhead costs. Therefore, using a task partitioning mechanism built to reduce interference in contexts in which interference is not the key issue, is likely to introduce costs without entailing benefits. This has a negative impact on performance, as indeed observed in many studies (see, for example Østergaard et al., 2001; Shell and Matarić, 2006).

We aim to overcome these limitations. Our goals are: i) decoupling the task partitioning process from the behavior of the robots, ii) implementing task partitioning mechanisms that are not explicitly built to

obtain a specific effect (i.e., reducing interference), and iii) proposing an approach that can be applied to tasks other than foraging.

To achieve these goals, we base our approach upon the two general concepts of *amount of work* and *cost*. The amount of work is used to express the size of a task or a sub-task. The costs express the quantity of resources that are utilized to perform a certain task or sub-task. In a nutshell, our approach to autonomous task partitioning consists in the robots deciding the amount of work they contribute to a task on the basis of cost estimates. We designed our approach to obtain autonomous task partitioning in foraging and therefore we only applied it to this context. However, we are confident that the same approach can also be applied to other contexts, once the definition of *amount of work* and *cost* have been redefined to suit the specific situation (see examples in Section 3.5). The decisional mechanisms that use costs and amount of work to implement task partitioning do not need to be modified and they can be directly used on top of the behavioral repertoire of the robots.

The rest of this chapter is organized as follows. In Section 3.1, we introduce the task partitioning problem as studied in this dissertation. In Section 3.2, we illustrate the relation between the amount of work and the size of a sub-task and we discuss the possible ways in which sub-tasks may interface with one another. In Section 3.3, we explain the concept of cost and we discuss the effects of task partitioning on costs. In Section 3.4, we describe the role of all these elements within our approach to autonomous task partitioning. Finally, in Section 3.5, we briefly consider two examples that illustrate that the concepts at the basis of our approach can be identified also in contexts different than foraging.

3.1 Tasks, Sub-tasks, and Task Partitioning

In Chapter 2 we defined task partitioning as a way to organize work that consists in dividing tasks into smaller sub-tasks. In this dissertation, we use the terms "task" and "sub-task" as they have been defined by Anderson and Franks (2001): a **task** is

> "an item of work that contributes potentially to fitness."
> (Anderson and Franks (2001))

This definition comes from biology and refers to animal societies (e.g., colonies of social insects) but it applies to swarms of robots and single individuals.

The core aspect of the definition of task is that, once the work is completed, there is a benefit for the swarm (or the individual). This distinguishes a task from a **sub-task**, that instead

> "makes a potential fitness contribution only if other sub-tasks are completed" (Anderson and Franks (2001))

which implies a dependency between sub-tasks.

In foraging, the sub-tasks exhibit a *sequential* dependency: all the sub-tasks must be executed in a certain order to complete the overall task. In other cases, not considered in this dissertation, a task may be partitioned into sub-tasks that do not have a sequential dependency. For example the task of cleaning a building can be partitioned in sub-tasks, each consisting in cleaning a different room. Usually the rooms can be cleaned in any order, therefore the sub-tasks do not have a sequential dependency.

The problem of partitioning a task into a sequence of sub-tasks shares similarities with planning, a problem studied in classical artificial intelligence. Planning consists in identifying a sequence of actions to be executed that lead from a start to a goal state. Analogously, in task partitioning, the execution of a sequence of sub-tasks leads to the completion of the overall task. In planning algorithms, the sequence of actions is built by reasoning on the effects of an action executed in a given system state (Russel and Norvig, 2009). The effectiveness of these algorithms therefore depends on whether the action effects can be predicted accurately. In swarm robotics applications, the environment in which the robots operate is often not completely known. Moreover, several robots act at the same time and modify the environment. Therefore, the uncertainty about the effects of the actions is very high and consequently existing planning algorithms cannot be applied to task partitioning in swarm robotics.

In general, task partitioning can be applied to a sub-task to further divide it into smaller units. Therefore, in the context of this dissertation, **task partitioning** is defined as:

> ***a mechanism to organize work that consists in dividing a task or a sub-task into a sequence of sub-tasks***

Figure 3.1 illustrates task partitioning in the context of foraging, as studied in this dissertation. In the figure, we represent both the robots performing foraging and the schematic notation that we use throughout the dissertation. The goal of the robots (Figure 3.1a) is to collect objects from the source (position X) and deliver them to the nest (position Y). In this context, the *task* is object transportation: collecting

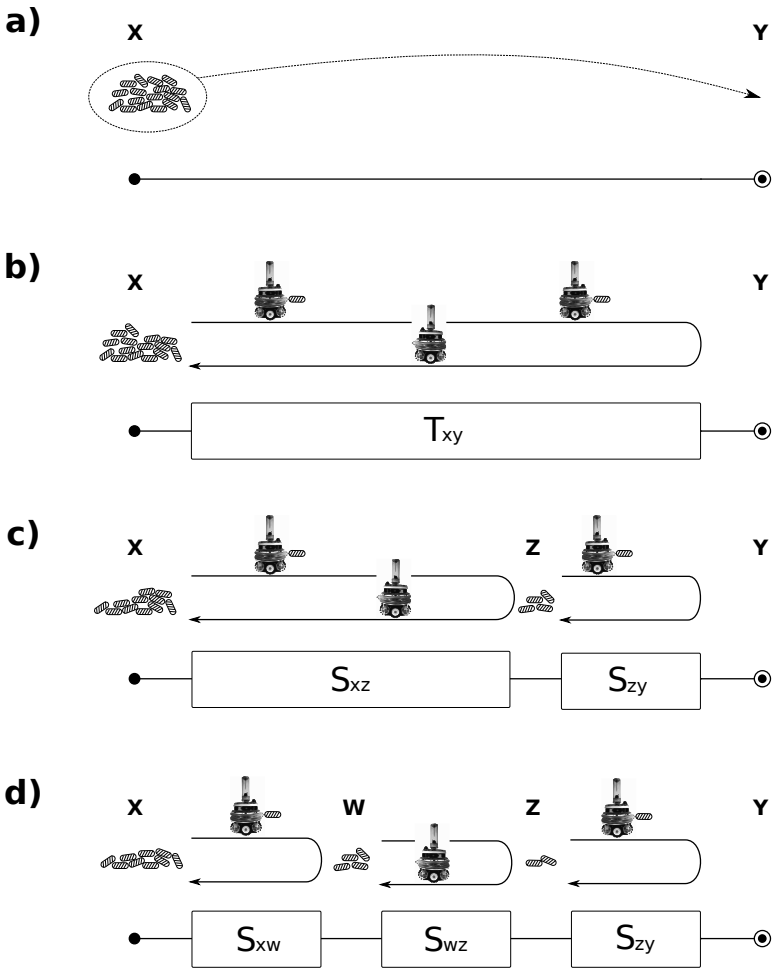

Figure 3.1: Representation of task partitioning in foraging. **(a)** Foraging is the repetition of the transportation task: transport an object from X to Y. **(b)** Execution of transportation as an unpartitioned task T_{xy}. **(c)** The task T_{xy} is partitioned into two sub-tasks S_{xz} and S_{zy}. **(d)** The sub-task S_{xz} is further partitioned into two sub-tasks S_{xw} and S_{wz}.

one object from the source and delivering it to the nest (i.e., transport an object from X to Y). Foraging consists in the repetition of different instances of the transportation task. Note that, since the swarm is composed of many robots, at a given moment multiple instances of the transportation task may be in execution. Using our notation, a task to be performed is represented with a line. The beginning of the task is represented with a black dot (left-hand side of Figure 3.1a); its end with a dot within a circle (right-hand side of Figure 3.1a). The curved arrow underneath the robots indicates the direction of motion: the

robots carrying objects move from left to right (i.e., towards the destination point Y), the ones not carrying objects move in the opposite direction.

Figure 3.1b illustrates the case in which the robots perform transportation as an unpartitioned task: each robot transports objects from X to Y. Using the schematic notation, we represent the execution of work with a rectangle, on top of the line representing the task. The rectangle is labeled with a name that can be used to refer to the corresponding task (T_{xy}). Figure 3.1c represents the case in which the robots partition the transportation task into the sequence of two subtasks S_{xz} and S_{zy}. In foraging, a *sub-task* consists in transporting one object for a limited distance towards the nest. In the example, S_{xz} consists in transporting an object from X to Z and S_{zy} in transporting an object from Z to Y. The sequential dependency between the sub-tasks resides in the fact that the robot executing S_{zy} can transport an object from Z to Y only if an object is first delivered in Z by another robot (i.e., an instance of S_{xz} must be performed before an instance of S_{zy}). In this case, the rectangles that represent the execution of work (Figure 3.1c, bottom part) are separated, indicating that the work is performed within different sub-tasks. Figure 3.1d shows that, in general, a sub-task can be further partitioned: the robots partition the sub-task S_{xz} into a sequence of two sub-tasks, S_{xw} and S_{wz}. In all the cases represented in Figure 3.1, the transportation of an object is completed (i.e., the swarm obtains a benefit) only when that object reaches the nest. This requires, in the cases in which transportation is partitioned into sub-tasks, that all the sub-tasks are executed once in the correct order.

3.2 Sub-tasks, Amount of Work, and Interfaces

When a task or a sub-task must be performed, one can measure the *amount of work* required for its execution. The concept of amount of work is a cornerstone of our approach to autonomous task partitioning. The amount of work relates to the actions that must be performed in order to complete a task or a sub-task and it expresses the dimension of a task or sub-task.

The specific way in which the amount of work is measured depends on the context. For example, if the task is executing a program on a computer, the amount of work can be measured as the number of instructions executed by the computer to run the program. In foraging, tasks and sub-tasks involve object transportation; therefore the amount of work can be directly associated to the distance traveled by the robots.

For this reason, in this dissertation we sometimes refer to the *length* of a task (or a sub-task) to indicate the amount of work required for its execution.

Interfaces. Another important concept for our approach is the one of *interface*. When a task is partitioned into a sequence of N sub-tasks, $N-1$ interfaces can be identified. An interface is an abstract notion, that indicates the linking point between two sub-tasks in the sequence: at the interface, the output of the first sub-task is used as input for the second. In the specific case of foraging, the interface between two sub-tasks is associated to a spatial *location* between source and nest.

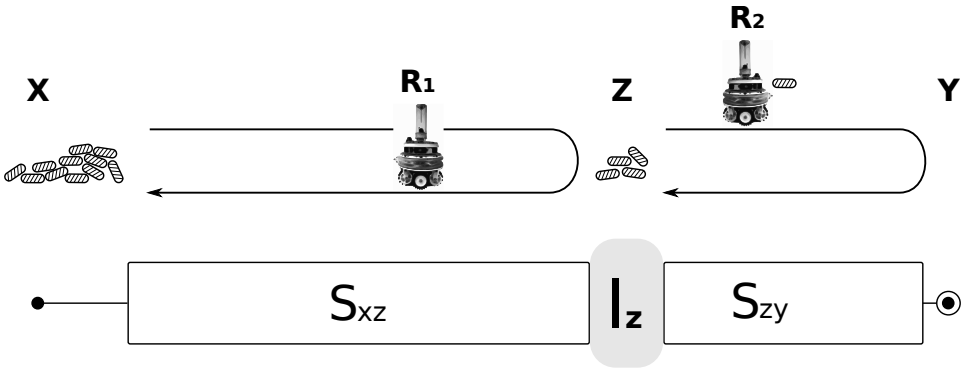

Figure 3.2: Illustration of an interface I_z between two sub-tasks S_{xz} and S_{zy}. At the interface, the output of S_{xz} becomes the input for S_{zy}. The interface therefore defines where the sub-task S_{xz} finishes and S_{zy} begins.

Figure 3.2 illustrates the object transportation task partitioned into two sub-tasks, S_{xz} and S_{zy}. In this example, an interface I_z links the two sub-tasks S_{xz} and S_{zy}. At the interface, the objects transported by the robot R_1, performing S_{xz}, are deposited on the ground. There, the robot R_2, performing S_{zy}, picks them up and completes transportation. In other words, at the interface Z the output of S_{xz} becomes the input for S_{zy}. We represent interfaces as colored areas between the two rectangles representing the sub-tasks. Analogously to tasks and sub-tasks, interfaces are labeled with a name.

Interfaces are important because of their relation with the amount of work: a certain amount of work defines the size of a sub-task which, in turn, identifies the location of the interface where that sub-task ends. In the example of Figure 3.2, the amount of work contributed by the robot R_1 defines the length of S_{xz} and the location of I_z. Notice that the same concept can be expressed also in the following terms: the robot R_1 decides the location Z where it deposits objects (i.e., the

location of the interface I_z) and this determines the amount of work the robot contributes to transportation (i.e., the length of S_{xz}). Due to the relations between the size of sub-tasks, amount of work, and interfaces, the two formulations are equivalent.

Interface type. In our study, we identify two types of interfaces: *fixed* and *movable* interfaces. As the name suggests, a fixed interface is characterized by the fact that its location is defined a priori. This has a profound implication on the task partitioning process. Since the position of an interface defines where a sub-task finishes and another begins, the fact that the location of the interface cannot be changed imposes a constraint on the amount of work contributed by the sub-tasks. The presence of fixed interfaces is typically due to a discontinuity in the space in which the task must be performed. The result is that the interface is implicitly defined within the task and its presence is imposed a priori. In foraging, fixed interfaces are typically due to obstacles that the robots cannot overcome.

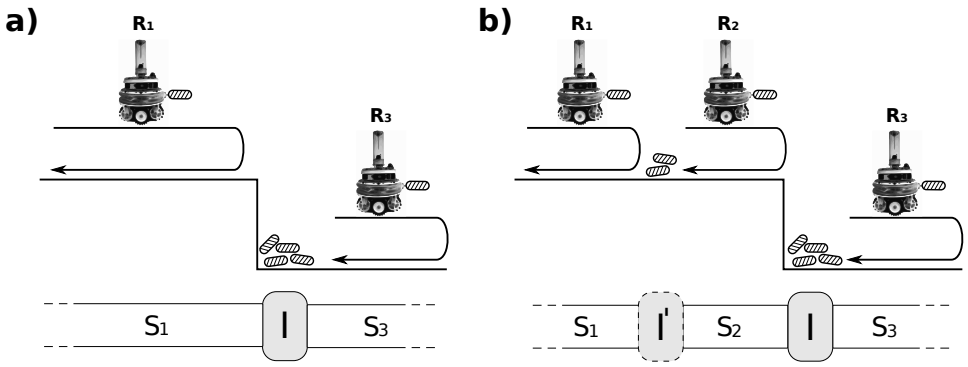

Figure 3.3: Illustration of a fixed interface.

For example, in Figure 3.3 a step is located between the source (left-hand side, not represented) and the nest (right-hand side, not represented). The step imposes a constraint on the amount of work that can be performed by the robot R_1. More specifically, the step limits the maximum amount of work the robot can contribute: an object cannot be directly transported by R_1 beyond the step. Therefore, the step limits the maximum size of the sub-task S_1 performed by R_1. Notice that, as represented in Figure 3.3b, R_1 may decide to contribute with a lesser amount of work. However this decision does not change the nature of the fixed interface I represented by the step: eventually another robot (R_2) will encounter the step on its way to the nest and its work will be constrained. In the schematic notation, we distinguish

between a fixed and a movable interface using the contour of the colored area: dashed for a movable interface and continuous for a fixed interface.

Differently from a fixed interface, a movable interface is not implicitly defined within the task. On the contrary, its presence is due to a decision made by the robots. For example, in Figure 3.3b, the presence of the interface I' is due to robot R1 decision to deposit objects on the ground. The robot can select the position of the interface I' (hence the name movable) and therefore the amount of work it contributes with its sub-task – i.e., the size of the sub-task S_1 it performs.

Type of transfer at the interfaces. Besides its type, a second property characterizes an interface: the input-output relation between the interfacing sub-tasks. This relation defines the way the output of a sub-task becomes the input for the sub-task that follows. There are two modalities by which this input-output relation can be implemented: *direct* and *indirect*. In biology, these terms refer to the way material is transferred between workers when a transportation task is partitioned into sub-tasks (see Ratnieks and Anderson, 1999). We use the same terminology, since we study similar contexts. For simplicity, we refer to the interface to be either "direct" or "indirect", even if this is somehow an abuse of language: it is the way the output of a sub-task becomes input for the following one that is direct or indirect, not the interface itself.

When the interface is direct, an individual completing a sub-task directly hands over the output of its sub-task to a second individual working on the sub-task that follows. In foraging, this entails directly handing over an object from a robot to another. The interface is synchronous since it requires two individuals to be present at the same time. An indirect interface allows an individual completing a sub-task to store the output of the sub-task at the interface, where it can subsequently be fetched and utilized as input for the following sub-task. In foraging, this consists in one robot depositing an object at the interface (e.g., on the ground or in a container) where another robot can pick it up at a later time. The interface is therefore asynchronous: it does not impose the presence of both individuals at the same time to perform the transfer.

Interface examples. Figure 3.4 illustrates the four possible combinations in relation to the type of interface and input-output relation between the sub-tasks linked by the interface. The first row represents fixed interfaces, the second row movable interfaces. The left-hand side

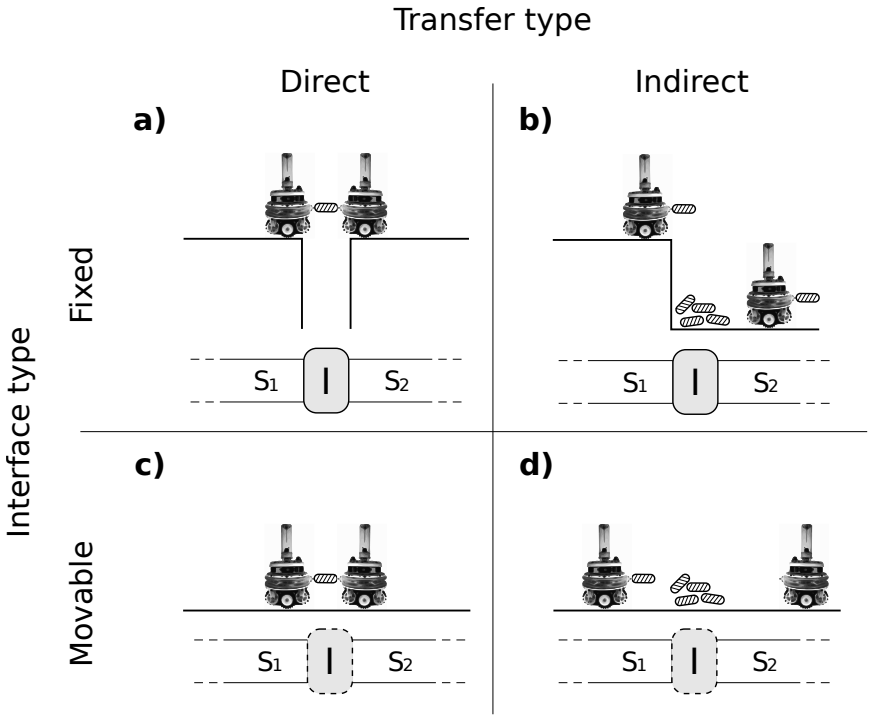

Figure 3.4: Types of interfaces (rows) and type of transfer at the interface (columns). If the interface is *fixed* (first row), the robots cannot choose its location autonomously. The position of a *movable* interface (second row) can be decided by the robots. *Direct transfer* (first column) requires two robots to be present at the same time at the interface, while *indirect transfer* (second column) does not.

column shows cases in which the interface is direct; the right-hand side column illustrates cases in which it is indirect.

As mentioned, in foraging the presence of a fixed interface is due to characteristics of the environment in which the robots perform foraging, such as the the gap illustrated in Figure 3.4a and the step in Figure 3.4b. The presence of a movable interface, on the other hand, depends on the decision of the robots to limit the amount of work they contribute to the overall task, as in the examples of Figure 3.4c and Figure 3.4d.

The input-output relation between the sub-tasks is determined by the characteristics of both the environment and the tasks. Analogously to the interface type, a discontinuity may impose what the input-output relation between sub-tasks must be. For example, the gap of Figure 3.4a imposes the interface to be direct. An object can cross the gap only if handed over from a robot to another. Analogously, the step

represented in Figure 3.4b forces the robots to drop objects and the interface is therefore indirect.

Characteristics of the task can also define the type of input-output relation between the sub-tasks. For example, bees utilize task partitioning in nectar foraging (Seeley, 1995). Since nectar is a liquid, it must be transfered directly from a bee to another. Robots transporting hazardous waste or precious items could do the same (see Figure 3.4c), to avoid that their load is left unattended in the environment. In other situations, it may be acceptable to release items in the environment (see Figure 3.4d).

Notice that, in principle, there may be cases in which the interface is both direct and indirect. For example, in situations such as the ones represented in Figure 3.4c and Figure 3.4d, the robots could in some occasions deposit objects to the ground and in others perform a direct transfer. In this dissertation however, we do not consider such cases: interfaces are either direct or indirect.

As explained in the following, the type of interface (fixed or movable) determines which decision a robot can make in terms of amount of work it contributes to the overall task. The type of transfer (direct or indirect), determines instead the overhead costs of task partitioning. The two properties of the interfaces therefore play an important role in the task partitioning process performed by the swarm.

3.3 Strategy

Given a task to be performed, there are different options to organize its execution. We define *partitioning strategy* (or more simply *strategy*) the way a task is performed in relation to i) the number of sub-tasks and ii) their size. Since, in general, there are several ways to partition a given task into sub-tasks, a strategy allows an external observer to describe how task partitioning has been applied to a task. Note that an individual robot merely executes an individual task or sub-task. A strategy expresses what can be observed at the level of execution of the overall task. Except for the case in which the task is executed as an unpartitioned task, the strategy that is used to tackle a task results from the combined actions of several robots involved in the execution of that task.

Figure 3.5 represents four possible strategies that a group of robots can utilize to perform the transportation task. A strategy may differ from another because either the number or the size of the sub-tasks are different. For example, the strategies represented in Figure 3.5a and Figure 3.5b are different because the former entails partitioning the

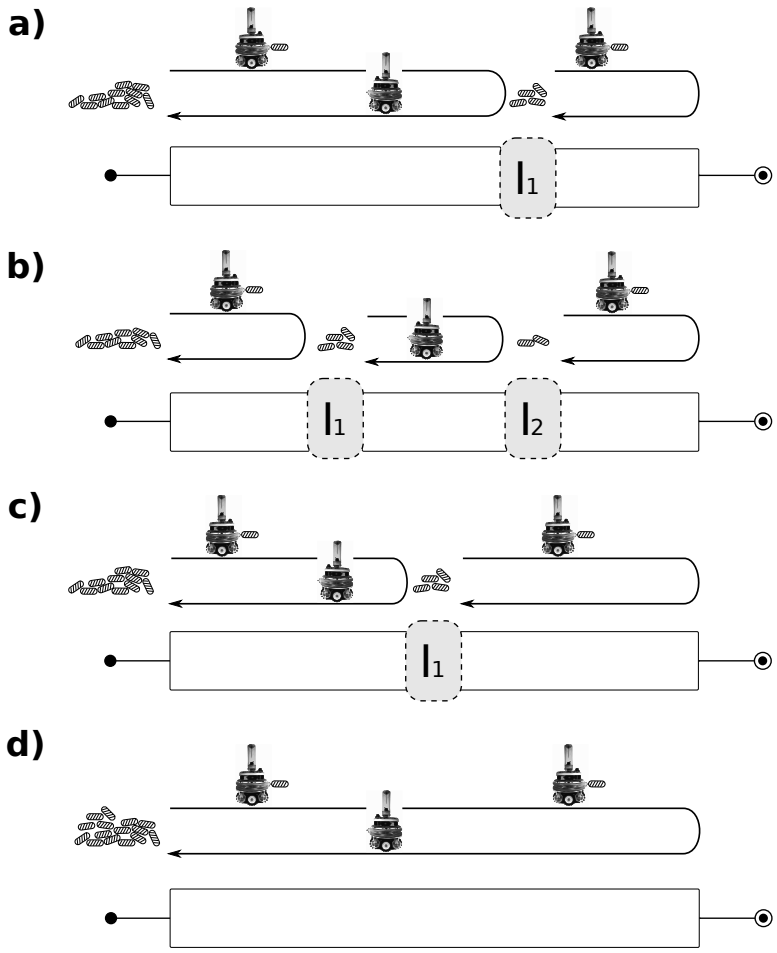

Figure 3.5: Different partitioning strategies applied to object transportation. **(a)** Object transportation is partitioned into two sub-tasks of different length. **(b)** Object transportation is partitioned into three sub-tasks of equal length. **(c)** Object transportation is partitioned into two sub-tasks of equal length. **(d)** Object transportation is performed as one unpartitioned task.

transportation task into two sub-tasks, the latter into three. On the other hand, the strategies represented in Figure 3.5a and Figure 3.5c both partition the transportation task in two sub-tasks. However the sub-tasks differ across the two in terms of amount of work contributed by the robots (i.e., the sizes of the sub-tasks are different). Consequently also the way the object transportation task is executed differs across the two examples. In general, the given task can be executed without employing task partitioning, as represented in Figure 3.5d. We refer to the strategy that consists in executing the overall task as

a single piece of work as the *non-partition strategy*.

Cost of a strategy. Our approach to autonomous task partitioning is based upon the idea of associating a *cost* to the execution of tasks. The cost measures the amount of resources that are spent to perform a task. The way costs are expressed depends on the specific context and application. For example, if the objective is to perform tasks as quickly as possible, costs can be expressed as time. Further examples of costs are energy and other resources, such as materials. Together with the concept of amount of work, the concept of cost is one of the foundations of our approach. The importance of costs is that they provide information about a strategy that can be used to evaluate how suited that strategy is to perform a task.

In general, the cost of performing a task varies with the strategy being employed. Consider, for example, the leaf foraging activity of the leaf-cutting ant *Atta sexdens*, described in Section 2.2.1. The task is the collection of a leaf piece and its delivery to the nest. *Atta* ants are known to employ task partitioning. Some individuals work on a tree, cutting and dropping leaves to the ground. Other ants cut pieces from the leaves found on the ground beneath the tree and transport them to the nest (Fowler and Robinson, 1979). In this case, the strategy consists in performing the task as a sequence of two sub-tasks (harvesting from the tree and transporting on the ground). An alternative strategy would be not to partition the task. In the latter case, an ant would climb the tree, cut a piece of leaf and transport it to the nest. The result, upon the completion of the task, would be the same as in the previous case: a leaf piece is transported to the nest (i.e., a task is completed). However, this strategy would be more costly in terms of energy spent, since each ant would be obliged to repeatedly climb up and down the tree.

Under a different definition of cost, the strategy that employs task partitioning could be disadvantageous. For example, if the goal were to maximize the exploitation efficiency of leaves on a tree, the partitioning strategy would not be advantageous. In fact, many of the leaves dropped to the ground are not found by any ant and therefore they cannot be utilized by the swarm (Hubbell et al., 1980). If each ant harvested leaf pieces directly on the tree, the amount of lost leaves would be reduced. However, the *Atta* ants employ task partitioning and this suggests that energy efficiency is their main concern.

In general, the costs of task partitioning are not only due to the execution of the sub-tasks: task partitioning requires additional coordination which results in overhead costs that concentrate at the inter-

faces. These overheads largely depend upon the input-output relation between interfacing sub-tasks. If the interface is direct, the overheads are due to the coordination required to find a transfer partner and to perform the transfer itself (Anderson and Ratnieks, 1999; Arnan et al., 2011). If the interface is indirect, overheads may occur in the form of losses at the task interface, such as in the leaf foraging example mentioned above.

3.4 Autonomous Task Partitioning in Swarms of Robots

We define **autonomous task partitioning in a robot swarm** as:

> ***the process by which the robots of the swarm autonomously define sub-tasks of a given task in terms of amount of work.***

There are two possible situations that the robots may face (see Section 3.2):

1. there are no constraints on the amount of work;
2. there are constraints on the amount of work, due to a fixed interface;

Concerning the latter situation, we only consider the cases in which the interface can be bypassed. Cases in which the use of the interface is the only possibility available to the robots are uninteresting since they do not require the robot to make any decision and therefore they are not studied in this dissertation. In the case of foraging, in which the task is transportation, bypassing a fixed interface means that the objects can be transported along a different path.

To achieve autonomous task partitioning in a swarm, the robots must be able to tackle both the case in which the amount of work is constrained and the case in which it is not. Therefore, they have to be able to make the following decisions:

1. decide the amount of work they contribute with their sub-task;
2. decide whether to use a fixed interface or to bypass it.

These two decisions are the fundamental building blocks upon which autonomous task partitioning is built and that allow the definition of complex partitioning strategies.

Within our approach, the robots make decisions individually: the overall partitioning strategy results in a self-organized manner from the

composition of individual decisions. More specifically, neither negotiation nor collective choice processes are involved in the definition of the strategy of the swarm.

Figure 3.6 illustrates, with the use of examples, how the individual decisions define the overall strategy. In the figure, the robots must collect objects located on top of a step, and deliver them to the nest (not represented), located on the right-hand side of the figure. The step is a fixed interface, that the robots can bypass using a ramp, located on the left-hand side of the figure.

In Figure 3.6a, the robot R_1 decides to employ the fixed interface: the robot drops objects from the step. The robots R_2 and R_3 collect these objects at the bottom of the step and complete the transportation –i.e., they decide to contribute an amount of work that completes transportation. The strategy that results from these individual decisions entails partitioning the transportation task into two sub-tasks, linked by a fixed interface located at the bottom of the step.

Figure 3.6b represents a situation where none of the robots employ the fixed interface. Each robot contributes with a different amount of work and the transportation task is partitioned into three sub-tasks. In this case the two interfaces are movable: the robot R_1 could modify the amount of work it contributes and change the location where its sub-task interfaces with the one performed by R_2. Analogously the decisions of R_2 have an effect on the work of R_3.

Figure 3.6c, illustrates that "hybrid" strategies may result from the individual decisions of the robots. In the example, R_1 uses the fixed interface, R_2 decides to perform transportation as an unpartitioned task, and R_3 collects the objects at the bottom of the step. The strategy is hybrid in the sense that, at the global level, there is no unique strategy being employed. In the example, certain instances of the transportation task are performed as sequences of two sub-tasks, while others are performed as unpartitioned tasks. When robots in a swarm make decisions independently, analogous situations are frequent.

In general, there is no single strategy that is always preferable to the others: depending on the goals, one strategy may be a good choice or it may not. For example, the strategy represented in Figure 3.6a is a good choice if the goal were maximizing the foraging speed. The strategy represented in Figure 3.6b, on the other hand, is a better option if the objects are fragile and the goal were maximizing the number of (intact) objects delivered to the nest.

Contribution of the dissertation. The task partitioning mechanisms that have been proposed so far are built with the sole purpose of reduc-

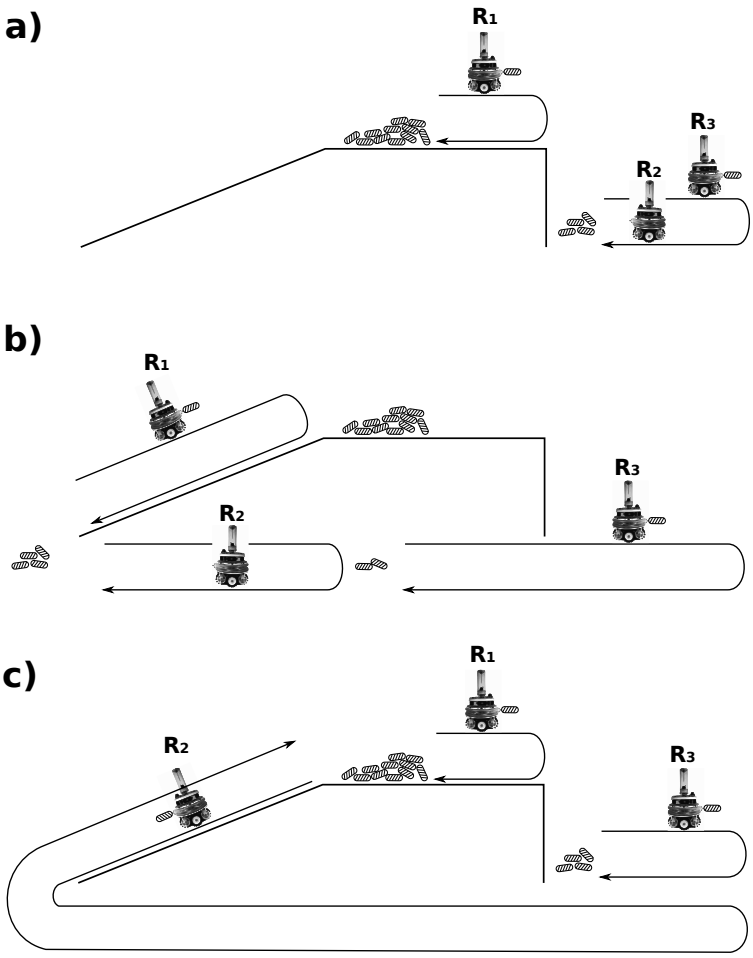

Figure 3.6: Examples of collective strategies resulting from the individual decisions of the robots. **(a)** The robot R_1 decides to drop objects from the step. At the bottom, the two robots R_2 and R_3 collect the objects and complete transportation. The resulting strategy entails partitioning transportation into two sub-tasks, linked by a fixed interface. **(b)** None of the robots decides to drop objects from the step. Each robot transports objects for a certain distance and transportation is partitioned into three sub-tasks, linked by movable interfaces. **(c)** R_1 drops objects from the step. At the bottom R_3 collects the objects and completes transportation. R_2 performs transportation as an unpartitioned task. The strategy employed by the robots is "hybrid": certain instances of the transportation task are performed as sequences of two sub-tasks, others as unpartitioned tasks.

ing physical interference. Such mechanisms are suitable in situations in which interference is indeed the dominant factor. However, there are also other phenomena that play a role in foraging. Using mecha-

nisms built to reduce interference in contexts where interference is not the key problem may bias the partitioning strategy towards a solution that is not appropriate for the situation. Moreover, in all the existing works, the task partitioning process is blended with the behavior of the robots and it therefore depends on the specific tasks and contexts in which these tasks are executed. This limits the applicability of the same mechanisms to other tasks and contexts.

The approach presented in this dissertation is different from what proposed so far in two aspects: i) the task partitioning process is built upon general concepts and ii) it is decoupled from the behavior of the robots. Even though the concepts of cost and of amount of work may need to be adapted to specific tasks, contexts, and applications, the process that implements task partitioning does not require modifications. This broadens the applicability of our approach, compared to existing approaches. Additionally, we do not make any a priori assumptions about the factors that determine the best partitioning strategy. In fact, each robot has the goal of reducing the task execution costs, and this results in a swarm-level partitioning strategy which implicitly takes into account the factors that have an impact on these costs.

3.5 Application to Other Domains

We developed the approach to autonomous task partitioning described in this chapter with the goal of using it in foraging. However, the approach is based upon generic concepts and therefore it can potentially be applied to other contexts. In this section, we provide two examples illustrating that the concepts at the basis of our approach can be identified also in other contexts: in Section 3.5.1 we discuss a construction scenario and in Section 3.5.2 the case of a baker preparing cakes. The latter case highlights that the elements that are relevant to our approach are general and can be found in contexts other than robotics.

3.5.1 A Construction Scenario

A swarm of robots has to build a structure composed of walls. Here, we can identify a *task* in the construction of one of the walls of the structure. A *sub-task* consists in building part of a wall. The sub-tasks exhibit a sequential dependency: walls are constructed stacking bricks or other materials starting from the ground and building upwards. This imposes an execution order between sub-tasks.

A natural way to express the *amount of work* is through a measure related to the actions performed by the robots, for example the energy

spent. The *cost* can be measured in different ways. One could measure time if the goal were completing the structure as fast as possible. Conversely, if the goal were minimizing the waste of resources, cost could be related to the quantity of construction material used.

In the example considered here, the input-output relation between sub-tasks consists in resuming the construction of an unfinished wall. The location of an *interface* can therefore be identified in the height of a partially constructed wall. The interfaces are indirect since they do not impose tight synchronization: a robot can continue at any moment in time the construction of a wall that was left unfinished by another robot. Therefore in this context the amount of work contributed by a robot defines the height of the wall it is constructing.

Each robot has a limited reach and therefore it can only build a wall up to a certain height. The amount of work a robot can contribute to the construction of a wall is therefore constrained. The maximum height of a wall identifies the location of a fixed interface. Differently from the case of foraging, in which characteristics of the environment impose the presence of the interface, in this case its presence is due to characteristics of the task and the robots. The interface can be bypassed with the aid of an external structure or a tool that allows the robot to extend its reach (e.g., a scaffold). The structure or tool require other actions to be performed by the robot and therefore its usage entails additional amount of work.

In the context described here, autonomous task partitioning is the process by which the robots define a strategy that they utilize to build the structure. One can imagine, for example, a strategy that entails each robot remaining on a certain level of a scaffold and build the part of the wall within reach. In this case the construction of walls would be implemented as a sequence of sub-tasks. An alternative possibility is the use of the non-partition strategy: the robots climb scaffolds and directly build complete walls.

3.5.2 A Tasty Scenario

In this example, instead of a group of individuals performing tasks, we consider a single worker: a baker that prepares cakes to sell in his shop. In this case, we identify a *task* in the preparation of one cake, while a *sub-task* consists in executing part of the recipe for preparing the cake.

As in the previous example, the *amount of work* can be related to the operations the baker must perform in order to bake a cake. An *interface* can be identified in the stage of an unfinished cake. The presence of a fixed interface may be due to operations that cannot be interrupted, such as beating egg whites. Once the cook initiates

the operation, he must complete it or the process will fail. Therefore beating egg whites constrains the work of the baker. In some situations, the interface can be bypassed. For example, imagine that an alternative recipe allows the cook to use a different ingredient than egg white, for example cream. Using other ingredients entails different operations to be performed and therefore a different amount of work.

Depending on the way *costs* are measured by the baker, the use of cream over egg whites could be an advantage. If costs are expressed as time, using cream may be preferable as it saves the cook some time that he can use to prepare more cakes. On the other hand, it might be that the customers prefer cakes made with egg white, therefore the eggs are a better option if the baker measured costs as customer dissatisfaction.

The example described allows us to illustrate an aspect of task partitioning that is neglected in foraging or construction: task partitioning can be used to delay the execution of certain operations (i.e., sub-tasks) in time, allowing a single worker to concurrently progress in many tasks. This is analogous to what happens in an operating system: the processes are executed as sequences of operations performed at different moments. The result is that several processes are active at a given moment in time. In the case of the baker, partitioning the preparation of cakes allows him to prepare many cakes in parallel and to optimize certain operations. For example he can first prepare a number of cakes and put them all together in the oven to be baked at once, thus saving energy. Analogously he can work on a cake while the dough of another cake becomes leaven, thus saving time.

3.6 Summary

In this chapter, we framed task partitioning within the context of foraging and described our approach to autonomous task partitioning in such a context. We concentrate our study on foraging because it represents an important benchmark in swarm robotics and because it is an abstraction of relevant real-world applications. Additionally, the robotic platforms available in our laboratory have characteristics that render them suited for applications similar to foraging (e.g., object collection, search and retrieval, transportation, exploration). Therefore foraging is a topic of general interest within our research group.

Despite the fact that we focus on obtaining autonomous task partitioning in a specific context, we base our approach on generic concepts. This has two main advantages, with respect to the task partitioning algorithms proposed in the literature. First, the use of generic concepts allows us to reason in an abstract way and to decouple the process

implementing task partitioning from the specific tasks, actions, and behaviors of the robots. We propose a decisional layer that can be used on top of the behavioral repertoire of the robots to implement task partitioning. The second advantage is that we do not make a priori assumptions about the specific factors determining the partitioning strategy that should be used to tackle a given task. Instead, the strategy is determined by the way costs are measured and therefore takes into account implicitly all the factors that have an impact on costs.

We are confident about the fact that our approach can be applied to tackle other problems than foraging without requiring major modifications. We provided examples illustrating that the very same concepts at the basis of our approach can be identified in other contexts.

As a final remark, we restrict our study to cases in which tasks are partitioned into sequences of sub-tasks. However, tasks can sometimes be partitioned into sub-tasks that do not impose any execution order. The study of an approach to tackle such situations is still an open problem and it is discussed in the final chapter of this dissertation.

Chapter 4

Tools

In this chapter, we describe the hardware and software tools that we utilized to carry out the experiments presented in this dissertation. The goal of the chapter is not to provide an exhaustive description of these tools, but to illustrate their main characteristics. In Section 4.1, we describe the tools that we utilize in the experiments presented in Chapter 5: the e-puck and the TAM. The e-puck is a small wheeled robot developed at EPFL.[1] The TAM is a device that we developed in our laboratory and that can be used to abstract tasks for an e-puck. In Section 4.2, we present the marXbot, a robotic platform developed within the Swarmanoid project and we describe its characteristics. The marXbot has been utilized in the foraging experiments presented in Chapter 6. In Section 4.3, we describe ARGoS, the simulator software that we utilized for the simulation-based experiments of both Chapter 5 and Chapter 6.

4.1 The e-puck and the TAM

The e-puck[2] is a mobile robot developed as an open-hardware project at École Polytechnique Fédérale de Lausanne (Switzerland). The e-puck has a cylindrical shape, with a diameter of 70 mm, and it is powered by a removable Li-ion battery. The main processor controlling the robot is a dsPIC[3] microcontroller. Extension boards, such as the ground-color sensors, the omnidirectional camera, and the infrared range and bearing communication device can be added to enhance the basic capabilities of the robot.

[1] École Polytechnique Fédérale de Lausanne.

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

[3] `http://www.microchip.com`

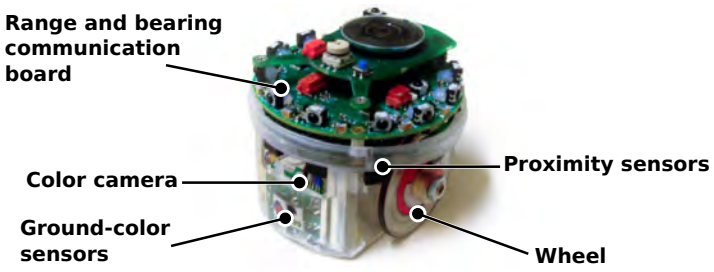

Figure 4.1: Robot e-puck, simulated in the experiments presented in Chapter 5. The figure highlights the sensors and actuators used in the experiments.

Figure 4.1 depicts an e-puck and highlights the sensors and actuators that are used in the simulation-based experiments described in Chapter 5. The e-puck can navigate the environment at a maximum speed of 0.12 m/s by means of a differential drive system. Eight infrared proximity sensors are used to implement obstacle avoidance. The sensors can perceive obstacles up to a distance of 0.05 m. The same sensors measure the intensity of the ambient light and they are used in the experiments to implement phototaxis behaviors. Three infrared sensors, mounted in front of the robot and pointing downwards, identify the color of the ground. A 640×480 pixels color camera, pointing forwards, provides the e-puck with basic vision capabilities. The camera allows the e-puck to perceive LEDs and reliably distinguish between 3 different colors in controlled lighting conditions. The infrared range and bearing extension board (Gutiérrez et al., 2009) allows line of sight communication between the e-pucks (details are given in Section 5.6.1). For more information about the e-puck, we refer the reader to the work of Mondada et al. (2009), and to the website `http://www.e-puck.org`.

The only actuators available on the e-puck are the wheels, the LEDs, and the speaker. The types of experiment that can be performed with the robot are therefore limited. To overcome this limitation, we designed and prototyped a device, the *TAM*[4] (Brutschy et al., 2010). Figure 4.2 reports an actual picture (left) and a schematic representation (right) of an e-puck entering a TAM. Each TAM features a light barrier, an infrared sensor, and 3 RGB LEDs. The e-puck enters in the TAM, approaching the LEDs that it can perceive using the RGB camera. The TAM detects the presence of a robot by means of the light barrier. The user can program the TAM and define its behavior, that is, the way LEDs are actuated on the basis of the events detected (i.e., a robot entering or exiting the TAM). The infrared sensor allows

[4]Acronym for *task abstraction module*.

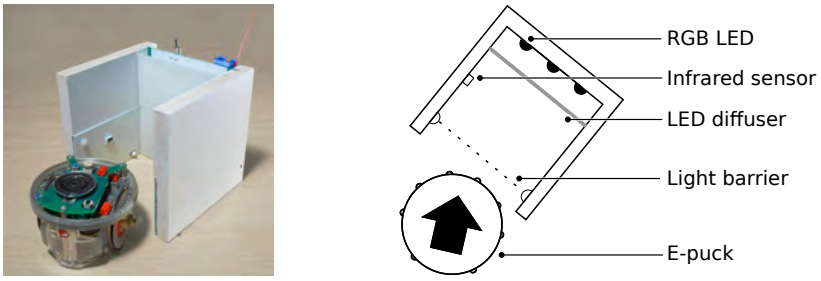

Figure 4.2: An e-puck entering a TAM: picture (left) and schematic representation (right)

the TAM to exchange information with the e-pucks.

The TAM can abstract tasks that the e-pucks perform. For example, in the experiments described in Chapter 5, the TAMs are used to abstract object handling. By entering a TAM, an e-puck picks up or drops a (virtual) object, depending on the color of the TAM and the state of the robot. For additional details about the TAM and its usage to abstract tasks, we refer the reader to the work of Brutschy et al. (2010).

4.2 The MarXbot

The marXbot is a mobile robotic platform that was developed within the Swarmanoid project.[5] The marXbot architecture is distributed: the robot is composed of a set of separate modules, each hosting part of the sensors and actuators available on the robot. Each module hosts a dedicated dsPIC microcontroller, that controls the sensors and actuators located on that module. The modules communicate via asynchronous events, which are sent on a bus. The main processor board is a 533 MHz ARM 11 i.MX31, running Linux. The distributed architecture allows sensor data preprocessing and low-level control to be performed directly on the modules. This reduces the computational load of the main processor and the communication load of the bus (Magnenat et al., 2011).

Figure 4.3 represents the marXbot, and highlights the sensors and actuators that are used in the experiments presented in Chapter 6. The marXbot is roughly cylindrical-shaped, it has a diameter of 170 mm, an height of 290 mm, and it weighs 1.8 kg. A hot-swappable Lithium-Polymer battery powers the robot. The marXbot can navigate in the environment using a differential drive system composed of two *treels*, a combination of tracks and wheels. The tracks allow the robot to move

[5]Within Swarmanoid, the marXbot is also called “foot-bot”.

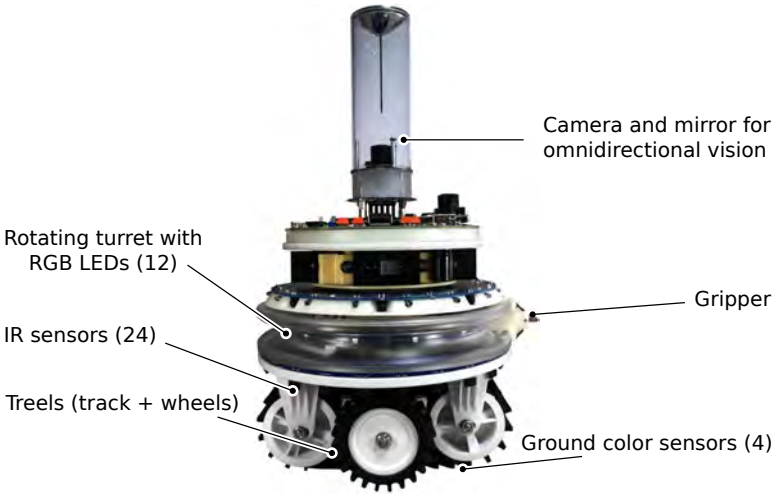

Figure 4.3: Photography of a marXbot. The figure highlights the sensors and actuators utilized in the experiments.

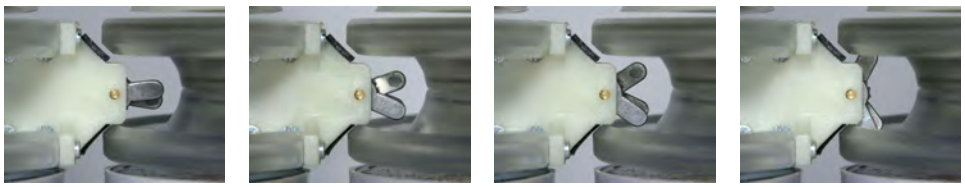

Figure 4.4: From left to right: the gripper of a marXbot opens inside the ring of another marXbot. The mechanism allows the marXbots to connect to each other and form multi-robot structures.

on moderately rough terrains, the wheels enhance dexterity while turning. A plastic ring hosts 12 RGB LEDs and a gripper. The ring can rotate around the vertical axis of the robot. The ring is used both to diffuse the light emitted by the LEDs and to allow the physical connection of another marXbot. When opened, the gripper of a marXbot fits the plastic ring, as illustrated by the sequence of pictures in Figure 4.4. This allows the marXbots to self-assemble and form multi-robot structures (see examples in Mathews et al., 2011).

The marXbot is equipped with 24 infrared proximity sensors, evenly distributed around the body of the robot. The infrared sensors are used as bumpers to perceive obstacles up to a distance of 0.05 m. The same sensors measure the intensity of the ambient light and they are used to detect light sources and perform phototaxis. Four infrared sensors, located underneath the robot between the two tracks, are used to recognize the color of the ground.

A 2 MP color camera, pointing to a spherical mirror on top of a

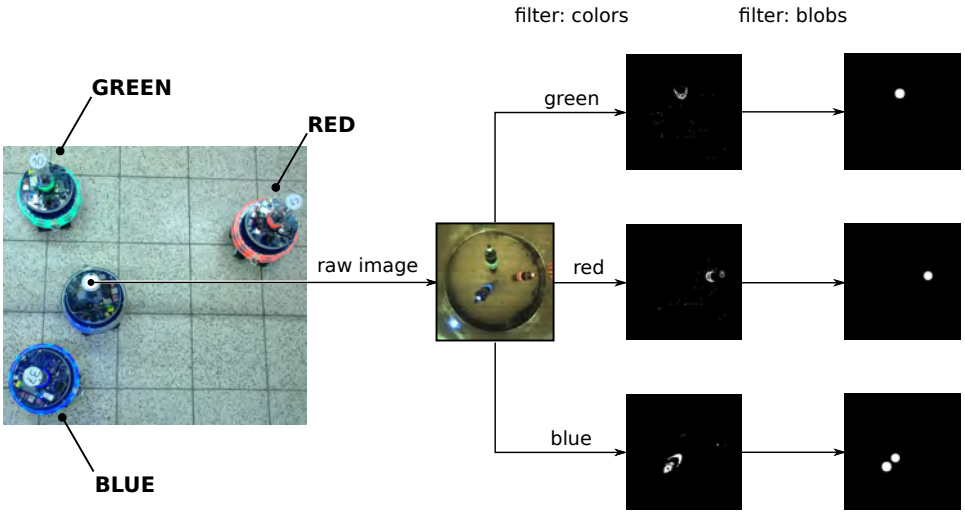

Figure 4.5: Representation of image processing on a marXbot. Left-hand side: a marXbot is surrounded by other 3 robots, each has its LEDs turned on in a different color. The omnidirectional camera of the central marXbot perceives the raw image reported in the center. A first filter is applied to this image. The filter extracts colors from the raw image and produces an additional image for each filtered color (first column of images). Each image is further filtered, to cluster pixels of the same color into blobs (last column). The information about blobs (distance, color, and angle) can be used in the controller of the robot.

transparent tube provides omnidirectional vision. Figure 4.5 illustrates how the vision system of the marXbot works. The camera perceives objects in the surrounding, up to a distance of around 1 m. In the figure, the marXbot in the center is surrounded by other 3 robots, each with the LEDs turned on in a different color. The raw camera image, shown in the center of Figure 4.5, is filtered to extract color information (first column of dark images). A second filter is applied to the resulting images to detect blobs of the corresponding color (see last column of images). Blobs represent areas of interest in the image: for example an object to be gripped, or the LEDs of a robot to be avoided. The filtering mechanism is based on the open source software *opencv*. [6] The filters to be applied to an image are specified and tuned via an XML configuration file.

In addition to the sensors and actuators described, that are used in our experiments, the marXbot is also equipped with a second color camera, a rotating scanner to measure distances, an infrared range and bearing communication board, 8 ground-color sensors for hole detec-

[6] `http://opencv.org`

Figure 4.6: Components of the the plastic ring of the object (left) and the plastic ring assembled (right).

tion,[7] gyroscopes, and accelerometers. A comprehensive description of the marXbot can be found in the work of Bonani et al. (2010).

To carry out the foraging experiments with the real marXbots, we use objects that we designed in our lab. Figure 4.7 depicts one of these objects; the schematic representation on the right-hand side reports its dimensions. Each object is composed of a 90 mm plastic ring (see Figure 4.6) that can host the gripper of a marXbot. The ring is fitted on top of a 60 mm tall PVC pipe with a diameter of 80 mm (bottom part). In the experiments we carried out with the real marXbots, a red cardboard disk is glued on top of the plastic ring.

The marXbots can perceive the colored cardboard disk on top of the object with their omnidirectional camera, up to a distance of roughly 0.5 m.[8] The marXbot gripper allows the robot to grasp the plastic ring of the object for transportation. Figure 4.8 depicts a marXbot carrying an object. Details about the design of the objects and the way they can be utilized in the experiments are available in the work of Brutschy et al. (2012a).

[7] The marXbot has two sets of ground-color sensors: a set is used for ground-color recognition, the other for hole detection.

[8] The actual range of the camera is wider, but the objects can be perceived reliably only from a shorter distance.

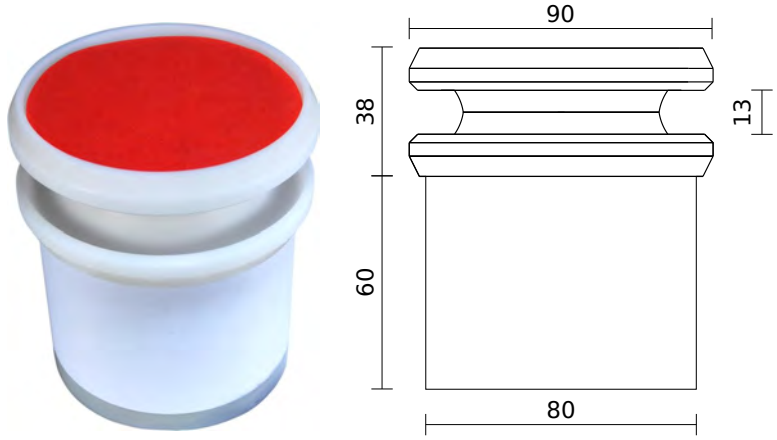

Figure 4.7: Objects utilized to perform foraging experiments with the marXbots. On the left-hand side an actual picture of an object; on the right-hand side a schematic representation reporting the dimensions in mm.

4.3 ARGoS

We perform all the simulation-based experiments described in this dissertation using *ARGoS*, acronym for *autonomous robots go swarming* (Pinciroli et al., 2012). ARGoS is a simulation software written in C++, that was developed within the Swarmanoid project. The software is open source and it is available online.[9] ARGoS allows the real-time simulation of swarms composed of thousands of robots.

The main characteristic of ARGoS is that it allows multiple physics engines to run in parallel. Different groups of robots can be assigned to separate physics engines, each of which models the physics of that subset of robots. Alternatively, different portions of the simulated space can be assigned to separate engines, simulating the physics in the portion of space assigned. Each engine can be executed in a dedicated thread to achieve parallel execution on multi-core computers and reduce the simulation time (Pinciroli et al., 2012).

Besides efficiency and speed, ARGoS was designed with the goal of being highly customizable by the end user. Since ARGoS is utilized by many researchers, it must answer different needs and it must allow new features to be easily added. The architecture of ARGoS, represented in Figure 4.9, renders this possible by allowing easy customization and extension of the software.

The core of ARGoS is the *space*, which contains information about the state of every simulated entity. The remaining elements (enti-

[9] `http://iridia.ulb.ac.be/argos`

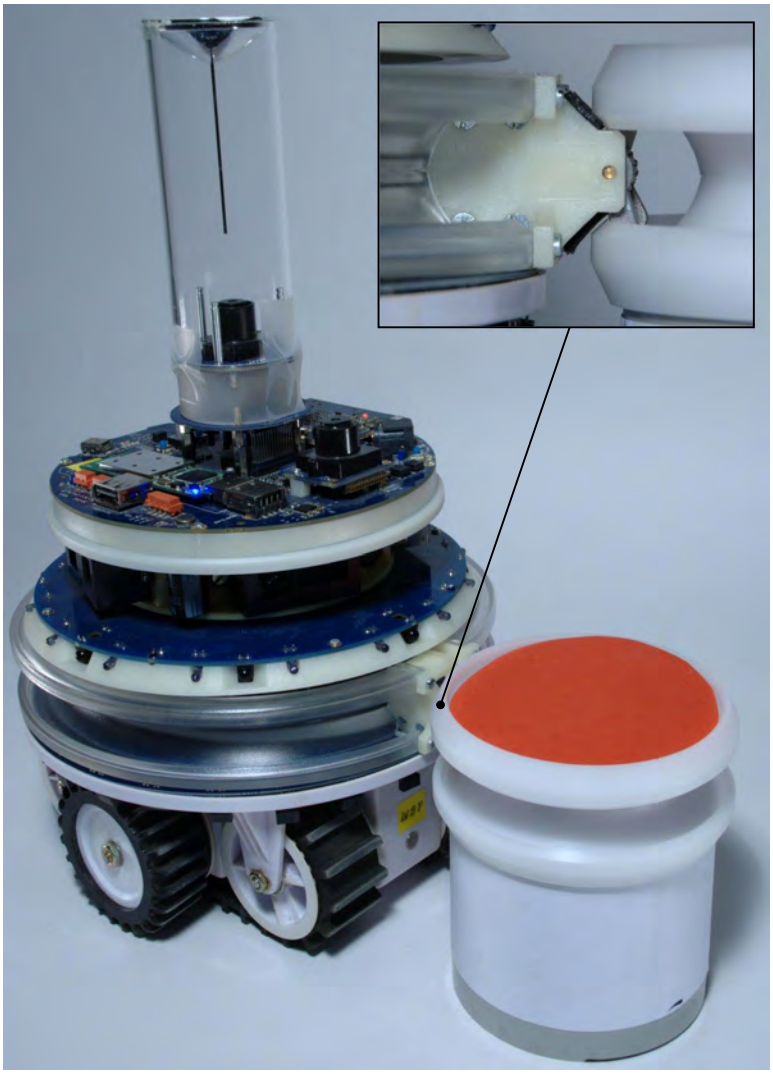

Figure 4.8: A marXbot carrying an object. The top part of the figure shows a closeup of the marXbot gripper inside the plastic ring of the object.

ties, sensors, actuators, visualizations, and physics engines) are implemented as plugins. Thanks to this design choice, each simulated entity (e.g., a sensor, or an actuator) can have multiple plugins that implement it. The user can customize the simulation by specifying via an XML configuration file which specific implementation of a given entity (i.e., which plugin) is instantiated at run-time. Implementations differ in the level of detail at which an entity is simulated and therefore the user can allocate computational resources where desired: he can select a fast and approximate simulation of some entities and a more

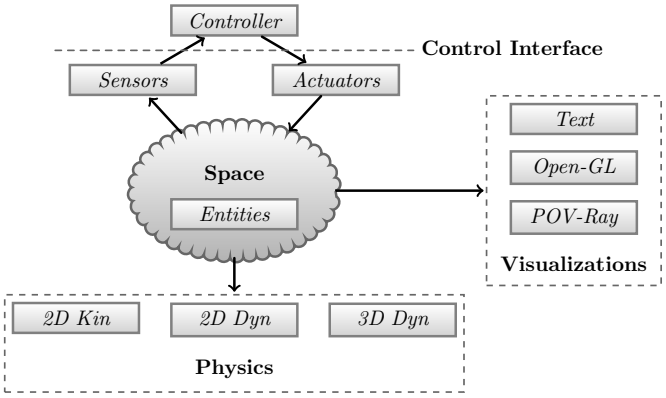

Figure 4.9: Architecture of ARGoS. The *space* is the core of ARGoS, all the other elements are implemented as plugins. The *control interface* is used by the controller to interact with sensors and actuators and it allows the same code to be used on real and simulated robots. The image has been taken from the work of Pinciroli et al. (2012) with permission of the author.

detailed and computationally expensive one for others. The fact that everything is implemented as a plugin also makes it possible to extend the software easily, without the need of modifying its core.

ARGoS allows easy porting of controller code to the real robots. This is possible thanks to the *control interface* (see top part of Figure 4.9). The control interface is an abstraction layer, used by the controller of a robot to access sensors and actuators. The control interface is shared across the simulated sensors and actuators and the real ones. In this way, to the controller it is transparent whether it is interacting with simulated or real devices. Cross-compilation allows the very same code developed in ARGoS to be used on the real robots. At the time of writing this dissertation, ARGoS supports cross-compilation for the Swarmanoid robots, the s-bots (Mondada et al., 2004), and the e-pucks.

Chapter 5

Deciding Whether to Use a Fixed Interface

In this chapter, we study the case in which a fixed interface constrains the amount of work that the robots can contribute with their sub-tasks. In general, the presence of fixed interfaces is due to a discontinuity either in the environment or in the task itself (see example described in Section 3.5.1). In foraging, fixed interfaces are typically due to an obstacle on the path to the nest, for example a gap in the ground or a step, that cannot be overcome by the robots.

In such cases, the robots can only decide whether to transfer objects at the interface or to bypass the interface using an alternative path leading to the nest. The two options entail, in general, a different amount of work to be performed by the robots and involve different costs. For example one of the two paths might be significantly longer than the other, or interference along one of the two paths might be higher.

As mentioned in Chapter 3, the capability of deciding whether to use a fixed interface is one of the two building blocks to define more complex partitioning strategies. In this chapter, we study a scenario in which such a decision must be made by the swarm and we propose algorithms based on our approach that can be used by the robots to make this decision. The case in which there is no fixed interface constraining the work of the robots is the focus of Chapter 6.

The rest of this chapter is organized as follows. In Section 5.1, we describe the foraging scenario that we use as a testbed and we frame the problem of selecting whether to use an interface in such a scenario. In Section 5.2, we describe the actual implementation of the scenario using ARGoS to simulate the TAM and the e-puck. In Section 5.3, we illustrate the application of our approach to autonomous task parti-

tioning in the studied scenario. In Section 5.4, we present an ad-hoc algorithm that we designed to tackle the studied problem. We report results of experiments in which we test the performance of the algorithm, its adaptivity to changes of the environmental conditions, and its scalability. In Section 5.5, we show that the problem of selecting whether to use an interface can be seen as a multi-armed bandit problem and tackled with existing algorithms. In Section 5.6, we study the implications of using explicit communication within the swarm. Finally, in Section 5.7, we summarize the contents and contributions of this chapter.

5.1 Description of the Problem

The studied problem. In this chapter we study the problem represented in Figure 5.1. The robots must perform transportation, and the amount of work a robot can contribute with a sub-task is constrained by the presence of a fixed interface. The interface can be bypassed using an alternative path to the nest.

If the robots decide to bypass the interface I, transportation is performed as an unpartitioned task T (Figure 5.1a). In this case we say that transportation is performed using the *non-partition strategy*. Conversely, if the robots use the interface I, the object transportation task is partitioned into a sequence of two sub-tasks S_1 and S_2, as represented in Figure 5.1b. The sub-task S_1 consists in transporting an object from the source to the interface, while the sub-task S_2 consists in transporting an object from the interface to the nest. In case the robots employ the interface I, we say that object transportation is performed using the *partition strategy*.

For simplicity, we consider here the case in which T corresponds to the overall transportation task. Since the interface is fixed, robots employing task partitioning are not free to define sub-tasks autonomously. Instead, the sub-tasks are determined a priori by the presence of the interface. In other words, the robots can only choose between two predetermined strategies that can be employed to perform transportation. Notice however that this is not a strict requirement: what presented in this chapter can also be applied to cases in which T is a sub-task of another task. In this latter case, the robots would have more freedom at the level of the definition of the overall partitioning strategy.

The goal of the robots is to maximize the number of objects delivered to the nest. This goal can be achieved by maximizing the object throughput and therefore we express costs in terms of time. Under this definition, the cost of the non-partition strategy is the time needed to

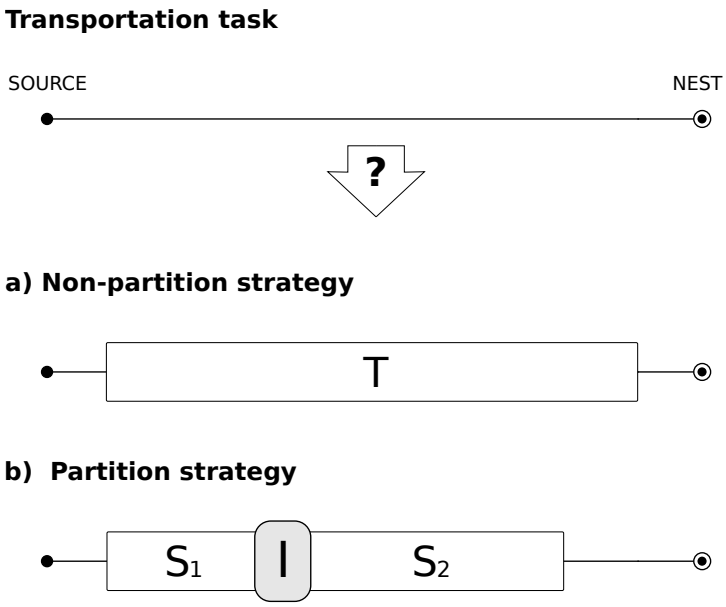

Figure 5.1: Representation of the task partitioning problem studied in this chapter using the schematic notation presented in Chapter 3. We study the case in which a fixed interface imposes a constraint on the amount of work a robot can contribute with a sub-task. The robots must decide whether to use the interface or bypass it. **(a)** The robots decide to bypass the interface: the given task is tackled as an unpartitioned task T. **(b)** The robots use the interface: the task is partitioned into a sequence of two sub-tasks S_1 and S_2, linked by the fixed interface I.

perform the object transportation task as an unpartitioned task T. The cost of the partition strategy is the sum of the times required to perform the two sub-tasks S_1 and S_2. The two strategies entail in general a different total cost due to a different amount of work required to perform transportation and, in the case of the partition strategy, to overhead interfacing costs. The costs of the two strategies determine which of the two is more advantageous to perform transportation.

Foraging scenario. The robots perform foraging in the environment represented in Figure 5.2. The environment is composed of two areas, one containing the source and the other containing the nest, separated by the fixed interface. We consider the case in which the source never depletes and it is situated in a known location.[1] The nest has unlim-

[1]Typically, foraging involves exploration of the environment with the goal of finding objects. Since the robots know the location of the source in advance, they are not, strictly speaking, performing foraging, but rather transportation. However, we use in this chapter

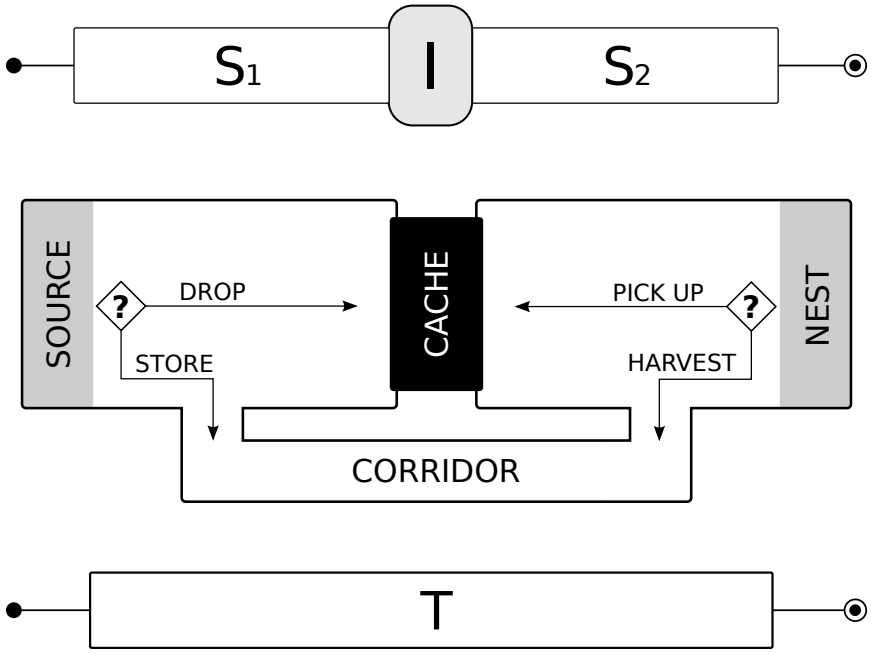

Figure 5.2: Representation of the studied foraging problem. The object transportation task consists in taking an object from the source and delivering it to the nest. The cache separates two ares of the environment, one containing the source and one containing the nest. The cache allows objects to be transferred from one area to the other. The corridor can be used by the robots to reach an area from the other. Robots choose between using the interface (i.e., the cache) or not (i.e., navigate through the corridor) in two cases, marked with "?" in the figure: after taking an object from the source, a robot chooses between STORING it at the nest or DROPPING it at the cache; after delivering an object in the nest, a robot chooses between PICKING UP the next at the cache or HARVEST one from the source.

ited storage capacity, while the capacity of the interface is limited. The interface allows objects to be transferred from one area to the other. We consider the case in which the interface is indirect (refer to Section 3.2). This means that an object can be dropped at the interface by a robot and subsequently picked up by another robot working on the opposite side without requiring that the first robot remains at the interface. In the rest of the chapter, we refer to the interface as the *cache*. Entomologists use the term *cache* to refer to locations in which insects temporarily store materials during transportation (Hart and Ratnieks, 2000). The cache can be bypassed using a *corridor*, which

the term *foraging* to refer to the overall swarm activity, composed of several instances of the object transportation task.

links the area containing the source and the one containing the nest and therefore allows the robots to reach an area from the other.

In the setup described, the use of the cache allows the robots to perform transportation using the partition strategy, as represented in the top part of Figure 5.2: the first sub-task (S_1) consists in taking an object from the source and drop it in the cache; the second sub-task (S_2) consists in picking up an object from the cache and delivering it to the nest. The corridor allows the robots to bypass the interface and perform transportation using the non-partition strategy (see bottom part of Figure 5.2): a robot directly reaches the source from the nest and the other way around, harvesting and storing objects. Which of the two strategies is more advantageous depends on the length and width of the corridor and on the cost of using the cache.

In the experiments we impose an *interfacing time* Π that the robots must spend to use the cache for either picking up or dropping an object. Π allows us to abstract the overhead costs of task partitioning and conveniently regulate the relative benefits of using the cache over the corridor.

The robots choose whether to use the cache in two situations, represented with a question mark in Figure 5.2. 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. After delivering 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. Notice that a robot can decide to use the cache in some cases and not to use it in others. Therefore, at the level of the swarm, the choice is not uniform: some robots may be using the cache while others may not.

5.2 Experimental Setup

In this section, we describe the experimental implementation of the foraging scenario presented in Section 5.1. All the experiments presented in this chapter are performed simulating e-pucks with ARGoS. Since the e-puck does not have the capability of grasping objects, we abstract object handling using the TAM. For more information about ARGoS, the e-puck, and the TAM, refer to Chapter 4.

Environment and simulation of the system. The environment in which the robots perform foraging, shown in Figure 5.3, is an actual implementation of the environment represented in Figure 5.2. The source, the cache, and the nest are implemented using TAMs organized

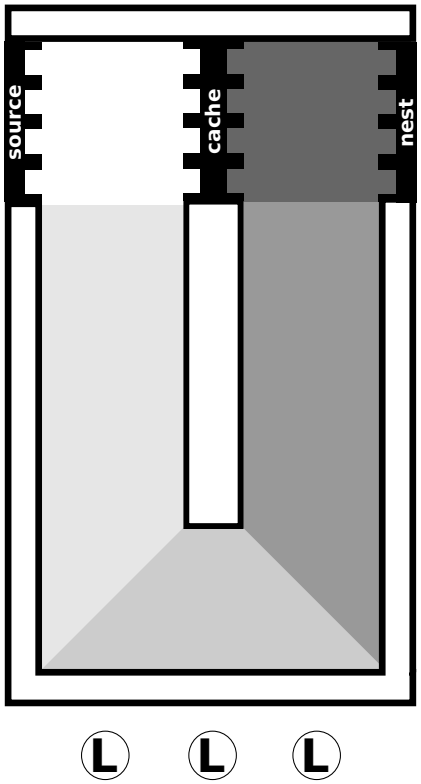

Figure 5.3: Representation of the experimental environment. Source, cache, and nest are implemented using TAMs. The different ground colors are used by the robots to localize themselves in the environment. Three lights, marked with "L", provide directional information.

into arrays of four. The source and the nest TAMs are located in opposite corners of a rectangular environment surrounded by walls.[2] The cache TAMs are located between source and nest. Different areas of the environment are marked with a specific ground color, which can be perceived by the robots and used to determine their location in the environment and to navigate through the corridor. Three light sources, located in proximity of one of the walls, provide directional information used for navigation.

In ARGoS, we simulate the robots and the environment in two dimensions, using a dynamics physics engine. The camera allows an e-puck to perceive the LEDs of the TAM up to a distance of 1.3 m. A random value is added to the perceived distance, between -5% and 5% of the actual value, and a random value between $-2°$ and $2°$ is added to the perceived angle. 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-color sensor readings. The maximum communication distance of the simulated range and bearing device, used in the experiments presented in Section 5.6, is set

[2]The size of the environment varies in the specific experiments.

to 0.75 m. Each robot has a 30% probability per simulation step[3] of not receiving any message, even if other robots were sending messages within communication range.

Object abstraction using the TAM. Each TAM abstracts either a spot where an object can be deposited, or an object that can be taken by a robot. A TAM whose LEDs are lit up in green represents an object available at the TAM. A TAM whose LEDs are lit up in blue represents a free spot where an object can be deposited. Using this representation, when a robot enters a TAM whose LEDs are lit up in green, we assume that the robot takes an object from that TAM. Analogously, when a robot transporting an object enters a TAM whose LEDs are lit up in blue, we assume that the object being carried is deposited in that TAM. In both cases the TAM acknowledges the robot presence by temporarily turning the LEDs to red. The robots themselves keep track of whether they are carrying an object or not. A robot without an object assumes it has picked up one from a TAM if it perceives the TAM LEDs turning red and the LEDs were green before. Conversely, a robot carrying an object assumes it has dropped it inside a TAM if it perceives the TAM LEDs turning red and the LEDs were blue before.

The behavior of the TAMs changes with their location in the environment. The nest TAMs are always blue, representing unlimited space where objects can be stored. The source TAMs are always green, representing the fact that objects can always be found at the source.

Implementation of the cache. The behavior of the TAMs implementing the cache is more complex. Figure 5.4 depicts the behavior of two paired TAMs that implement a slot of the cache. At the beginning, the cache TAM oriented towards the source is lit up in blue, and the one oriented towards the nest have its LEDs off (Figure 5.4a). In this configuration the cache slot is empty. Once a robot has dropped an object in the cache, by subsequently entering and exiting a free (i.e., blue) TAM oriented towards the source, the TAM turns off so that it is no longer available to accept an object, and the corresponding TAM oriented towards the nest turns to green (Figure 5.4b). This represents an object deposited in the cache that becomes available on the nest side. A robot on the other side can pick up this object by subsequently entering and exiting the TAM oriented towards the nest (Figure 5.4c). Once the object has been picked up, the TAM oriented towards the nest turns off and the one oriented towards the source turns to blue,

[3]A step simulates a time of 0.1 seconds.

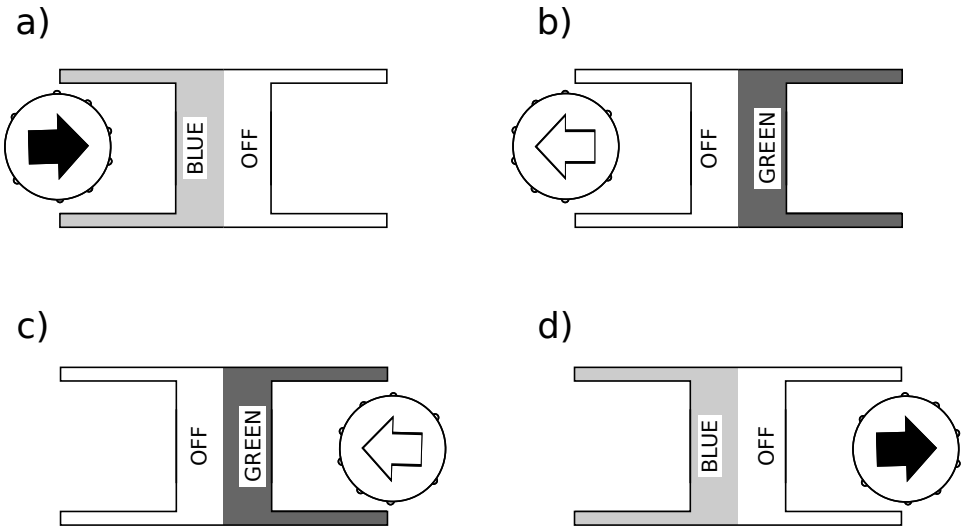

Figure 5.4: Implementation of a cache slot using two paired TAMs. The source (not represented) is located on the left-hand side, the nest (not represented) on the right-hand side. Robots carrying an object are marked with a black arrow, robots not carrying an object with a white arrow. **(a)** Initial 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 switches off the LEDs and the paired TAM oriented towards the nest 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 the robot leaves with the object, the two TAMs return to their initial configuration.

to signal that the TAM is available again for dropping an object (Figure 5.4d). A video illustrating the described behavior is available with the supplementary material (see Annex B).

The interfacing time. The TAM allows us to regulate the relative benefits of utilizing the cache over the corridor in an easy way. We do so by defining the value of the interfacing time Π: the lower the value, the more the cache is preferable to the corridor. In other words, the TAMs allow us to tune the degree at which using the interface is advantageous over bypassing it. In this way we can verify that, using the algorithms we propose, the swarm makes a good decision pertaining the usage of the interface. The interfacing time Π is implemented as a delay between the two phases represented in Figure 5.4a and Figure 5.4b. 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 does not leave the TAM) until a time equal to Π has passed. Analogously, the transition between the phases represented in Figure 5.4c and Figure 5.4d, requires a time Π.

5.3 Application of the Proposed Approach

In this section, we describe the application of the approach presented in Chapter 3 to the foraging scenario.

Estimation of costs. Each robot keeps a cost estimate for each of the possible four actions: i) HARVEST an object from the source (i.e., 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 (i.e., using the corridor). Each estimate $\hat{t_i}$ is computed as a recency-weighted average of the measured costs:

$$\hat{t_i} = (1 - \alpha)\ \hat{t_i'} + \alpha\ t_M\ , \tag{5.1}$$

where $\alpha \in (0, 1]$ is the weight factor, $\hat{t_i'}$ is the value of the estimate before the update, and t_M is the measured time required to perform the corresponding action i. The way t_M is measured depends on the estimate being updated (refer to Figure 5.5).

In all the cases, t_M measures the time taken between two subsequent decisions, made by the robot (see question marks in Figure 5.2). When the robot estimates the time $\hat{t_H}$ required to HARVEST an object from the source (Figure 5.5a), 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 (Figure 5.5b), 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 (Figure 5.5c) 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 (Figure 5.5d) 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.

Behavior of the robots. Figure 5.6 illustrates the behavior of each robot. Upon taking an object from the source, a robot decides whether

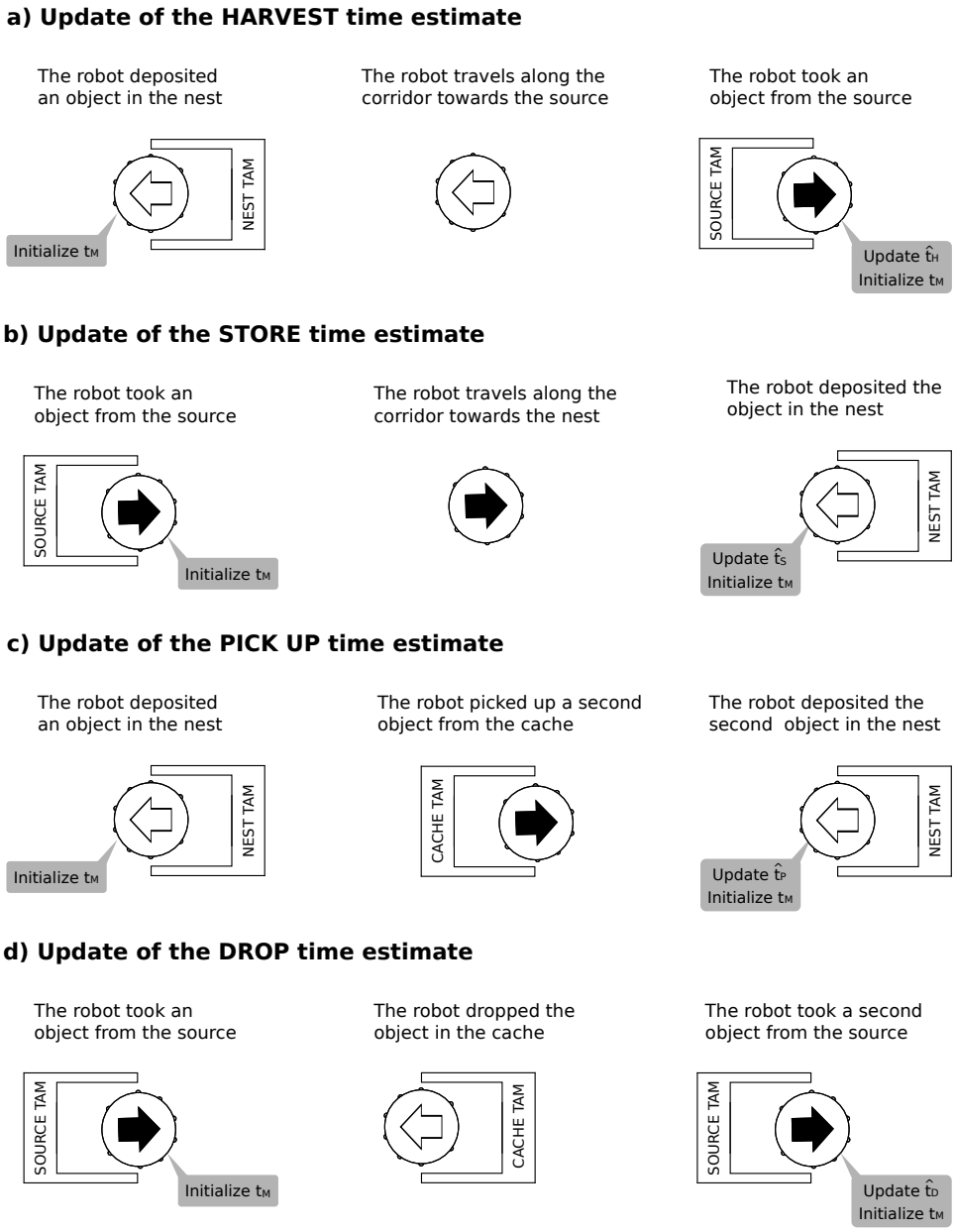

Figure 5.5: Update of the time estimates $\hat{t_i}$. **(a)** Harvest time estimate $\hat{t_H}$. **(b)** Store time estimate $\hat{t_S}$. **(c)** Pick up time estimate $\hat{t_P}$. **(d)** Drop time estimate $\hat{t_D}$. Robots carrying an object are marked with a black arrow, robots not carrying an object with a white arrow.

to drop the object in the cache or to take the corridor and store the object in the nest. Analogously, upon depositing an object at the nest, a robot decides whether to pick up an object from the cache or to

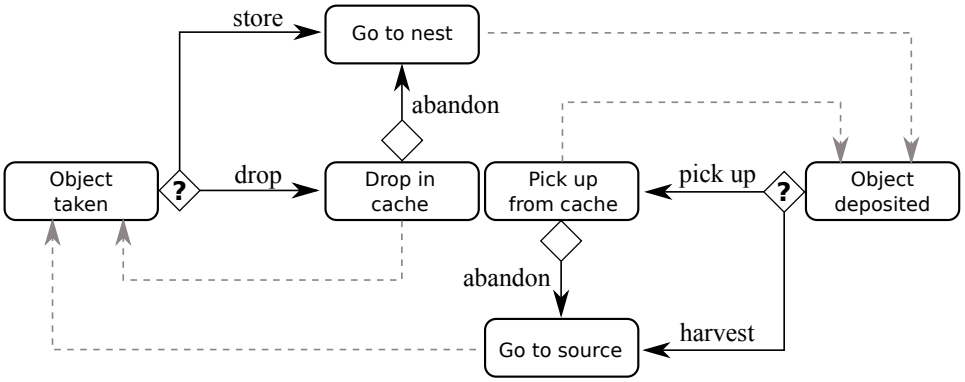

Figure 5.6: Finite state machine describing the high-level behavior of each robot. The small squares represent decision points, in which the robots are free to choose the action to perform. The corresponding choice is represented with a black continuous arrow marked with a label. Dashed gray arrows represent state transitions when the robot does not make a choice. The two squares labeled with a question mark represent the decision points at which the robot decides whether to use the interface or not. After taking an object from the source, the robot decides whether to DROP it in the cache, or STORE it at the nest. After depositing an object in the nest, the robot decides whether to PICK UP the following one at the cache, or HARVEST it from the source. The robot can abandon a previous decision of using the cache.

take the corridor and harvest an object in the source. These decisions are made on the basis of the perceived cost of each action, computed using Equation (5.1). Algorithms that the robots can use to make such decisions are presented in Section 5.4 and 5.5.

Besides selecting whether to use the cache or the corridor, the robots can also quit the decision of using the cache. An *abandoning* mechanism governs this choice and prevents two deadlock situations from happening. The first situation happens if all the robots are trying to drop objects in the cache and the cache is full. The second, dual situation happens if all the robots are trying to pick up objects from the cache, and the cache is empty. The abandoning mechanism is part of the algorithm employed by the robots to make decisions. Specific implementations of the mechanism are presented in Section 5.4.1 and Section 5.5.1, together with the corresponding algorithms.

5.4 The AdHoc Algorithm

In this section, we present an ad-hoc algorithm that the robots can use to select whether to employ or bypass the fixed interface. The algorithm was first proposed in Pini et al. (2011), but the results presented in this section are not the ones that were published in the article. This is due to the fact that the results published were obtained with an older version of ARGoS. We decided to reimplement the experiments with the version of ARGoS available at the moment of writing. With minor differences due to reimplementation, the results presented here confirm the original results of Pini et al. (2011) (reported in Annex A).

The rest of this section is organized as follows. In Section 5.4.1 we describe in detail the algorithm proposed in Pini et al. (2011), which we refer to as the *AdHoc* algorithm. In Section 5.4.2 we present experiments in which we test the performance, adaptiveness, and scalability of the algorithm.

5.4.1 The Algorithm

Robots employing the AdHoc algorithm stochastically choose between using the interface or not. Each robot has a probability P of using the cache (i.e., dropping or picking up an object) computed as:

$$P = \begin{cases} \left[1 + e^{-S\left(\left(\hat{t}_H+\hat{t}_S\right)/\left(\hat{t}_P+\hat{t}_D\right)-1\right)}\right]^{-1}, & \text{if } \hat{t}_H + \hat{t}_S > (\hat{t}_P + \hat{t}_D) \\ \left[1 + e^{-S\left(1-\left(\hat{t}_P+\hat{t}_D\right)/\left(\hat{t}_H+\hat{t}_S\right)\right)}\right]^{-1}, & \text{if } \hat{t}_H + \hat{t}_S \leq (\hat{t}_P + \hat{t}_D) \end{cases} \tag{5.2}$$

The probability of employing the cache is the same both for picking up and for dropping objects. Equation (5.2) defines the probability P of employing the cache on the basis of the ratio between the estimated time required to use the corridor $(\hat{t}_H + \hat{t}_S)$ and the cache $(\hat{t}_P + \hat{t}_D)$: the higher the ratio, the higher the probability. P grows with the value of the ratio following an "S" shaped curve, whose slope is determined by the value of the parameter S. The lower the value of S, the smaller the inclination of the curve, and the higher the exploration of the algorithm. Exploration consists in sampling the less-advantageous solution in order to detect variations in the environment that possibly made this solution more advantageous.

The cache abandoning mechanism employed in the AdHoc algorithm is also based on a stochastic decision. If a robot is trying to use the cache to drop an object, it can decide to abandon with a probability defined as:

$$Pa_D\left(w_t\right) = \left[1 + e^{\Theta\left(w_t,\hat{t}_D\right)}\right]^{-1}, \tag{5.3}$$

Table 5.1: Default values of the experimental parameters.

Parameter	**Default value**
Duration of an experimental run	60000 s
Experimental runs per setting	25
Swarm size	14 robots
Initialization of $\hat{t}_P$ and $\hat{t}_D$	uniform sampling in $[20\,\text{s}, 40\,\text{s}]$
Initialization of $\hat{t}_H$ and $\hat{t}_S$	uniform sampling in $[40\,\text{s}, 80\,\text{s}]$
Environment size	1.8 m by 3.0 m

where w_t is the current waiting time. $\Theta\left(w_t, \hat{t}_D\right)$ is computed as:

$$\Theta\left(w_t, \hat{t}_D\right) = K\left(\frac{w_t - \hat{t}_D}{\hat{t}_D + \hat{t}_P} + O\right) . \tag{5.4}$$

The function Θ compares the difference between the current waiting time w_t and the average drop time $\hat{t}_D$ to the total (estimated) time required to use the cache ($\hat{t}_D + \hat{t}_P$). The higher the ratio of the two, the higher the probability of abandoning the decision of using the cache. The parameter K defines the steepness of the probability curve defined by Equation (5.3), while O shifts the curve. If a robot abandons the choice of using the cache and its current value of w_t is greater than $\hat{t}_D$, the value of $\hat{t}_D$ is updated using Equation (5.1) (w_t replaces t_M in the formula). The probability Pa_P defining whether a robot abandons the choice of using the cache for picking up an object is computed using the same Equation (5.3) and replacing $\hat{t}_D$ with $\hat{t}_P$.[4]

5.4.2 Experiments and Results

In the following, we describe the experiments we run to test the AdHoc algorithm. The experiments are grouped into three sets. In the first set of experiments, we evaluate the performance of the algorithm and compare it to the performance of two reference algorithms. In the second set of experiments, we test the adaptivity of the algorithm with respect to changes in the environment. Finally, in the third set of experiments, we study the scalability of the algorithm.

All the experiments are carried out using ARGoS. Unless stated explicitly in the corresponding section, the parameters of the experiments are set as described below (also refer to Table 5.1).

Each experimental run lasts 60000 s; which amounts to a simulated time of 16 hours and 40 minutes. The simulation proceeds at steps

[4]In Equation (5.4), the denominator is still computed as $\hat{t}_D + \hat{t}_P$.

Table 5.2: Selected parameter values for the AdHoc algorithm.

Parameter	**Selected value**
Parameter S	2
Weight factor α	0.5
Parameter O	-2
Parameter K	-2

of 0.1 s. For each experimental setting we run 25 randomly seeded simulations. The size of the environment is 1.8 m by 3.0 m (horizontal and vertical dimensions, with respect to Figure 5.3). The swarm is composed of 14 simulated e-puck.

At the beginning of each experimental run, half of the swarm is positioned in the area containing the nest, the rest of the swarm in the area containing the source. The initial position and orientation of each robot are selected randomly. To avoid biases in the behavior of the robots, the estimates $\hat{t}_H$, $\hat{t}_S$, $\hat{t}_P$, and $\hat{t}_D$ are randomly initialized: $\hat{t}_H$ and $\hat{t}_S$ are uniformly sampled in $[40\,\text{s}, 80\,\text{s}]$; $\hat{t}_P$ and $\hat{t}_D$ in $[20\,\text{s}, 40\,\text{s}]$.

We select the values for the parameters of the algorithm using ad-hoc experiments. Details about the experiments and the complete results are available with the supplementary material (see Annex B). Table 5.2 reports the selected parameter values, which are utilized in all the experiments presented in the following.

Performance Evaluation

To assess the performance of the AdHoc algorithm, we compare it with the performance of two reference algorithms, referred to as *never-partition* and *always-partition* algorithms. The never-partition algorithm always uses the corridor, while the always-partition algorithm always uses the cache without abandoning. We test 7 different settings, each corresponding to a value of the interfacing time Π: 0, 5, 10, 20, 40, 80, and 160 seconds. The upper bound for the value of Π is selected so that the resulting cost of using the cache is considerably higher than the one of using the corridor.

The graph in Figure 5.7 shows the results of the experiments. The graph plots the average performance of the three algorithms for the different values of the interfacing time Π. The performance of an algorithm is given by the average number of objects delivered to the nest per robot. Each reference algorithm performs well only for a subset of values of Π. The always-partition algorithm performs better when Π is

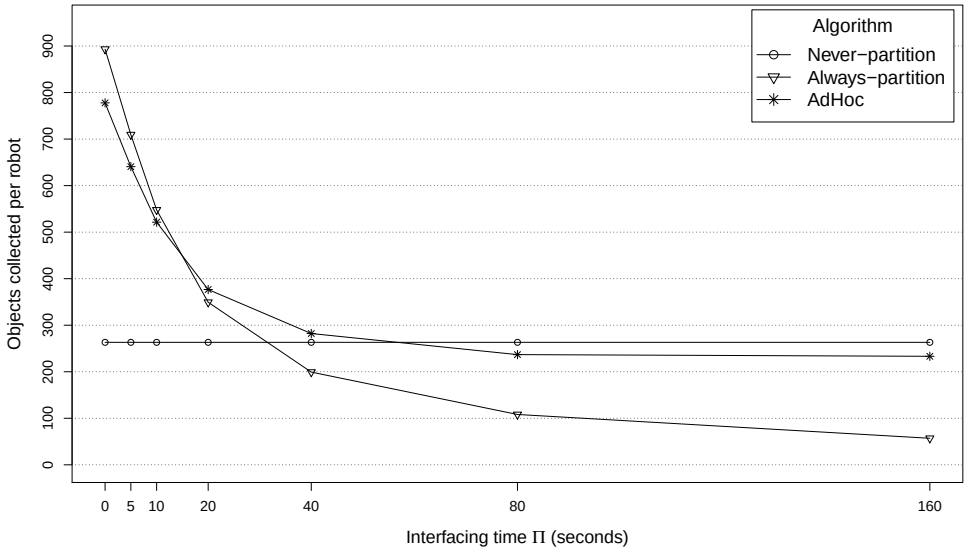

Figure 5.7: Average number of objects delivered to the nest per robot for different values of the interfacing time Π.

low. In this case, the time needed to travel along the corridor is much higher than the one to use the cache, therefore the corridor should be avoided and the object transportation task should be partitioned into sub-tasks. On the other hand, the never-partition algorithm performs better when the interfacing time is high. The AdHoc algorithm performs well across the whole spectrum of the parameter Π.

Figure 5.8 summarizes the strategy employed by the robots of the swarm when using the AdHoc algorithm, for the different values of Π. Each bar reports the percentage of times each action was performed by the robots; the reported values are averages computed over 25 runs. The graph confirms that the robots select the corridor with a higher frequency for increasing values of Π. For small values of Π, the preferred choice is the cache.

Figure 5.9 reports the time taken by the robots to use the corridor ($t_H + t_S$) and the cache (t_P and t_D) employing the three different algorithms, for interfacing times of 5 s (top), 20 s (center), and 40 s (bottom). For $\Pi = 5$ s the always-partition algorithm performs better than the never-partition algorithm (see Figure 5.7). Figure 5.9 (top) shows that the time required to pick up an object from the cache (t_P) is considerably lower than the one required to store an object at the nest via the corridor (which can reasonably estimated to be roughly the half of the value reported in Figure 5.9). Therefore the objects are delivered to the nest faster using the cache. Despite the fact that fewer

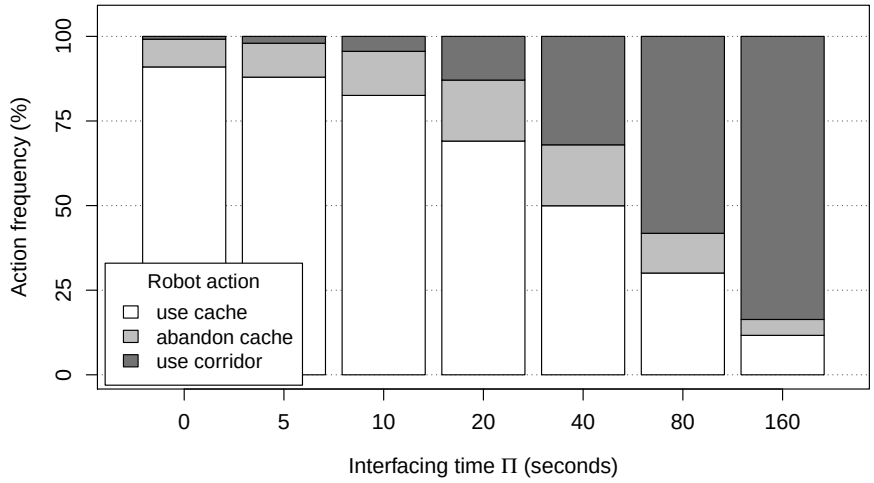

Figure 5.8: Actions performed by the robots employing the AdHoc algorithm, for different values of the interfacing time Π. Each bar reports, for a given value of Π, the percentage of times an action was performed. The actions reported are: selection and actual usage of the cache (white), selection of the cache and abandon (light gray), and selection of the corridor (dark gray). The percentage of times the robots chose to employ the cache is the sum of the values reported by the white and the light gray bars. The values reported are the averages, computed over 25 experimental runs.

robots are contributing to depositing objects in the nest (half of the swarm works in the area containing the source), the increased speed due to the usage of the cache makes the always-partition algorithm preferable to the never-partition algorithm. For increasing values of Π, the relative speed gain progressively decreases (Figure 5.9, center and bottom), and for $\Pi = 40\,\mathrm{s}$ always using the corridor becomes more advantageous than never using it (see performance in Figure 5.7).

Figure 5.9 shows that, using the AdHoc algorithm, the times needed to access the cache (t_P and t_D), for $\Pi = 20\,\mathrm{s}$ and $\Pi = 40\,\mathrm{s}$, are lower than the corresponding ones required by the always-partition algorithm. In fact, the robots employing the always-partition algorithm, all try to access the cache. This results in an increased competition for the cache, and therefore it is likely that a robot finds the cache busy when trying to use it. For large values of Π, a robot that finds the cache busy must wait a long time for a TAM to become available, since the robots using the cache spend more time inside the TAMs (as result of the high Π). This effect on the always-partition algorithm is highlighted in Figure 5.9 by an increasing variance of the pick up and drop times for an increasing value of Π.

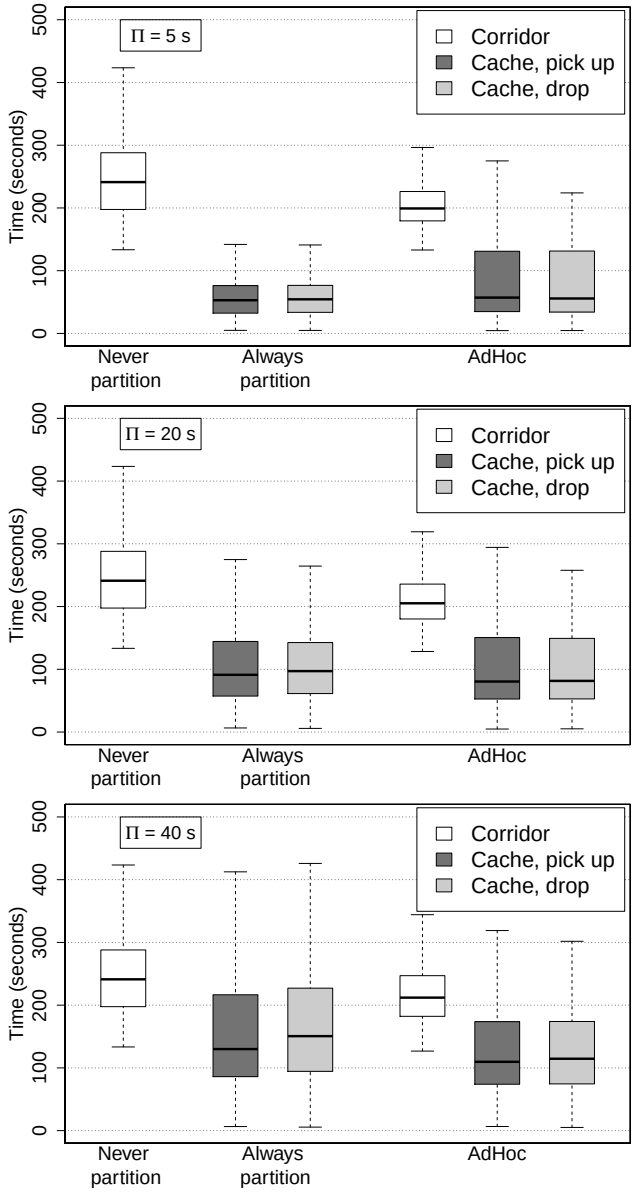

Figure 5.9: Time required to use the corridor ($t_H + t_S$) and the cache (t_P and t_D) for the three studied algorithms. The graphs report the data for $\Pi = 5\,\mathrm{s}$ (top), $\Pi = 20\,\mathrm{s}$ (center), and $\Pi = 40\,\mathrm{s}$ (bottom).

The AdHoc algorithm can reduce the competition for the cache TAMs when the cache is not advantageous by increasing the frequency at which the robots use the corridor (see actions reported in Figure 5.8). Additionally, the robots employing the AdHoc algorithm can benefit from the abandoning mechanism which prevents them from waiting

too long when the cache is not immediately available. Note that the fact that the AdHoc algorithm balances the usage of the cache and the corridor also reduces the time required to navigate through the corridor.

Figure 5.10 shows the evolution in time of the average number of robots dropping objects in the cache (left-hand side column) and picking up objects from the cache (right-hand side column) when the AdHoc algorithm is employed. The first row of plots reports the data collected for $\Pi = 5\,\mathrm{s}$, the second for $\Pi = 20\,\mathrm{s}$, and the third row for $\Pi = 40\,\mathrm{s}$. The gray area around the value of the mean reports the 95% confidence interval on the value of the mean. In each plot, the gray horizontal line marks the optimal number of robots that should work on each side of the cache (i.e., that maximize the total number of objects delivered to the nest). This number depends on the value of Π: in general, the lower the value, the higher the number of robots that should use the cache. The slanted gray lines indicate values of the number of robots working on each side of the cache that lead to a swarm performance that is at least 95% of the optimal.

To determine the optimal way of using the cache (i.e., the optimal number of robots working on each side), we performed experiments in which some of the robots were forced to always use the cache. For each value of Π, we exhaustively tested all the possible values of the number of robots that were forced to use the cache and recorded the corresponding performance. Figure 5.10 shows that the swarm employing the AdHoc algorithm is able to regulate the number of robots working on each side of the cache. In fact, in each moment in time, this number is close to the optimal value and therefore the cache is exploited efficiently.

Adaptivity to Changes

We test the adaptiveness of the AdHoc algorithm in response to a sudden variation of the environmental conditions. The variation consists in a change of the relative costs of the two strategies. In the experiments, we periodically switch the value of the interfacing time Π from 10 s to 80 s and back. The value changes every quarter of experiment (at the times $t1 = 15000\,\mathrm{s}$, $t2 = 30000\,\mathrm{s}$, and $t3 = 45000\,\mathrm{s}$). We test two cases: one in which the starting value of Π is 10 s and another in which it is 80 s. The swarm is expected to adapt the partition strategy to the value of Π.

Figure 5.11 reports the results of the experiments. The graph on top shows the data for the case in which the initial value of Π is 10 s; the graph at the bottom for the case in which the initial value of Π is 80 s.

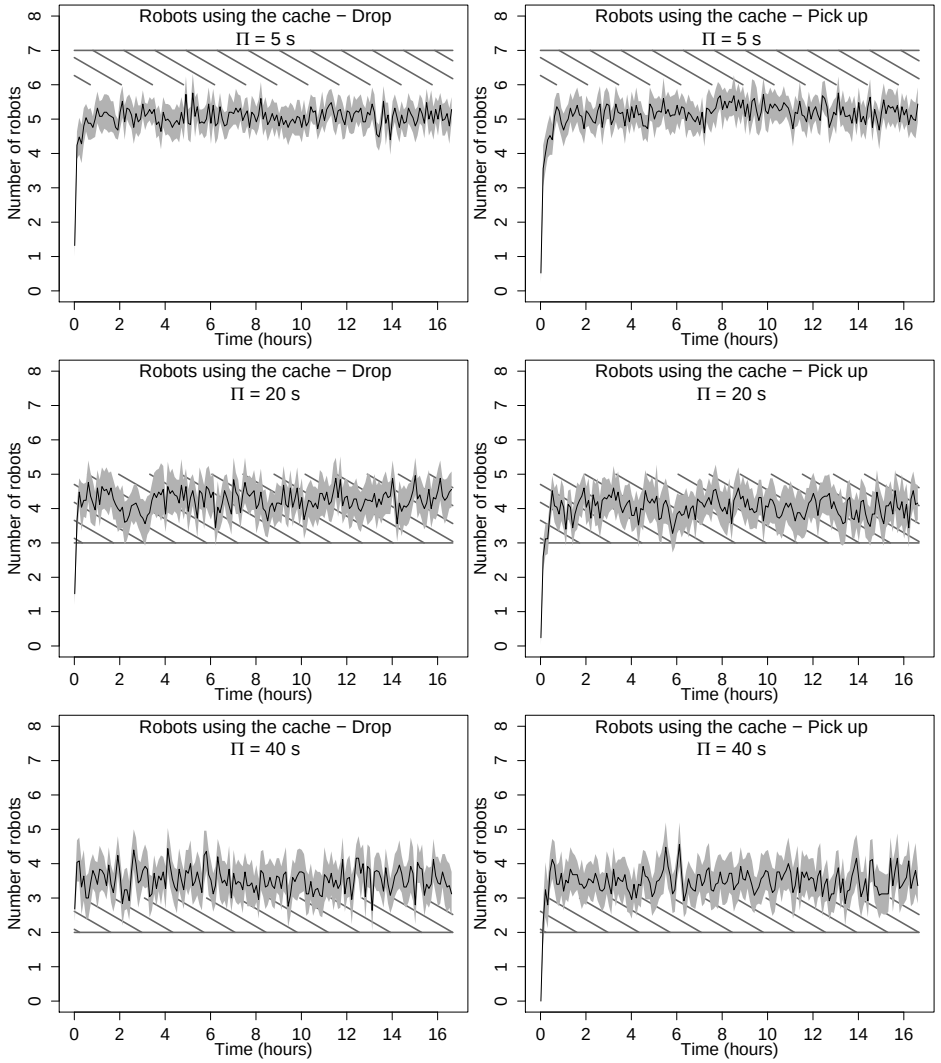

Figure 5.10: Average number of robots using the cache in time to drop (left-hand side plots) and pick up objects (right-hand side plots). The graphs report the data for $\Pi = 5\,\mathrm{s}$ (first row), $\Pi = 20\,\mathrm{s}$ (second row), and $\Pi = 40\,\mathrm{s}$ (third row). The gray area around the value of the mean reports the 95% confidence interval on the value of the mean. The gray horizontal line marks the number of robots that should work on each side of the cache in order to maximize performance. The gray slanted lines indicate values of the number of robots working on each side of the cache that result in a performance of at least 95% of the maximum.

In the graphs of Figure 5.11, the total time frame of the experiment is divided into windows of 60 minutes. Each box in the plot reports the percentage of usage of the cache in the time window preceding the

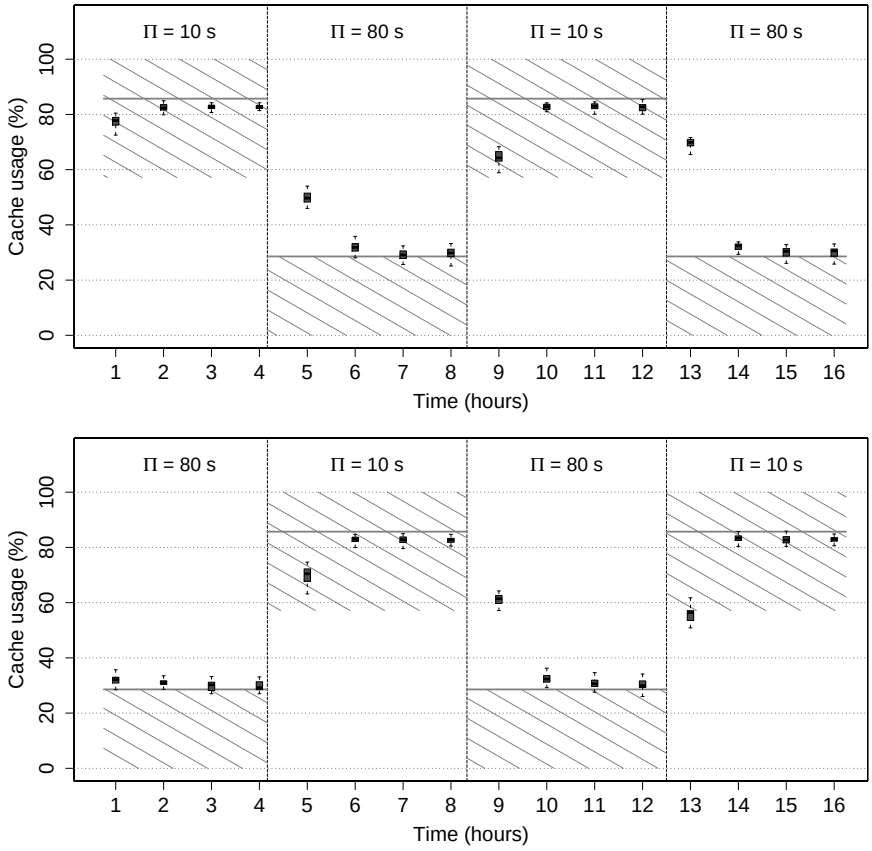

Figure 5.11: Percentage of usage of the cache in time. The cache interfacing time Π switches between the values of 10 and 80 seconds every quarter of experiment. Vertical dashed lines mark the instants at which the value of Π changes. The graph on top reports the case in which the initial value of Π is 10 s, the one at the bottom the case in which the initial value of Π is 80 s. Each box reports the percentage of usage of the cache in the hour preceding the time reported on the X axis. The gray horizontal line reports the optimal cache usage. The gray slanted lines report percentages of cache usage that lead to a number of objects delivered to the nest that is at least 95% of the maximum.

value indicated on the X axis. The gray horizontal segments in the figure report the optimal cache usage (i.e., the one that maximizes the number of objects delivered to the nest), which varies with the value of Π. The gray slanted lines report percentages of cache usage that lead to a performance that is at least 95% of the maximum.

The plots in Figure 5.11 show that the strategy employed by the robots changes in time, depending on the value of the interfacing time Π. Each time Π changes, the swarm converges to a new strategy more

suited to the new value of Π. In most of the cases, the new strategy employed by the swarm is close to the optimal. The results indicate that the AdHoc algorithm is not only able to utilize a near-optimal strategy, but it is also flexible with respect to changes occurring in the environmental conditions. Therefore it can be employed in situations where the benefits of using the interface vary in time.

Scalability

We test the scalability of the AdHoc algorithm and the two reference algorithms, for three different values of Π. The values chosen for Π are: 5, 20, and 40 seconds. The values correspond to a low, an intermediate, and a high value of the interfacing time, respectively. The swarm size is taken from the set $\{4, 6, 8, 10, 14, 18, 22, 26, 30\}$.

Figure 5.12 reports the results of the scalability experiments for $\Pi = 5\,\mathrm{s}$ (a), $\Pi = 20\,\mathrm{s}$ (b), and $\Pi = 40\,\mathrm{s}$ (c). The graphs report, for each algorithm, the total number of objects collected by swarms of different size.

The graphs highlight the negative effect of physical interference, that grows with the swarm size. In general, the benefits of adding robots to the swarm gradually decrease. The always-partition algorithm is affected the most by interference, since all the robots use the cache at the same time. The AdHoc and the never-partition algorithms are more scalable. For an increasing swarm size, the performance of the AdHoc algorithm is higher than the performance of both the reference algorithms, for the tested values of Π. The fact that the always-partition algorithm is less scalable than the other algorithms indicates that, in the studied setup, the competition to access the cache has an higher negative impact than the competition for space along the corridor. Indeed the area of the corridor is sufficiently large to allow the robots to navigate relatively easily, also in the cases in which the swarm is large.

Figure 5.13 summarizes the strategy employed by a swarm of 30 robots employing the AdHoc algorithm, for the different values of Π. Each bar reports the percentage of times that each action was performed by the robots, computed across 25 experimental runs. The data reported in figure suggests that the the performance of the AdHoc algorithm is higher than the one of the reference algorithms (refer to Figure 5.12) because the AdHoc algorithm balances the number of robots using the cache and the corridor. This reduces the competition to access the cache (thus the expected waiting time) and increases the navigability in the corridor with respect to the never-partition algorithm.

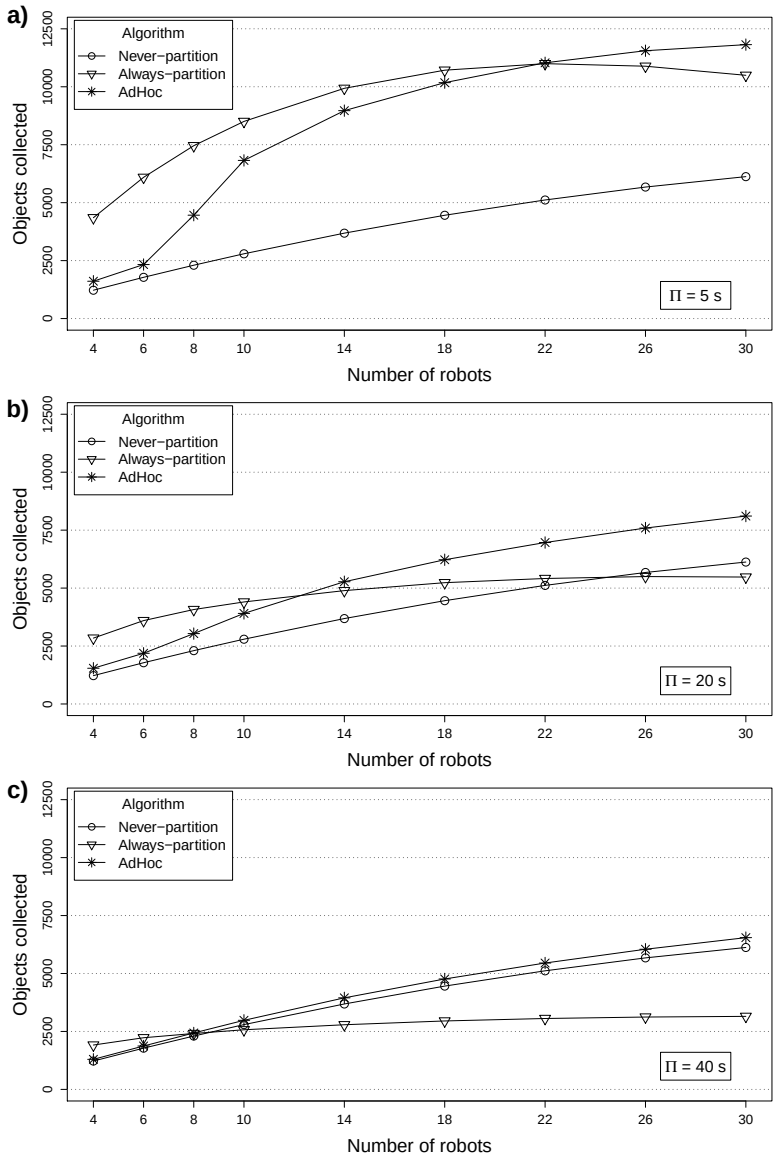

Figure 5.12: Results of the scalability experiment for **(a)** $\Pi = 5\,\mathrm{s}$, **(b)** $\Pi = 20\,\mathrm{s}$, and **(c)** $\Pi = 40\,\mathrm{s}$. The plots report, for each algorithm, the number of objects collected by swarms of different size.

However, the AdHoc algorithm does not perform better than the reference algorithms in all the tested conditions. In particular, the plot in Figure 5.12a shows that, for $\Pi = 5\,\mathrm{s}$ and in small swarms, the AdHoc algorithm does not perform well. As explained in the following, this is due to the fact that the AdHoc algorithm does not exploit the cache efficiently when the robots are too few.

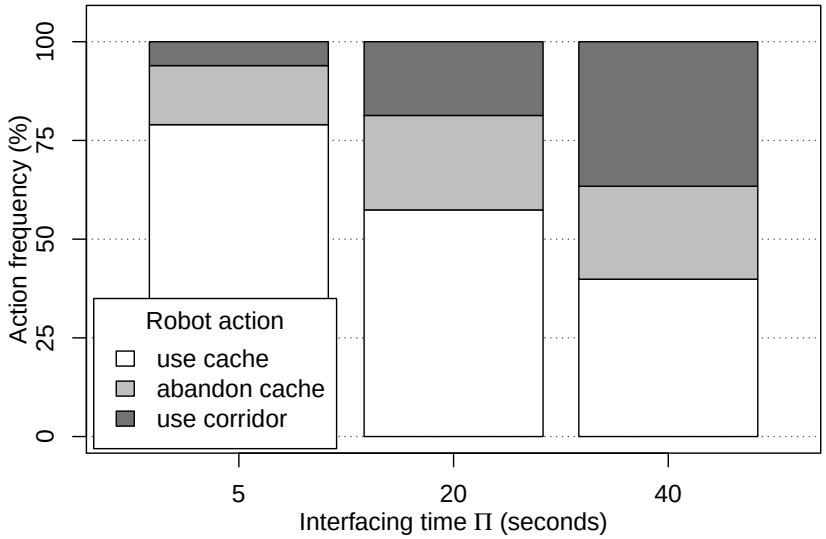

Figure 5.13: Actions performed by a swarm of 30 robots using the AdHoc algorithm, for different values of the interfacing time Π. Each bar reports, for each value of Π, the percentage of times an action was performed. The actions reported are: selection and actual usage of the cache (white), selection of the cache and abandon (light gray), and selection of the corridor (dark gray). The percentage of times the robots chose to employ the cache is the sum of the values reported by the white and the light gray bars. The values reported are averages computed over 25 experimental runs.

The plots in Figure 5.14 summarize the situation of the system at the moments at which one of the robots abandoned picking up an object from the cache. The data reported in the figure has been collected across the 25 experimental runs with a swarm of 4 robots employing the AdHoc algorithm. Each time a robot abandoned the choice of using the cache to pick up an object we recorded the number of objects available in the cache and the number of robots that were dropping objects (i.e., that were working on the other side of the cache). The main plot in Figure 5.14 is the histogram of the number of objects available at the cache when the robot abandoned. The smaller plots at the bottom are histograms of the number of robots that were on the other side of the cache. Each of these histograms reports a subset of the data, corresponding to a given number of objects available at the cache. Thus, the first plot on the left indicates the number of robots that were dropping objects in the cache when a robot abandoned picking up and the cache contained no objects. The second plot indicates analogous data, but for the cases in which there was one object available in the cache. The third plot for the cases in which there were two objects in

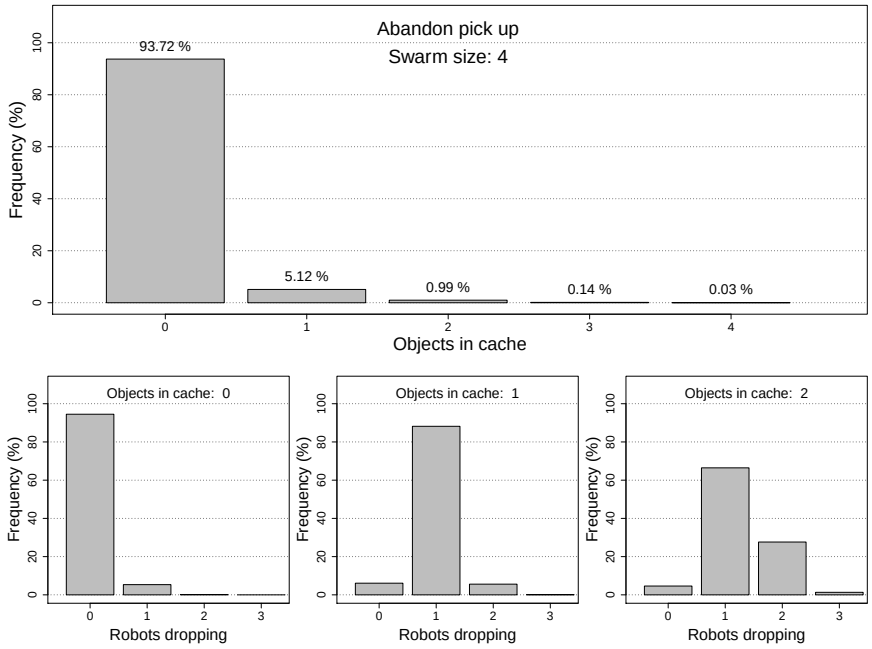

Figure 5.14: Robots abandoning the decision of picking up an object from the cache. The plots report the overall data collected in 25 runs with swarms of 4 robots employing the AdHoc algorithm. Main plot: frequency at which a given number of objects were in the cache when a robot abandoned. Small plots: frequency at which a certain number of robots were on the other side of the cache at the moment the robot gave up.

the cache. Analogous plots are reported in the Figure 5.15, for swarms of 6 robots, in Figure 5.16, for swarms of 8 robots, and in Figure 5.17, for swarms of 10 robots. Similar plots, reporting the data collected when the robots abandon dropping an object in the cache are available with the supplementary material (see Annex B). The results reported in the supplementary material are analogous to the ones presented here.

The AdHoc algorithm does not exploit efficiently the cache, in case the swarm is composed of few robots (less than 10). In the majority of the cases, the robots abandon picking up objects from the cache because the cache is empty (see main histograms). The cache empties because on the other side, few robots are dropping objects (see smaller histograms). The dual situation happens in the cases in which the robots abandon the decision of dropping objects in the cache: the cache gets full and few robots are on the other side picking up objects and freeing slots in the cache. Figure 5.17 shows that increasing the number of robots allows the swarm to exploit the cache more efficiently: the cache is less likely to get empty and the swarm performs better, as

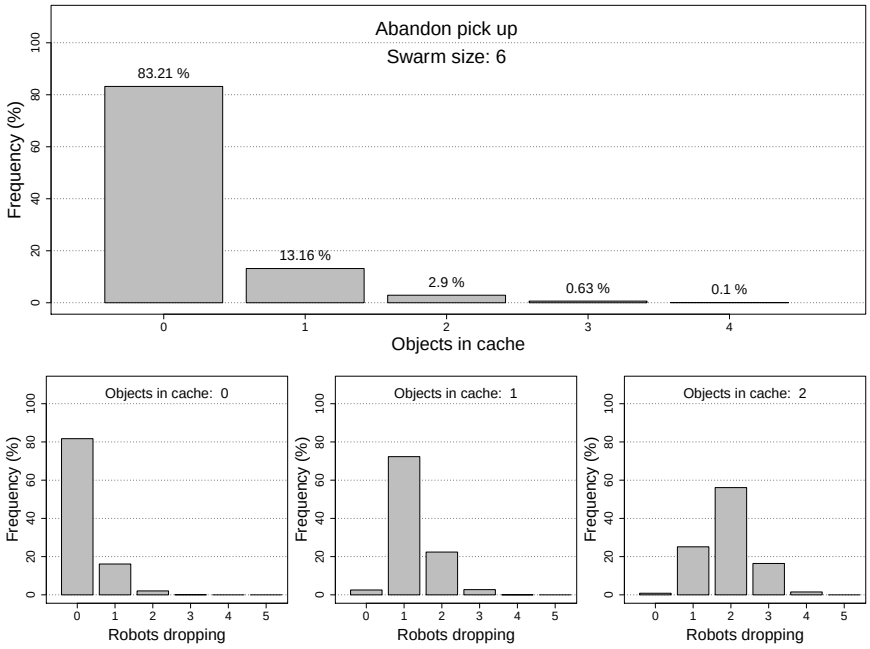

Figure 5.15: Robots abandoning the decision of picking up an object from the cache. The plots report the overall data collected in 25 runs with swarms of 6 robots employing the AdHoc algorithm. Main plot: frequency at which a given number of objects were in the cache when a robot abandoned. Small plots: frequency at which a certain number of robots were on the other side of the cache at the moment the robot gave up.

reported in Figure 5.12.

5.5 Task Partitioning as a Bandit Problem

In this section we show that the problem of selecting whether to utilize an interface, and therefore whether to partition a given task or sub-task, can be seen as a multi-armed bandit problem (Sutton and Barto, 1998; Cesa-Bianchi and Lugosi, 2006). In the multi-armed bandit problem, a player chooses in a sequence of trials one of the arms of a multi-armed bandit and receives a certain reward associated with the chosen arm. The goal of the player is to maximize the reward obtained in the sequence of trials. Within the approach presented in this dissertation, deciding whether to utilize an interface induces an analogous problem. The robots must perform transportation and they have to decide whether to employ the interface or an alternative route. The goal of the robots, in this case, is minimizing the costs associated to performing tasks, instead of maximizing rewards.

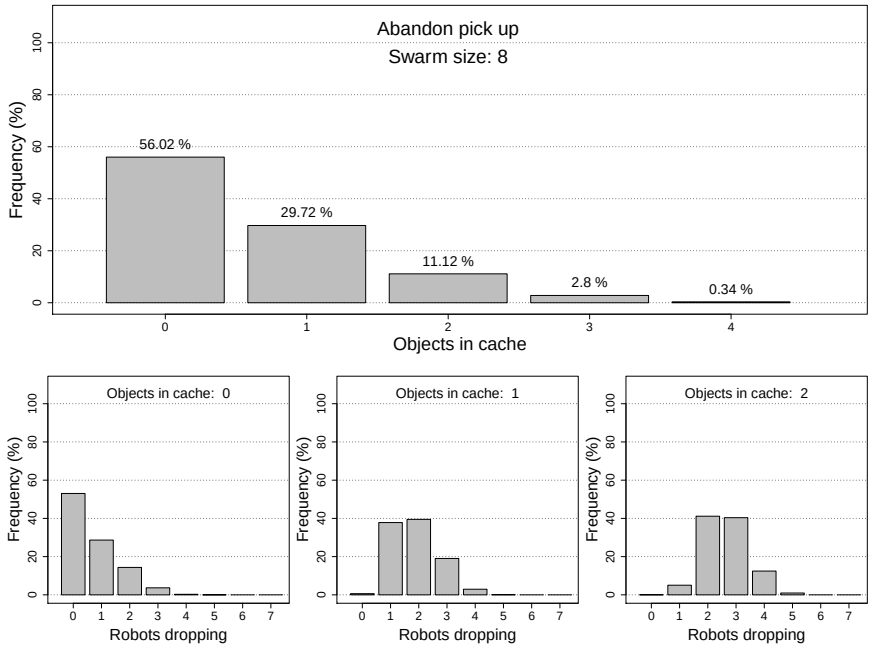

Figure 5.16: Robots abandoning the decision of picking up an object from the cache. The plots report the overall data collected in 25 runs with swarms of 8 robots employing the AdHoc algorithm. Main plot: frequency at which a given number of objects were in the cache when a robot abandoned. Small plots: frequency at which a certain number of robots were on the other side of the cache at the moment the robot gave up.

The advantage of seeing task partitioning as a multi-armed bandit problem stems from the fact that the latter is a widely studied problem. Consequently, its theoretical properties are well understood and, most importantly, one can use already existing algorithms, without the need of implementing ad-hoc solutions.

In this section, we test the applicability of existing algorithms, that have been proposed for the multi-armed bandit problem, to the problem of selecting whether to utilize an interface. In Section 5.5.1, we illustrate the algorithms considered in our study. In Section 5.5.2, we present the experiments that we carried out to test the algorithms.

5.5.1 Studied Algorithms

The algorithms. We test two algorithms, the ε-*Greedy* and the *UCB*, which have been proposed in the reinforcement learning literature and

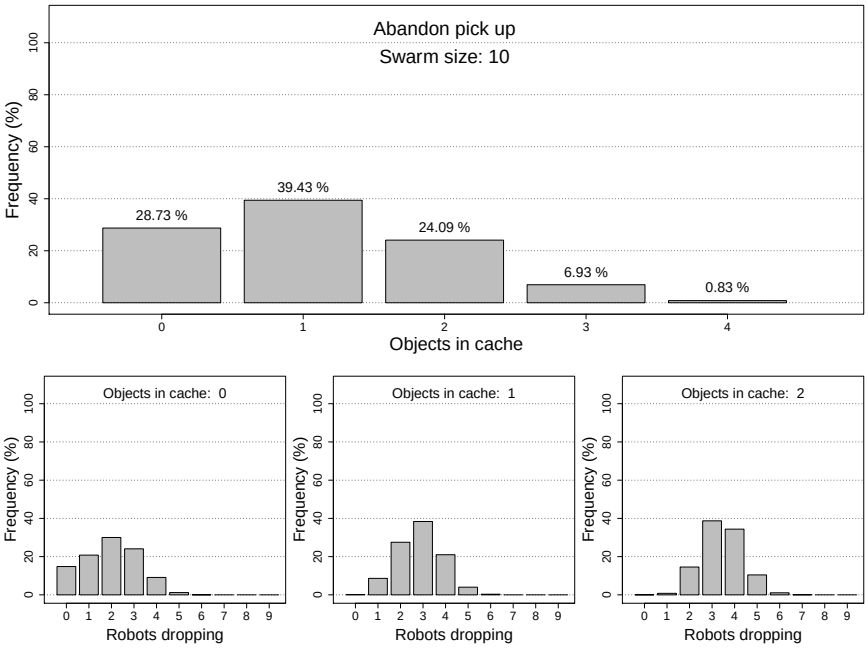

Figure 5.17: Robots abandoning the decision of picking up an object from the cache. The plots report the overall data collected in 25 runs with swarms of 10 robots employing the AdHoc algorithm. Main plot: frequency at which a given number of objects were in the cache when a robot abandoned. Small plots: frequency at which a certain number of robots were on the other side of the cache at the moment the robot gave up.

have been employed to tackle the multi-armed bandit problem.[5]

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 estimate $\hat{t}_i$ and with a probability ε a random action. ε is the only parameter of the algorithm and it defines the degree of exploration: the higher the value of ε, the higher the exploration.

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 was originally designed for stationary problems and it is characterized by a rapid convergence (Auer et al., 2002). The robots employing the UCB algorithm deterministically select the action to perform. For example, after de-

[5]The contents of this section are based upon the work described in Pini et al. (2013b). In Pini et al. (2013b), we also tested the state-of-the-art algorithm *Exp3* (see Cesa-Bianchi and Lugosi (2006))). Since the algorithm does not perform well in any of the tested conditions, we decided not to report its results here.

positing an object in the nest, a robot PICKS UP the following one from the cache if:

$$\hat{t}_P - \gamma\sqrt{\frac{2\ln(n_P + n_H)}{n_P}} < \hat{t}_H - \gamma\sqrt{\frac{2\ln(n_P + n_H)}{n_H}}\ , \qquad (5.5)$$

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., using the corridor). γ is a parameter that defines the degree of exploration: higher values of γ correspond to a higher exploration. The robots use a formula analogous to Equation (5.5) to decide whether to DROP and object in the cache or STORE it in the nest. Each of the two expressions of Equation (5.5) can be interpreted as an upper bound of a confidence interval (refer to Auer and Ortner (2010) for more details).

In addition to the two algorithms described, in the experiments we also test the AdHoc algorithm proposed in Pini et al. (2011) and presented in Section 5.4.1. Notice that there is a difference between the ε-Greedy and UCB algorithms and the AdHoc algorithm. In the AdHoc algorithm, no distinction is made between the two decision points: both at the nest and at the source there is the same probability of using the interface, defined by Equation (5.2). Using the UCB and the ε-Greedy algorithms, the robots discriminate between the two cases when making their choice.

We also test four reference algorithms, in which the decisions do not depend on cost estimates. These algorithms are used as a yardstick to evaluate the performance of the other algorithms. 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. The third reference algorithm is the *random* algorithm. Using this algorithm, the robots select the actions to perform stochastically, with equal probability. Finally, the *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 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.

The abandoning mechanism. The abandoning mechanism that allows a robot to quit the decision of using the interface is implemented with a timeout. The robot measures the time it has been trying to access the interface and abandons its choice when this time exceeds a given threshold. Compared to the abandoning mechanism governed by Equation (5.3), originally proposed in Pini et al. (2011), using a timeout-based mechanism reduces the number of parameters of the algorithms. To allow for a fair comparison of the algorithms, in the experiments presented in Section 5.5.2, we employ a timeout-based abandoning mechanism also for the AdHoc algorithm.

We utilize two methods to compute the value of the timeout threshold. The specific method utilized to compute the threshold depends on the algorithm being employed (details are given in Section 5.5.2). The first method consists in computing two timeout thresholds, τ_P and τ_D. The first is used when a robot is trying to pick up an object from the cache, the second when a robot is trying to drop an object in the cache. The two thresholds are computed as follows:

$$\tau_P = 3\hat{t_P}, \quad \tau_D = 3\hat{t_D} \ . \tag{5.6}$$

The second method uses a single threshold τ, derived using a formula analogous to the one used to compute the estimates $\hat{t}_i$ (Equation (5.1)). 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 $\tilde{t}$:

$$\tilde{t} = (1 - \alpha)\ \tilde{t} + \alpha\ t_M \ , \tag{5.7}$$

α is the same weight factor used in Equation (5.1) to compute 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:

$$\tau = 3\tilde{t} \ . \tag{5.8}$$

5.5.2 Experiments and Results

We carry out all the experiments in the environment represented in Figure 5.3. The size of the environment is 1.8 m (horizontal) by 4.4 m (vertical). We perform simulation-based experiments with swarms 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. The cost estimates are initialized stochastically: we uniformly sample $\hat{t}_P$ and $\hat{t}_D$ from the interval $[20\,\text{s}, 40\,\text{s}]$ and $\hat{t}_H$ and $\hat{t}_S$ from the interval $[40\,\text{s}, 80\,\text{s}]$. Each experimental run lasts 10 simulated hours. For each experimental setting we perform 100 runs, varying the

Table 5.3: Values of the experimental parameters.

Parameter	**Default value**
Duration of an experimental run	10 simulated hours
Experimental runs per setting	100
Swarm size	20 robots
Initialization of $\hat{t}_P$ and $\hat{t}_D$	uniform sampling in $[20\,\text{s}, 40\,\text{s}]$
Initialization of $\hat{t}_H$ and $\hat{t}_S$	uniform sampling in $[40\,\text{s}, 80\,\text{s}]$
Environment size	1.8 m by 4.4 m

Table 5.4: Selected parameters for the exploiting and exploring versions of the studied algorithms.

Algorithm	**Version**	**Parameter**	**Abandoning**
AdHoc	exploiting	$S = 6.0$	Equation (5.8)
	exploring	$S = 1.0$	Equation (5.8)
UCB	exploiting	$\gamma = 100$	Equation (5.8)
	exploring	$\gamma = 1000$	Equation (5.6)
ε-Greedy	exploiting	$\varepsilon = 0.01$	Equation (5.6)
	exploring	$\varepsilon = 0.11$	Equation (5.6)

seed of the pseudo-random number generator. Table 5.3 reports the values of the experimental parameters.

We select the values for the parameters of the algorithms and the formula used to calculate the timeout threshold on the basis of ad-hoc experiments. Details about these experiments and their complete results are available with the supplementary material (see Annex B). On the basis of the results of these experiments, we select two parameter settings for each algorithm, one corresponding to an *exploring* version and one to an *exploiting* version of the algorithm. Table 5.4 summarizes the parameter settings of each algorithm, which are utilized in the experiments described in the following.

We perform two set of experiments. In the first set of experiments, we test the algorithms in a setup where the environmental conditions do not change in time. In the second set of experiments, we study the behavior of the algorithms in the case in which the environmental conditions vary over time.

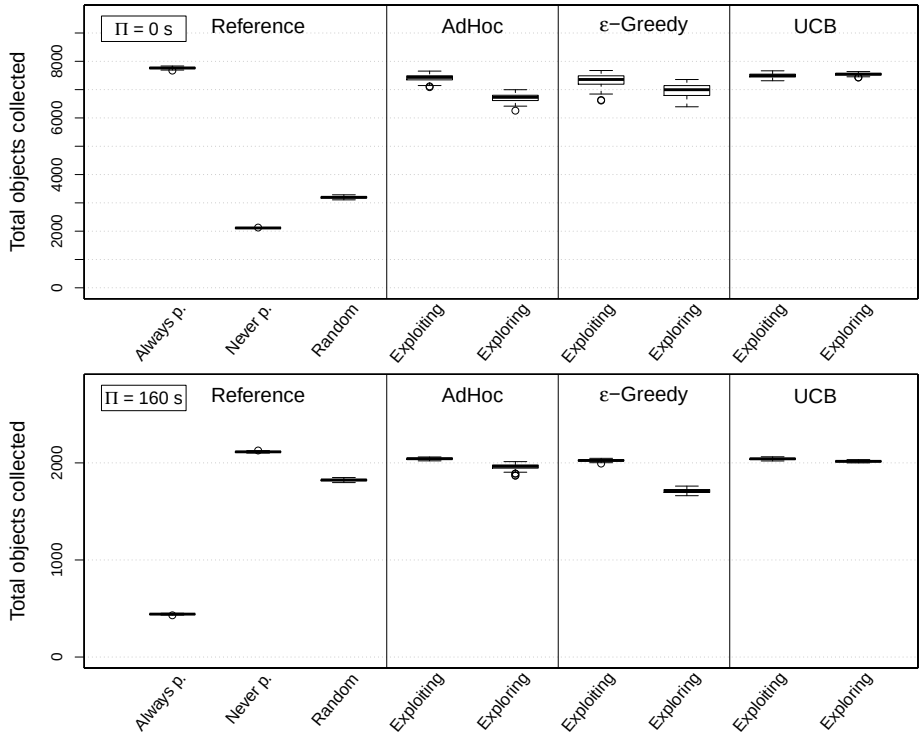

Figure 5.18: Performance of the studied algorithms, measured as objects delivered to the nest by the swarm. The top plot reports the results obtained for $\Pi = 0\,\text{s}$, the bottom plot for $\Pi = 160\,\text{s}$.

Stationary Environmental Conditions

In the experiments described in this section, we test the capability of the swarm to decide whether to use the cache or the corridor for two values of the interfacing time. In one case, we set Π to 0 s, value that renders the cache preferable over the corridor; in the other, we set Π to 160 s, which renders the corridor preferable.

Figure 5.18 reports the performance of the studied algorithms, for $\Pi = 0\,\text{s}$ (top) and $\Pi = 160\,\text{s}$ (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 $\Pi = 0\,\text{s}$, the cache is advantageous and the best performing algorithm is the always-partition algorithm. Dually, for $\Pi = 160\,\text{s}$, the corridor is advantageous and the never-partition algorithm is the best performing. 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 to tackle the multi-armed bandit problem can be successfully applied to the problem of selecting

whether to utilize an interface and partition a given task.

Since 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.[6] In fact, once the robots have determined whether the cache is advantageous over the corridor or not, they do not need to explore the less-advantageous option anymore and the exploiting version of the algorithms is more efficient.

Non-stationary Environmental Conditions

In the experiments described in this section, we study a non-stationary environment in which we vary Π during the course of the experimental run. We test two cases; in one case the interfacing time is initialized to $\Pi = 0\,\text{s}$ and, at time $t = 2.5\,\text{hours}$ (i.e., one quarter of experiment), we set it to 160 s. Therefore, the cache is initially preferable (for $t < 2.5\,\text{hours}$). 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 0 s at $t = 2.5\,\text{hours}$. 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 take the corridor. The informed algorithm is used as an upper bound for the other algorithms.

Figure 5.19 reports the performance of the two versions of each algorithm and the four reference algorithms, for the case in which Π varies from 0 s to 160 s. The results reported in the figure show that

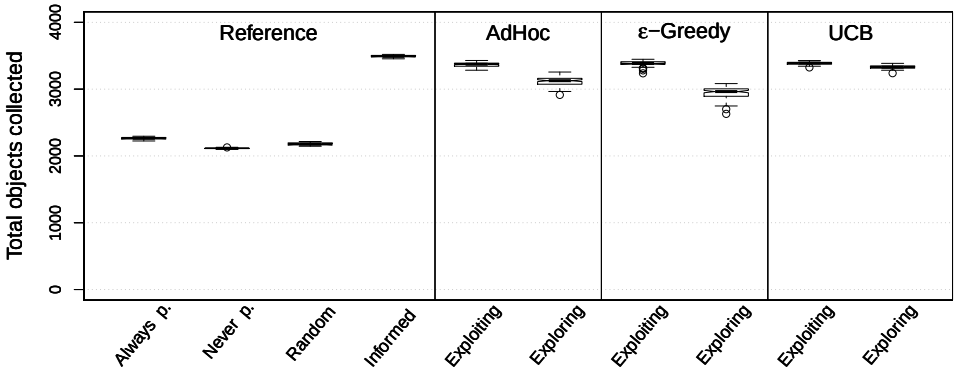

Figure 5.19: Performance, measured as objects delivered to the nest by the swarm. The figure reports the results obtained for the experiments in which Π is initialized to 0 s and set to 160 s when the experimental run reaches a quarter of its duration.

[6] With the exception of UCB, in which the difference between the two versions is minimal.

the exploiting version of each algorithm performs better than the corresponding exploring version and reaches a performance close to the one of the informed algorithm.

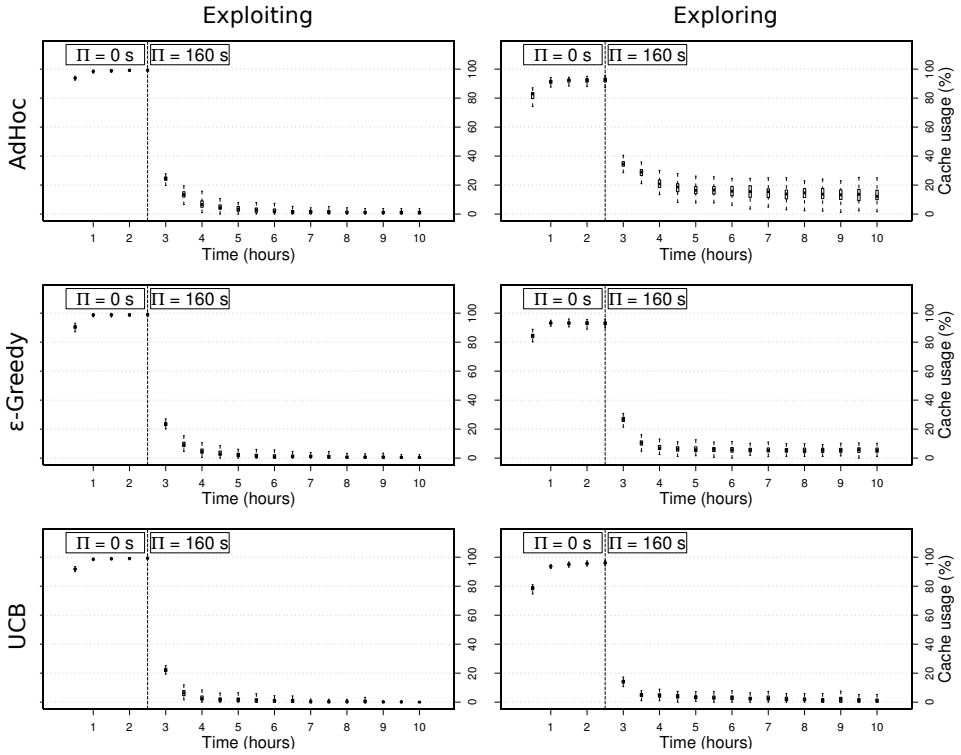

Figure 5.20: Percentage of usage of the cache in time, for the different versions of the algorithms. The plots report the results of the experiments in which Π is initialized to 0 s. Each box reports the data collected over 100 experimental runs, for the period of 30 minutes that precedes the value reported on the X axis. The vertical dashed line marks the instant in which the value of Π is changed.

Figure 5.20 reports the percentage of use of the cache in time for the three studied algorithms. The plots in the same row refer to the same algorithm, from top to bottom: AdHoc, ε-Greedy, and UCB. 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 preceding 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 time. 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 Figure 5.19. 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) 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.

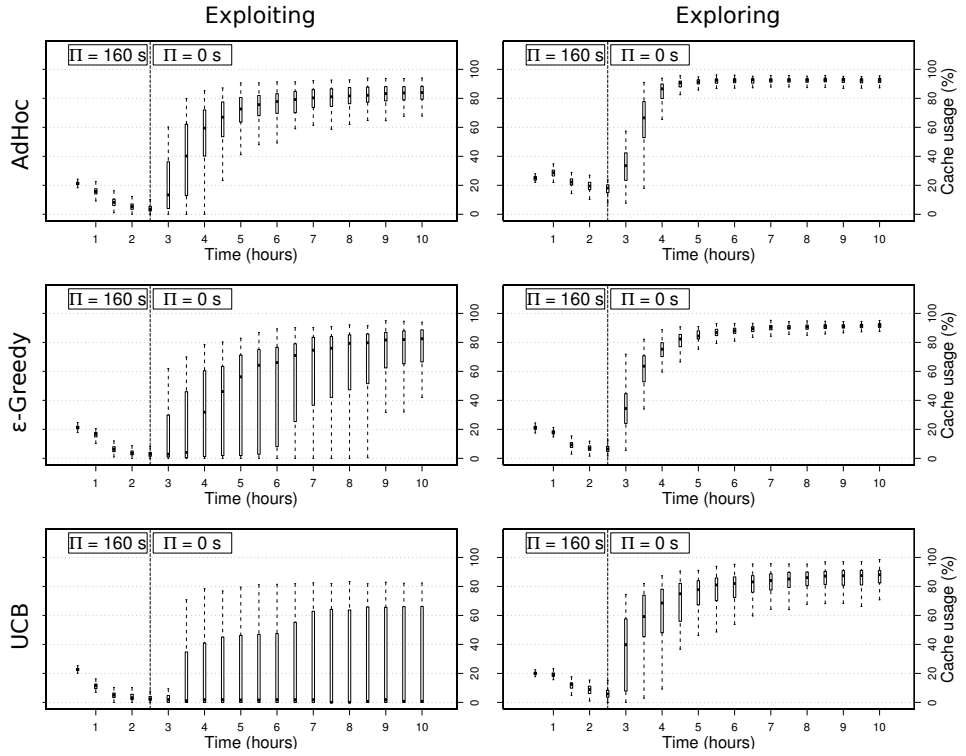

Figure 5.21: Percentage of usage of the cache in time, for the different versions of the algorithms. The plots report the results of the experiments in which Π is initialized to 160 s. Each box reports the data collected over 100 experimental runs, for the period of 30 minutes that precedes the value reported on the X axis. The vertical dashed line marks the instant in which the value of Π is changed.

Figure 5.21 reports, analogously to Figure 5.20, the percentage of use 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 se-

lect the corridor more frequently than the cache. This implies that, differently from the previous case, the variation occurring at the cache cannot be detected directly by all the robots in the swarm. The first column of plots in Figure 5.21 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.

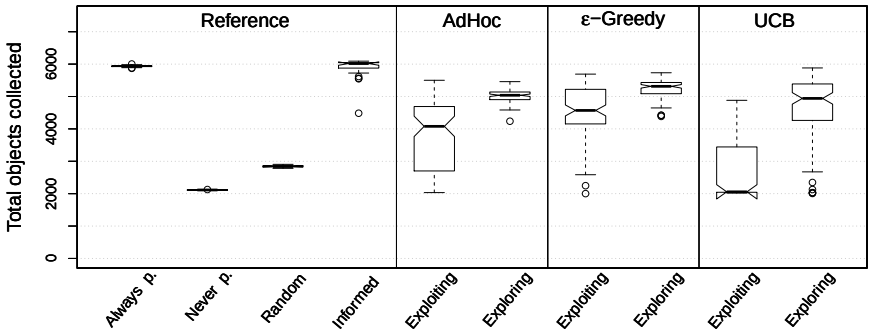

Figure 5.22: Performance of the algorithms for the case in which Π is initialized to 160 s and set to 0 s when the experimental run reaches a quarter of its duration.

Figure 5.22 reports the performance of the algorithms and 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 problem of selecting whether to use an interface it exists a tradeoff between exploiting the cumulated knowledge and exploring options perceived as less advantageous to detect changes and react to them.

5.6 The Use of Communication

In this section we extend the work presented in Section 5.5, by adding explicit communication to the system. The robots exchange information about the environment and utilize the information they receive to integrate their knowledge. In Section 5.6.1, we describe how communication is implemented in the studied system. In Section 5.6.2, we present the experiments that we perform to test the effects of communication.

5.6.1 The Communication Protocol

In the experiments, each robot communicates the time t_A associated to the last action A 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 (5.1) is utilized to update the estimate using the information received (A identifies the index i and t_A replaces t_M).

In our experiments, the e-pucks communicate using the infrared range and bearing board. This communication device imposes a number of 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 at most one message per control step. Third, each message has a payload limited to 16 bits.

As mentioned, a robot communicates the measured time t_A associated to the last performed action A. This information is encoded in the 16 bits payload in the following way. The first 2 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.[7] 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[8] policy: if a new message must be added 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.

5.6.2 Experiments and Results

In the experiments presented in this section, we use the same experimental setup and algorithm settings presented in Section 5.5. We compare each algorithm along an additional dimension, which is whether

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

the algorithm uses communication or not. We refer to the first case as the *social* version of an algorithm and to the second case as the *non-social* version.

In the following two sections we illustrate experiments in which we compare the social and the non-social versions of the algorithms and we propose a modification to the AdHoc and UCB algorithms. This modification has been introduced to tackle an issue that emerged in the experiments performed with the social version of the two algorithms.

The Effect of Communication

We focus here 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.

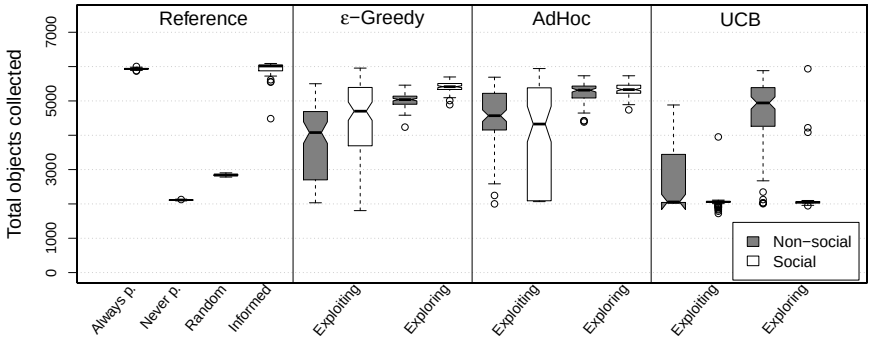

Figure 5.23: Performance of the studied algorithms for the case in which Π is initialized to 160 s and set to 0 s when the experimental run reaches a quarter of its duration. For each algorithm (excluding the reference algorithms) we report the results obtained with the non-social (gray boxes) and social version (white boxes).

Figure 5.23, reports the performance of the algorithms. Each plot reports the performance of the four reference algorithms and the four different versions of the studied 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 shows 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. 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 increases slightly, while there is an increase in the variability of the results obtained with the exploiting version.

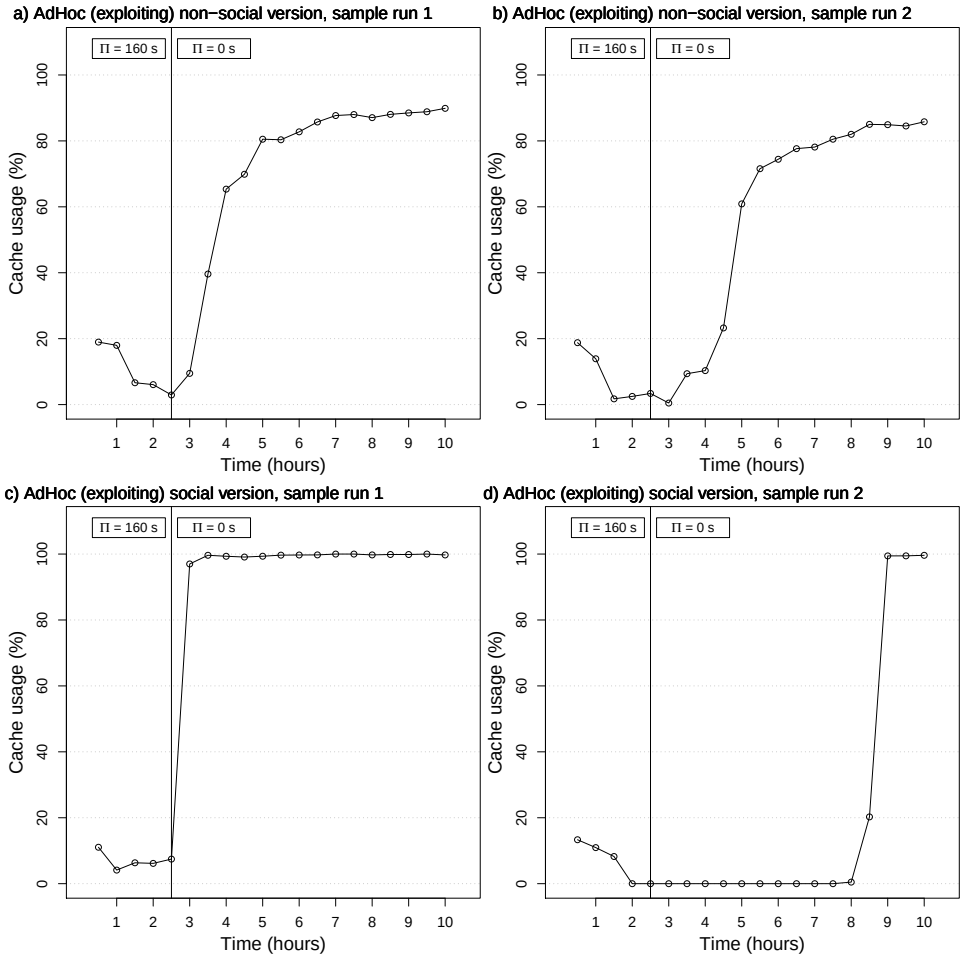

Figure 5.24: Cache usage in time, as observed in 4 selected experimental runs. Top: non-social version of the AdHoc algorithm (exploiting). Bottom: social version of the AdHoc algorithm (exploiting).

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 Figure 5.24, 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, while the plots in the second row report the ones obtained with the social version.

The runs reported here are examples that allow us to illustrate general trends that we observe in the experiments. Analogous plots, reporting the data of each experimental run, are available with the supplementary material (see Annex B). 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 Figure 5.24c and Figure 5.24d, the transitions are sharper. If a few robots detect that the cache became advantageous, information spreads within the swarm and 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 (Figure 5.24a and Figure 5.24b), 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 Figure 5.24d, 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 the 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 Figure 5.23). This is likely due to the fact that, as mentioned in Section 5.5.1, 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, the ε-Greedy does not suffer the mentioned problem. This is due to the fact that the probability of selecting the action perceived as the less advantageous does not change with the estimated action cost, being always ε. Therefore, the ε-Greedy draws only benefits from the increased flow of information.

Algorithms with ε-exploration

In order to face the problems we encountered with the AdHoc and UCB algorithms, we modify the two 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 refer to the modified versions of the AdHoc and UCB algorithms as the ε-AdHoc and ε-UCB algorithms, respectively. We select the value $\varepsilon = 0.01$, which is the same utilized in the exploiting version of the ε-Greedy algorithm.

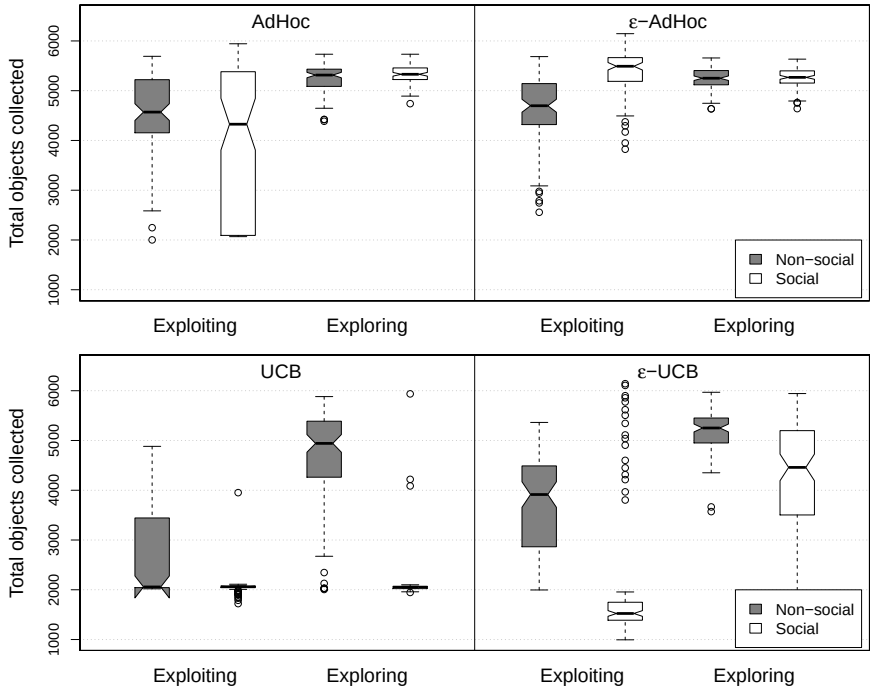

Figure 5.25: Performance of the UCB (top) and AdHoc (bottom) algorithms, without (left) and with ε-exploration (right). The gray boxes report the data for the case in which communication is not used, the white boxes for the case in which communication is used.

Figure 5.25 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 exploit knowledge, which allows the swarm to use efficiently the cache (or the corridor) and, at the same time, to remain flexible with respect to changes.

5.7 Summary

In this chapter we focused on the case in which a fixed interface limits the amount of work that a robot can contribute to a task. In this case, the choices available to a robot are either use the interface or

bypass it. As mentioned in Chapter 3, the capability of autonomously deciding for one of the two options is a fundamental building block for the implementation of complex partitioning strategies.

We studied a foraging scenario in which the robots must decide whether to employ an interface and partition the transportation task, or to bypass the interface and perform transportation as an unpartitioned task. We used simulation-based experiments to study algorithms that the robots can employ to make the decision on the basis of cost estimates. We concentrated our study on the case in which the transfer at the interface is indirect. However, the same algorithms can be applied also to the case of direct transfer.

We proposed an ad-hoc algorithm that the robots can use to make the decision and we tested it in situations that vary for what concerns the cost of employing the fixed interface. The results of the experiments show that the use of the algorithm leads to good performance in all the tested conditions. The algorithm is also adaptive with respect to changes occurring in the environment that impact the costs and it scales with the size of the swarm.

We have also shown that the studied problem can be seen as a multi-armed bandit problem. We performed experiments to assess whether algorithms proposed in the reinforcement learning literature to tackle multi-armed bandit problems can be employed in the studied task partitioning problem. The results of the experiments confirm that these algorithms can indeed be applied. This is an important result, since many algorithms have been proposed to tackle the multi-armed bandit problem and they can potentially be applied to task partitioning.

Finally, we studied the effect of explicit communication within the swarm. Contrary to what one might expect, communication is not always beneficial. Communication causes a fast spread of information within the swarm. On the one hand, this renders the swarm faster in converging to a particular partitioning solution, which may be beneficial in terms of performance. On the other hand, the capability of the swarm to adapt to varying environmental conditions may be harmed.

Chapter 6

Deciding the Amount of Work Contributed by a Sub-task

In the previous chapter, we studied the case in which a fixed interface limits the amount of work a robot can contribute to transportation. With such a limitation, the choice available to the robots is binary: either utilize or bypass the interface. In this chapter, we consider the case in which the amount of work is not subject to such a constraint. As mentioned in Chapter 3, deciding the amount of work to contribute with a sub-task is equivalent to defining the size of that sub-task. The capability of making this decision autonomously is one of the building blocks to define complex partitioning strategies.

Compared to the problem presented in the previous chapter, the one considered here is more complex as the choice available to each robot is not binary. In the foraging scenario studied in this chapter, the robots are free to select the distance traveled with an object and therefore each robot has an infinite number of options to select from.

A further complication resides in the fact that the interfaces between sub-tasks are movable and thus there is a tighter dependency between the work of different robots. In fact, the position of an interface is defined by the robot transporting objects to that interface. A second robot collecting objects from the interface can do so only if the first robot indeed transports objects there. The work of the second robot therefore depends on the decisions made by the first. As, in general, transportation involves several steps (i.e., sub-tasks), the decision of a single robot can disrupt the work of many other robots downstream.

In the foraging scenario studied in this chapter, the robots must tackle a localization problem. Differently from the problem studied

in the previous chapter, the robots do not know a priori the location of the source nor the interfaces. For this reason, upon collecting an object from the environment, a robot maintains an estimate of its position relatively to the location where the object was found. This estimate is used by the robot to return to that location with the goal of collecting other objects. Several localization techniques have been proposed in the robotics literature; in Section 6.1, we discuss research work on localization that is relevant to what presented in this chapter. In Section 6.2, we present the foraging problem studied in this chapter. In Section 6.3, we illustrate the application of our approach for autonomous task partitioning to the foraging scenario. In Section 6.4, we describe the experimental tools that we utilize to implement the scenario. Even though we perform the experiments in simulation, we built the simulation models on the basis of data collected in real-robot experiments. In Section 6.5, we present these experiments and we compare their results with the results obtained in equivalent simulations. This comparison serves as a validation of the models employed in simulation. In Section 6.6 we report experiments in which we test the properties of the studied robotic system and we verify the applicability of our approach to achieve autonomous task partitioning in foraging. Finally, in Section 6.7, we summarize the contents and contributions of this chapter.

6.1 Localization

As mentioned above, in the scenario studied in this chapter each robot faces a *localization* problem, that is how to estimate its position with respect to the location where an object was found (details are given in Section 6.2). Localization consists in determining the position of a moving object in an environment and it is a common problem in robotics. Several techniques to tackle the localization problem have been proposed in the literature (see Feng et al., 1994, for a review). Most of the existing techniques are not suited to swarm robotics, either because they require complex sensing and computational capabilities (e.g., using maps) or because they depend on external infrastructures that are not always available (e.g., GPS).

In swarm robotics, different approaches have been proposed to perform localization. A common approach is to use robots of the swarm as landmarks. This is done by letting some of the robots move, while others stand still and serve as reference points for the moving robots.[1]

[1] Therefore while acting as a landmark, a robot does not contribute to the execution of tasks.

Examples of application of this approach can be found in the works of Kurazume and Hirose (2000), Grabowski et al. (2000), and Rekleitis et al. (2001). A similar approach is to form chains of robots that link points of interest in the environment. The chains can be followed by other robots to navigate in the environment. Examples of this approach are the works of Werger and Matarić (1996), Nouyan et al. (2009), and Dorigo et al. (2013). Finally, approaches inspired by the pheromone trails used by ants have also been proposed in the literature. In such approaches, the robots have the capability of marking the environment with information (representing pheromone), which is used for navigational purposes. Examples of this approach are, among others, the works of Johansson and Saffiotti (2009), and Mayet et al. (2010).

Differently from the research works mentioned, the work presented here combines the use of odometry with task partitioning. Odometry[2] is the computation of the position of a robot (or a vehicle in general) by the integration over time of measures coming from motion sensors. Odometry is appealing because it is simple, it requires low computational and sensing capabilities, it does not require any robot of the swarm to stand still and serve as a landmark, and it does not require the robots to be able to mark the environment with pheromone. The main problem with odometry is that the integration of noisy information over time leads to errors that can grow unbounded. Different solutions have been proposed to deal with this problem. In some cases, dedicated hardware systems are installed with the sole purpose of reducing the magnitude of odometry errors (see, for example, Hongo et al., 1987), or measuring and correcting it, as in the work of Borenstein (1994). Calibration procedures can also be utilized to measure and correct systematic components of the odometry error (Borenstein and Liqiang, 1996). However, calibration must be performed on a per-robot basis and therefore it might be prohibitive when the number of robots is large, as in swarm robotics applications. A widely used approach to improve odometry in robots (and vehicles in general) is the use of Kalman filters, initially proposed by Kalman (1960). Kalman filters are based upon models of the system in which they are applied, thus their effectiveness depends on the accuracy of these models.

The foraging problem studied in this chapter shares many similarities with the ones tackled in the works of Vaughan et al. (2002), Gutiérrez et al. (2010), and Ducatelle et al. (2011). In these works, localization is also based upon odometry, which is complemented with the use of explicit communication between the robots. In the system

[2]Odometry is also referred to as "dead-reckoning" in the literature.

proposed by Vaughan et al. (2002), the robots create trails of waypoints between locations of interest in the environment. The trails are virtual: the waypoints, computed through odometry, are broadcast via radio and internally stored by the receiving robots. Ducatelle et al. (2011) propose a very similar approach: the robots exchange positional information that is used to reach target locations. However, the robots themselves are the waypoints that must be followed in order to reach a target. Gutiérrez et al. (2010) describe a localization strategy based on "social odometry": the robots use the information communicated by peers to correct their estimated target locations. Compared to the three works mentioned, our solution requires the robots neither to communicate nor to share knowledge. This removes many of the requirements and renders the implementation of the system feasible also with minimalistic robotic platforms.

6.2 Problem Description

In the foraging scenario studied in this chapter, the objects to be collected are clustered in the source, that we assume never depletes. The robots have no a priori knowledge about the location of the source and they must explore the environment in order to find it. On the contrary, the direction of the nest can be perceived by the robots from every position in the environment.

The behavior of the robots. Figure 6.1 illustrates a finite state machine that describes how the robots perform foraging. Initially, each robot performs a random walk in order to explore the environment and find objects. Upon finding and collecting an object, the robot navigates towards the nest. While navigating, the robot keeps an estimate of its own position, relatively to the location where the object was collected. Each robot faces a localization problem: when moving, the robot must update its position estimate so that it can subsequently return to the location where it found the object. As mentioned in the previous section, the robots use odometry to tackle this problem: the positional information is derived from the integration of the measures of the motor encoders of the wheels.

Due to noise, the estimate a robot has of its own position relatively to the location where the object was collected can deviate from the real one. The quality of this estimate depends on the distance traveled by the robot: the longer the distance traveled, the less accurate the estimate becomes. This is mainly due to three factors. First, errors on the bearing have a stronger impact at longer distances. Second, the longer

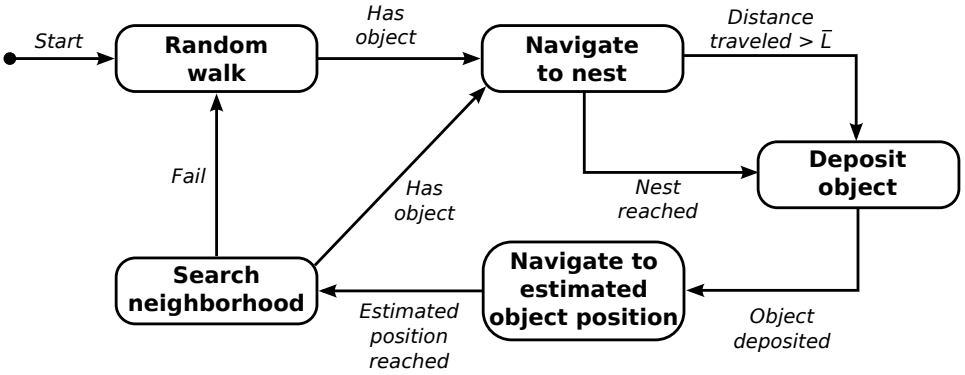

Figure 6.1: Finite state machine describing the high level behavior of each robot.

a robot travels, the more it accumulates errors due to sensor and actuator noise. Third, traveling for a longer time increases the probability that the estimate is distorted by non-systematic components such as collisions, uneven terrain, and wheel slipping.

Upon reaching the nest, the robot releases the object and uses the position estimate to return to the location where the object was collected. When the robot reaches the position it estimated the objects to be, it performs a *neighborhood search*, which consists in searching for objects within a limited area centered around the estimate. Neighborhood search can compensate relatively small estimation errors. In case the estimation error is big, the neighborhood search is likely to fail. If this happens, the robot resumes performing random walk to locate again objects. We say that the robot *got lost* when the neighborhood search fails and the robot must return to random walk. Random walk is a time consuming operation due to its stochastic nature, therefore its cost is high and it is desirable to minimize the frequency with which robots get lost.

The use of task partitioning. Task partitioning can be used to reduce the negative effect of odometry errors and to improve the localization capabilities of the robots. Instead of performing the whole transportation task, each robot only contributes with a lesser amount of work, traveling a limited distance $\bar{L}$ from the location where an object was collected (i.e., performing a sub-task of length $\bar{L}$). Upon traveling such a distance, referred to as *partition length*, the robot deposits the object on the ground and, using its odometry estimate, it returns to the location where that object was collected. Since the odometry error grows with the distance traveled, the expected estimation error

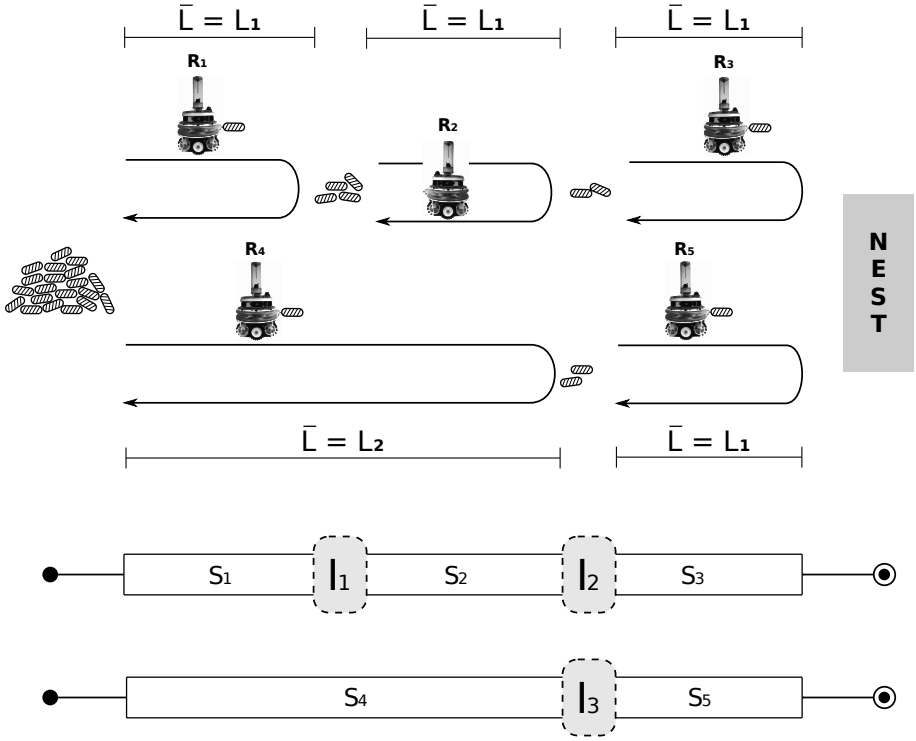

Figure 6.2: Representation of the foraging scenario and contribution of the robots to object transportation. Each robot contributes a certain amount of work to transportation, which corresponds to the partition length $\bar{L}$ traveled by that robot (in general the value varies from robot to robot). The strategy employed to perform object transportation depends on the amount of work contributed by the each robot. In this case, the interfaces between sub-tasks are movable, since the robots are free to select the amount of work they contribute.

is smaller compared to the case in which the robot performs transportation as an unpartitioned task. The result is that the probability that a robot gets lost diminishes, therefore reducing the total expected random walk time of the swarm.

As objects can be deposited everywhere in the environment, the robots can find them not only at the source, but also in other locations along the way from source to nest. The result is that the transportation of an object is carried out as a sequence of sub-tasks performed by different robots, as represented in Figure 6.2.

Each robot faces the problem of deciding the amount of work it contributes to transportation. In the context described, this problem corresponds to selecting the distance $\bar{L}$ traveled towards the nest with an object (i.e., the length of the sub-task to perform). As mentioned in

Chapter 3, each robot makes decisions independently from the others and therefore the overall strategy employed to perform transportation results from a combination of individual decisions. For example in Figure 6.2, the robots R_1, R_2, and R_3 all select the same partition length value. The result is that transportation is partitioned into three sub-tasks of equal length. The robots R_4 and R_5 employ a partitioning strategy such that the task is partitioned into two sub-tasks, each contributing a different amount of work to transportation. Since the robots deposit objects on the ground, the interfaces between the sub-tasks are indirect.

Notice that a robot cannot discriminate between the source and a location where objects are deposited by another robot. Therefore, the odometry estimate of a robot might be relative to a location of the environment which is not the source. For example, robot R_2 in Figure 6.2 would estimate its position relatively to the location where R_1 is depositing objects and it would try to return there upon depositing an object on the ground.

The use of movable interfaces renders the dependency between the work of different robots tighter. In the example of Figure 6.2, the decisions of R_1 impact the work of R_2 which, in turn, has an impact on R_3. For example if R_1 decided to transport objects all the way to the nest, R_2 would not find objects deposited on the ground.

The robots must decide their contribution in terms of amount of work based on a trade-off between costs of different nature. On the one hand, contributing with a large amount of work is desirable: the robots travel further while carrying objects and the number of times objects are deposited on the ground is reduced. Depositing an object on the ground entails overhead costs because the same object must be found and picked up by different robots. On the other hand, traveling further increases the magnitude of the odometry error and the likelihood that the robot gets lost; consequently the expected search costs are higher.

6.3 Application of the Proposed Approach

In this section, we describe the application of our approach for autonomous task partitioning to the foraging problem presented in Section 6.2. The robots decide the length of the sub-task they perform on the basis of a mapping between the amount of work and the resulting costs to perform the transportation task. This mapping is defined by a *cost function*. In general, the cost function cannot be defined a priori since it depends on properties of the environment and of the tasks that may be unknown. As a consequence, the cost function must be deter-

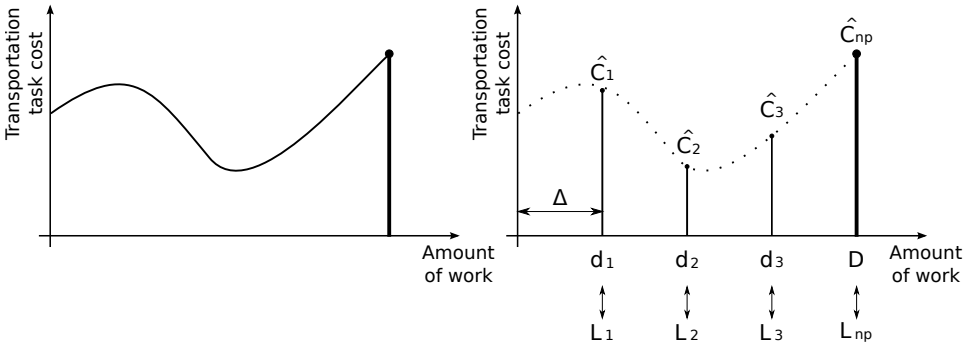

Figure 6.3: Cost function (left) and its model as identified by a robot (right). The cost function maps the amount of work contributed by a robot performing a sub-task to the resulting cost to perform the overall transportation task. The robot builds a discrete model of the cost function. This model consists of a finite set of cost estimates $\hat{C}_i$, each associated to an element L_i which identifies a certain distance value d_i that the robot can travel.

mined with an online process performed by the robots. Additionally, the cost function may change in time, since the costs associated to the execution of tasks are likely to vary as a result of the actions performed by the robots. Each robot is merely able to build a model of the real cost function, based on sensory input. In the following, we describe how this model is built by each robot and we present algorithms that can be utilized to decide the amount of work contributed by the robot.

The Model of the Cost Function

As mentioned, in the foraging scenario presented in this chapter, the amount of work corresponds to the distance traveled by a robot towards the source. Since the goal is transporting as many objects as possible, we express the costs in terms of time. Therefore, the *cost function* maps the distance traveled by a robot to the time required to perform the object transportation task (see Figure 6.3 left).

The robots build a discrete model of the cost function, as represented in Figure 6.3 (right). This model consists of a finite set of cost estimates $\hat{C}_i$, each associated to an element L_i which identifies a certain amount of work that the robot can contribute to transportation (i.e., a certain distance d_i traveled by the robot). Selecting the length of a sub-task therefore corresponds to selecting an element L_i on the basis of the estimates $\hat{C}_i$. Note that the domain of the cost function is bounded: a robot can contribute with a maximum amount of work that corresponds to performing the whole task (see thick vertical bar

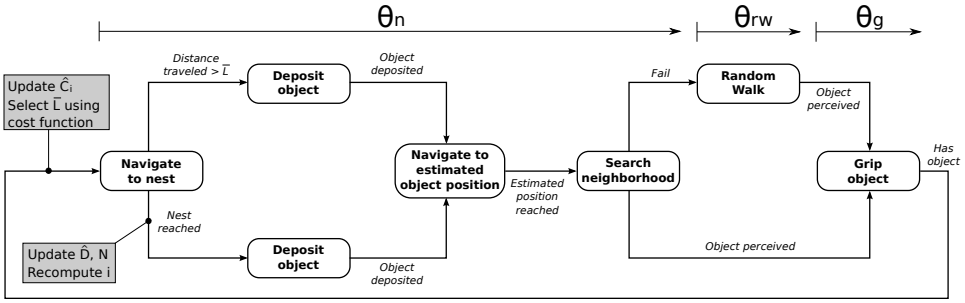

Figure 6.4: High level representation of the behavior of the robots. The finite state machine also indicates when information is updated by the robot (gray rectangles).

in Figures 6.3 left and right).

Figure 6.4 illustrates the high level behavior of each robot and can be used as a reference to better understand the concepts explained throughout the rest of this section. In the figure, the white rectangles indicate actions performed by the robot. The gray rectangles indicate moments in which the robot updates internal information.

Each robot individually builds the set $\mathbb{L}$, containing the elements L_i. Upon gripping an object, a robot selects an element in $\mathbb{L}$ and uses the associated value d_i as the partition length $\bar{L}$ to be traveled with that particular object. In this section, we illustrate how robots build the set $\mathbb{L}$; possible algorithms to select the partition length value are presented in Section 6.3.1.

Number of elements in $\mathbb{L}$. The set $\mathbb{L}$ is composed of N elements, each associated to a possible value of partition length. Each robot builds the set $\mathbb{L}$ by discretizing the transportation task length D, which corresponds to the distance between source and nest, with a discretization step Δ. The value of N is computed by a robot as:

$$N = \left\lceil \frac{\hat{D}}{\Delta} \right\rceil \ . \tag{6.1}$$

We fixed the discretization step Δ to 0.5 m which corresponds (roughly) to the visual perception range of the robots (refer to Section 4.2). Discretization is performed by a robot on the basis of $\hat{D}$, which is its estimate of the real distance D between source and nest (i.e., an estimate of the task length). Notice that the robot can find objects deposited by other robots along the way between source and nest (i.e., closer to the nest than the source actually is) and therefore $\hat{D}$ can be an un-

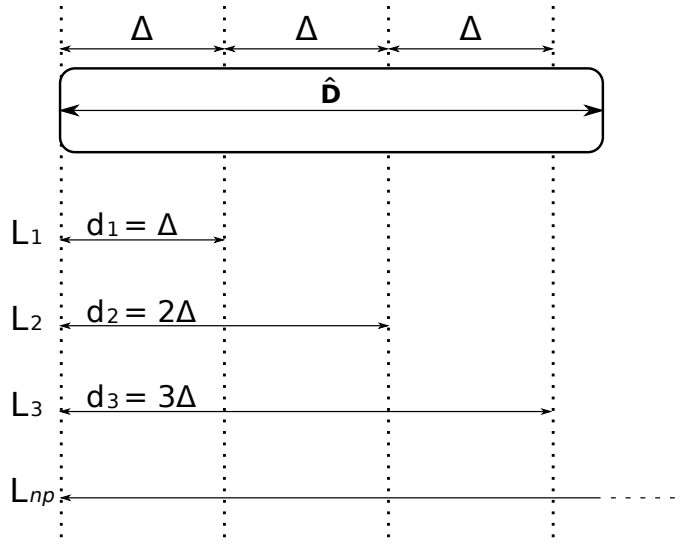

Figure 6.5: Example representing the set $\mathbb{L}$ from which a robot selects the value of the partition length. The robot builds the set by discretizing the estimated task length $\hat{D}$ with a step Δ, obtaining $N-1$ values L_i. The figure reports the case in which $N=4$. The special value L_{np} is also an element of $\mathbb{L}$ and corresponds to performing the task without employing task partitioning.

derestimate of D. For this reason, the robot must keep $\hat{D}$ and the set $\mathbb{L}$ up to date. Details about the way the robots update $\hat{D}$ and $\mathbb{L}$ are given in the following.

Elements of $\mathbb{L}$. The elements L_i that compose the set $\mathbb{L}$ are associated each with a certain amount of work (distance d_i), that the robot contributes to the transportation task. Among the others, $\mathbb{L}$ contains the special element L_{np}, which corresponds to performing the transportation task as an unpartitioned task (represented with a thick vertical bar in Figure 6.3, left and right). A robot selecting L_{np} transports the carried object all the way to the nest. Note that the distance d_{np} associated to L_{np} is D. As the robots cannot determine an exact value of D, in the actual implementation the condition "Distance traveled $> \bar{L}$" (Figure 6.4 top left) always evaluate to false (i.e., the robot travels all the way to the nest).

The remaining $N-1$ elements that compose the set $\mathbb{L}$ are each associated to a distance value d_i computed as follows:

$$L_i \leftrightarrow d_i = \Delta \cdot i \qquad \text{with } i \in \{1, 2, \ldots, N-1\} \; . \tag{6.2}$$

Figure 6.5 represents the set $\mathbb{L}$ and illustrates the relationship between

the number of elements N (in figure $N = 4$), the estimated length of the task $\hat{D}$, the discretization step Δ, and the elements L_i.

Each time a robot grips an object, it selects an element L_i (see Figure 6.4, left-hand side) and therefore the associated distance d_i to travel with that object. Different task instances can therefore be partitioned in different ways, depending on the L_i selected by the robots involved in the transportation (see, for example, Figure 6.2). Recall that, as mentioned in Chapter 3, the decisions of the robots are independent and each robot is not aware of the choice of L_i made by the other robots. The global process by which the transportation of an object is partitioned into sub-tasks results from independent choices in a self-organized manner.

Estimation of D. Each time a robot reaches the nest (label *nest reached* in Figure 6.4), it checks if $\hat{D}$ underestimates the current distance to the source. In this case, the robot sets $\hat{D}$ to the current (estimated) distance to the source and, if needed, it updates the set $\mathbb{L}$ using Equations (6.1) and (6.2). Note that odometry errors can cause a robot to overestimate D. However, the robots can only determine whether $\hat{D}$ underestimates D (by measuring a value greater than $\hat{D}$), but not whether it overestimates D. Initially, robots have no information about the source-to-nest distance. The value of $\hat{D}$ is initialized to zero and the set $\mathbb{L}$ only contains L_{np}. Consequently, the robots travel all the way to the nest in their first trip, thus calculating a first estimate of the value of D.

Estimation of the Costs

The model of the cost function built by a robot (Figure 6.3, right) consists of a set of values $\hat{C}_i$, each associated with an element of $\mathbb{L}$. Each value $\hat{C}_i$ is the robot's estimate of the time required to perform the overall transportation task, when the robot transports an object for a distance d_i. Notice that the overall transportation time depends on the decisions of all the robots involved in transportation. A robot estimating such a time does not have any information about the decisions of the others and therefore it builds its estimate on the basis of the cost experienced when performing its own sub-task (details are given later on in this section).

A robot updates the estimate $\hat{C}_i$ at the moment at which an object is gripped. At that moment, the robot has the information it needs to update the estimate: the measured time it took to travel with the object, deposit it in the environment, return to the location where the

object was initially found, and grip another one. Gripping an object marks the completion of a sub-task for a robot (i.e., the transportation of the previous object) and the beginning of a new one. Therefore, at that moment, the robot must update its model of the cost function and use the updated model to select a new element L_i.

The cost estimates $\hat{C}_i$ are updated as a recency-weighted average of the observed costs:

$$\hat{C}_i = (1 - \alpha)\, \hat{C}'_i + \alpha\, \bar{C}\ , \tag{6.3}$$

where $\alpha \in (0, 1]$ is a memory factor, $\hat{C}'_i$ is the value of the cost estimate before the update, and $\bar{C}$ is the observed cost associated to the last trip towards the nest (i.e., the last sub-task performed). The use of a recency-weighted average gives more weight to recent observations, resulting in a more reactive behavior in non-stationary environments compared to a simple average (Sutton and Barto, 1998). The observed cost $\bar{C}$ is computed as:

$$\bar{C} = \frac{\hat{D}}{d_i}\,(\Theta_n + \Theta_g) + \Theta_{rw}\ , \tag{6.4}$$

where $\Theta_n + \Theta_g$ is the measured cost of the last sub-task performed (see Figure 6.4, top). This cost is composed of two parts. Θ_n accounts for the navigational costs (i.e., transporting and depositing an object, returning to the source, and performing neighborhood search). Θ_g is the cost of gripping another object, an action that must be performed before the following sub-task can be started. Θ_{rw} measures the time spent performing random walk in case the robot got lost.

Note that $\Theta_n + \Theta_g$ measures the cost of the actions needed to perform sub-tasks, while Θ_{rw} is a penalty due to a robot getting lost. As mentioned, the overall transportation time depends on the decisions of all the robots involved. Since robots are not aware of the decisions of each other, they estimate the overall transportation time on the basis of their own measures. For this reason, $\Theta_n + \Theta_g$ is scaled using the ratio $\hat{D}/d_i$. This ratio expresses the contribution of the sub-task to the overall transportation task. Again, since the robots cannot determine an exact value for d_{np}, the ratio $\hat{D}/d_{np}$ evaluates to 1 in the actual implementation.

The cost estimate $\hat{C}_i$ updated by a robot depends on whether or not that robot reached the nest during the last sub-task execution. Consider, for example, the two cases reported in Figure 6.6. In the situation represented in the figure, R_1 and R_2 just gripped an object and selected an element L_i. The element L_1 selected by R_1 is such that the corresponding distance value d_1 does not exceed the current distance

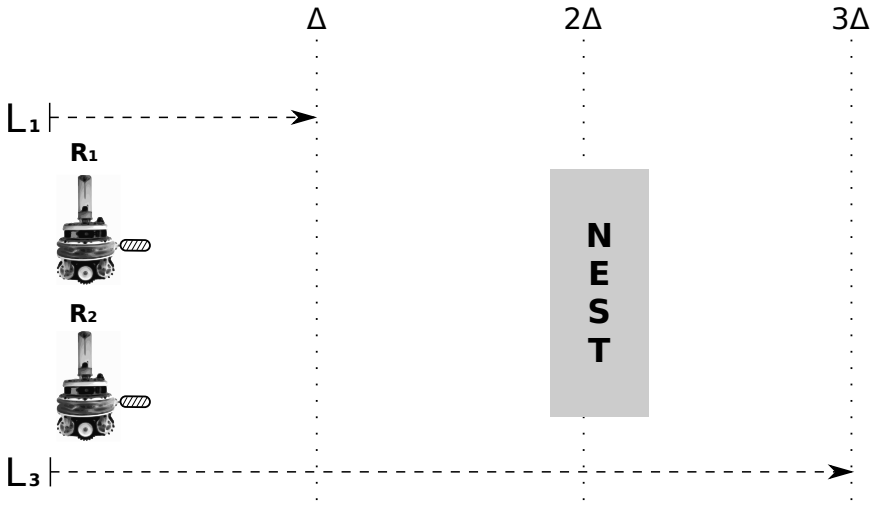

Figure 6.6: Possible situations happening upon gripping an object. The robot R_1 selects the element L_1 such that the corresponding distance value does not exceed the distance between the robot and the nest. On the contrary, the element L_3 selected by R_2 is such that the robot will reach the nest in its subsequent trip. Upon reaching the nest, R_2 will recompute the index i corresponding to the distance actually traveled (in this example i becomes 2).

between R_1 and the nest. The same does not hold for the element L_3 selected by R_2. R_2 will encounter the nest before traveling a distance d_3 and therefore the actual contribution of R_2 to transportation is smaller.

To take into account this event, upon reaching the nest (transition labeled *nest reached* in Figure 6.4) R_2 recomputes the index i on the basis of the distance that was actually traveled ($i = 2$ for the case reported in figure). Upon gripping the following object, R_2 will therefore update the estimate $\hat{C}_2$ instead of the estimate $\hat{C}_3$ that was the one associated to the value L_3 initially selected by the robot. R_1, on the other hand, will not reach the nest and it will actually travel a distance d_1. Therefore the cost estimate that will subsequently by updated by the robot is $\hat{C}_1$, associated to the selected element L_1.

The cost estimates are initialized randomly: when a robot reaches the nest and, according to Equation (6.2), adds new values L_i to the set $\mathbb{L}$, it initializes the associated costs $\hat{C}_i$ with a random value.

6.3.1 Task Partitioning Algorithms

In the experiments presented in Section 6.6, we compare different task partitioning algorithms that can be utilized to select the value of the partition length from the set $\mathbb{L}$. We compare an algorithm based on our approach with a set of reference algorithms. All the reference algorithms select an element L_i from the set $\mathbb{L}$. However, in none of them the selection is based upon the cost estimates $\hat{C}_i$ that model the cost function. In the rest of this section we describe in detail each algorithm.

The Cost-based Partitioning Algorithm

To apply our approach to autonomous task partitioning, we need a mechanism that selects an element L_i among the possible elements in $\mathbb{L}$, on the basis of the cost estimates $\hat{C}_i$. Here, we use the *ε-Greedy* (Sutton and Barto, 1998) algorithm to perform the selection. ε-Greedy selects with a probability $1-\varepsilon$ the element L_i with the minimal associated cost and with a probability ε a random value. We call *cost-based partitioning algorithm* the task partitioning algorithm that utilizes *ε-Greedy* and the cost estimates to select an element L_i and the associated partition length value.

Notice that our approach does not require any specific algorithm to select an element in $\mathbb{L}$. Other algorithms, such as reinforcement learning techniques (see Sutton and Barto (1998)), could be used in place of ε-Greedy. We decided to use ε-Greedy because of its simplicity and because its only parameter ε directly expresses the degree of exploration of the algorithm.

The Fixed Algorithms

A first family of reference algorithms are the *fixed* algorithms: the element L_i is fixed a priori. This element is the same for all the robots and remains constant over time. Therefore, the partition length value is also the same for all the robots and it never changes. We refer to a fixed algorithm with the label *fixed X*, where X identifies the element L_i that is used by the algorithm. A special case of fixed algorithm is the *never-partition algorithm*: in this case, the robots do not employ task partitioning and transport objects from the source to the nest.

The Random Initialization Algorithm

The *random initialization algorithm* consists in stochastically selecting the element L_i from $\mathbb{L}$ at the beginning of the experiment. Each robot

selects its own element L_i and never changes it during the course of the experiment.

The algorithm requires to initialize the set $\mathbb{L}$. To this aim, the first time a robot grips an object, it transports it directly to the nest, so that $\hat{D}$ can be estimated. 10% of the value is then added to the estimate, to partially compensate for underestimation errors. The resulting value is used to initialize the set $\mathbb{L}$ as described by Equations (6.1) and (6.2). The robot then stochastically selects an element L_i from the set $\mathbb{L}$ and the selected element is used by that robot throughout the rest of the experiment. We also tested a variation of this algorithm, whereby the robots stochastically select an element L_i from $\mathbb{L}$ each time an object is gripped. The algorithm performed badly across all the experiments and we decided not to include its results in this dissertation.

6.4 Experimental Setup

In this section, we present the implementation of the system described in the Section 6.2. We perform the experiments using ARGoS to simulate the marXbot robotic platform. In Section 6.4.1, we illustrate the environment used for the foraging experiments presented in this chapter. In Section 6.4.2, we mention characteristics of the real marXbots and of their behavior that are relevant to what presented in this chapter. In Section 6.4.3, we describe simulation models that were proposed in Pini et al. (2012b) and that are used in all the simulation-based experiments presented in this chapter.

6.4.1 Experimental Environment

The robots perform foraging in the environment represented in Figure 6.7. The width W and the length L of the environment depend on the specific experiment. The nest is located close to one of the borders and it is marked by a black patch on the floor, which is 1.4 m wide and 0.45 m long. The robots can determine whether they are inside the nest by the color of this patch. Three lights (crossed circles in the figure) mark the nest and are used by the robots to determine its direction.

The source is located in front of the nest, at a distance D, measured from the center of the nest to the center of the source. The source is composed of 5 objects (see Section 4.2) positioned as shown in Figure 6.7 at a distance of 0.17 m from the object in the center. Each time a robot removes an object from the source, a new one is added in the same location. Thus, the source never depletes. When a robot releases an object within the boundaries of the nest, transportation is

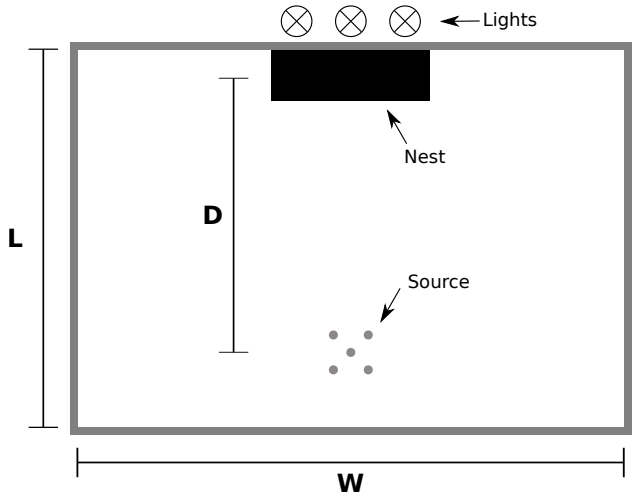

Figure 6.7: Representation of the environment in which the robots perform foraging. The nest (black rectangle on top) is marked by three light sources (crossed circles) located nearby, that can be perceived by the robots. The source, located in front of the nest, is composed of 5 objects (gray circles), positioned as shown. The size of the environment (W, L) and the distance of the source from the nest (D) depend on the specific experiment.

completed and the object is removed from the environment.

The parameters of the environment (D, W, and L) influence the effectiveness of the partitioning strategy used to perform foraging. As mentioned in Section 6.2, the longer a robot travels, the lower the accuracy of its odometry estimate becomes (i.e., the higher the probability that the robot gets lost). The relative improvement of using task partitioning over not using it depends on the value of D: when D is small, the accuracy in locating the source may be good also if the robots travel all the way from source to nest. Conversely, for high values of D, task partitioning can lead to significant improvements. In general, the higher the value of D (i.e., the longer the task), the more advantageous task partitioning becomes.

The parameters W and L define the difficulty of finding the source when searching the environment: the larger the surface of the environment, the longer it takes (on average) to find the source. In other words, the two parameters determine the expected cost of getting lost and therefore they also have an influence on which strategy is preferable to tackle transportation. For example, in a large environment, performing transportation as an unpartitioned task may be disadvantageous even if D is small. In fact, even if the robots get lost rarely due to the short distance traveled, when they get lost they might need

to perform random walk for a long time before they find an object.

Conversely, for high values of D, task partitioning may not be advantageous if the environment is small. In fact, a small environment requires only a short period of time to be explored. Therefore, the cost of getting lost may be lower than the overhead costs of task partitioning, rendering preferable to perform transportation as an unpartitioned task.

6.4.2 Behavior and Characteristics of the MarXbot

In this section, we provide implementation details about the behavior of the robots and describe the odometry error that characterizes the real marXbots. In the simulation-based experiments, we utilize the same behaviors and implement a model of the odometry error, based on what observed in reality.

Random walk. The robots search for objects in the environment using a random walk implemented as follows. By default, the robots move straight, at a maximum velocity of 0.1 m/s, avoiding obstacles. The stochastic component consists in randomly generating a new direction of motion, uniformly sampled in $[-115°, 115°]$. The random direction is generated with a 5% probability per control-step[3] (i.e., the robot keeps moving straight with a 95% probability). The neighborhood search is also performed using random walk, but the robot remains in a circular area of radius 0.5 m, centered around the position at which the robot expected to find objects. If a robot is about to leave the search area, it generates a new random direction biased towards the center of the area. As mentioned, a robot may abandon the search of a neighborhood when unsuccessful. A timeout mechanism governs abandoning. The robot abandons in case it has been performing the neighborhood search for 60 s without detecting any object. In this case, in fact, it is likely that the robot reached a position far away from the location of the objects.

Object gripping. Object gripping is based on vision; the marXbot uses information from the omnidirectional camera to perceive objects in the surroundings and approach them. Once the robot is close to an object to be gripped, it uses the front proximity sensors to refine its alignment. A repulsion mechanism is also part of object gripping: the robots ignore any red blob (i.e., an object) which is perceived in proximity of a blue blob. This prevents that more robots grip the same

[3] A control-step lasts 0.1 s.

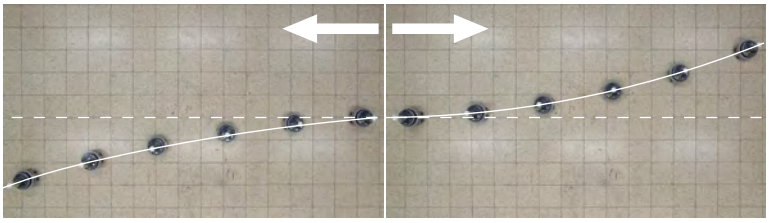

Figure 6.8: Trajectory followed by a real marXbot when the speed of both wheels is set to $-0.1\,\mathrm{m/s}$ (left-hand side) and $0.1\,\mathrm{m/s}$ (right-hand side). The white arrows on top indicate the direction of motion of the robot. The continuous curved line indicates the trajectory followed by the robot and the dashed line a straight reference trajectory.

object at the same time. While approaching an object with the intent of gripping it, a robot lights up its LEDs in blue, to repel other robots from the same object. For the same reason, the LEDs are lit up in blue also during the transportation of an object towards the nest.

Check for unreachable destination. Due to odometry errors, the robot may try to reach a position that lies outside the perimeter of the environment while navigating to the source. To determine whether it is trying to reach a position that is not within the boundaries of the environment, each robot periodically checks its estimated distance to the source. If the distance does not decrease in time, the robot assumes that it is trying to reach a position outside the perimeter and it returns performing random walk (i.e., it gets lost). In fact, if the estimated distance to the target position is not decreasing, it means that an obstacle blocks the movements of the robot. If this happens for a long period of time (30 seconds in the experiments), it is likely that the obstacle is a wall marking the perimeter of the environment, rather than another robot. Notice that the mechanism is prone to errors: if the density of robots is high, the movements are harder and the mechanism described here can be triggered by the presence of other robots.

Odometry error. While carrying out experiments with the real marXbots, we noticed that they suffer from a systematic drift towards the left-hand side with respect to the direction of motion. Figure 6.8 shows the shape of the trajectory followed by a marXbot when the speed of both wheels is set to $-0.1\,\mathrm{m/s}$ (left-hand side) and $0.1\,\mathrm{m/s}$ (right-hand side). The figure is built using snapshots taken from a video that is available with the supplementary material (see Annex B). The figure reports the direction of motion of the robot (white arrows on

top), the trajectory followed by the robot (white continuous line) and a reference straight trajectory (white dashed line). The drift towards the left-hand side is not constant: the amount by which the same marXbot drifts varies from trip to trip. The marXbot cannot measure such a drift: using odometry, the robot would estimate its trajectory to be (roughly) straight.

Solutions such as the calibration procedures described by Borenstein and Liqiang (1996) or Kalman filtering (Kalman, 1960) could improve odometry. However, as mentioned in Section 6.1, their applicability and effectiveness is not granted. Calibration and filtering could potentially reduce, but never eliminate errors and therefore the problems due to imprecise localization would eventually arise and the idea that task partitioning can improve localization would still be valid. Recall that our main goal here is not to maximize the foraging efficiency of the robots, but to test the proposed approach for autonomous task partitioning in a realistic setup. Therefore, for simplicity, we do not try to correct or reduce the odometry errors in any way. In real-world applications, techniques to enhance odometry can be coupled with task partitioning to further improve the system.

6.4.3 Simulation of the System Using ARGoS

In the foraging experiments presented in this chapter, we employ the 2D-dynamics physics engine offered by ARGoS. The simulation proceeds at discrete time-steps of 0.1 simulated seconds. All the simulated sensors and actuators are subject to noise. Gaussian noise with 0.02 m standard deviation is added to the distance readings of the omnidirectional camera. At each simulation step, a uniform random value between -5% and 5% of the reading is added to the measures of the ground-color, ambient-light, and proximity sensors.

Object gripping. In simulation we model object gripping using 80 time samples collected in the real-robot experiments presented in Section 6.5, with six marXbots performing foraging in a $W = 6.7\,\text{m}$ by $L = 4.5\,\text{m}$ environment. Each sample records the time spent by a robot to grip an object (i.e., the time from the moment the object was perceived to the moment it was gripped). Figure 6.9 reports the empirical distribution of the grip time samples. The samples are utilized in simulation to model the time spent by a robot to grip an object. Each time a robot grips an object, a random value is selected from the set of samples. The robot waits in place for a time corresponding to the selected value, before it can undertake the following action. The

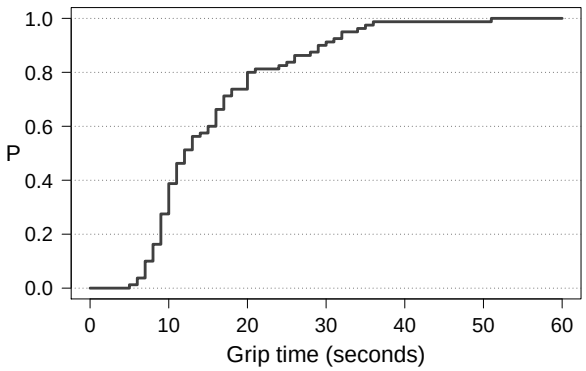

Figure 6.9: Empirical distribution function (P) of the 80 samples utilized to simulate the object gripping time. The samples have been collected with a group of six marXbots performing foraging.

amount of time the robot waits is discounted by the time already spent to reach and grip the object. This solution allows us to model in a simple but realistic way the variability that can be observed with the real marXbots in the time needed for gripping objects.

Model of the odometry error. The odometry error plays an important role in our experimental setup since it defines how successful the robots are in finding objects when returning to the source. The model that we use in simulation is built on the basis of odometry error samples collected with real marXbots performing the foraging experiments described in Section 6.5.

The setup of the experiments is as described in the following. A group of six robots performs foraging in the 6.7 m by 4.5 m rectangular environment represented in Figure 6.7. The source is located at a distance of $D = 4$ m from the nest. Upon gripping an object from the source, a robot travels all the way to the nest (i.e., task partitioning is not utilized). When the robot reaches the nest, it releases the object and it returns to the source using odometry.

The data upon which the odometry error model is built was sampled from video recordings of such experiments. A total of 61 error samples were collected from the recordings of a subset of the experimental runs, using the tiles on the floor as a reference to calculate the odometry error. These measures include effects such as collisions, wheel slippage, and avoidance as they are in the experiment at hand.

Figure 6.10 reports the 61 error samples collected from the videos. The origin of the axes in the plot represents the position in which an object was gripped by a robot. A point in the plot reports the X, Y

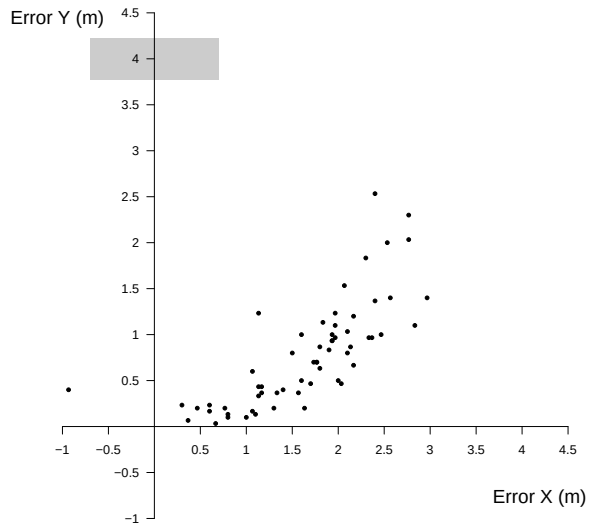

Figure 6.10: Odometry error samples collected in experiments performed with the marXbots (61 samples). Each sample reports the X, Y estimation error at the end of a trip from source to nest and back ($D = 4\,\text{m}$). The light gray rectangle marks the position of the nest.

error between the position at which the object was gripped, and the final position reached by the robot when returning to the source using the odometry estimate.

Algorithm 1 Pseudo-code for the odometry error model

1: $actuatedWheelSpeedR, actuatedWheelSpeedL \leftarrow ControllerStep()$
2: **if** $actuatedWheelSpeedR > 0$ **then**
3: $\quad wheelSpeedR = actuatedWheelSpeedR + \mu_{error} * actuatedWheelSpeedR$
4: **end if**
5: **if** $actuatedWheelSpeedL < 0$ **then**
6: $\quad wheelSpeedL = actuatedWheelSpeedL - \mu_{error} * actuatedWheelSpeedL$
7: **end if**
8: **if** *Object gripped* **then**
9: $\quad \mu_{error} = RAYLEIGH(\sigma)$
10: **end if**

Algorithm 1 reports the pseudo-code that describes how the odometry error is implemented in simulation. The drift towards the left-hand side is obtained by modifying the actuated values of the two wheel speeds (lines 2 to 7): the speed of the right wheel is incremented when the actuated value is positive, the speed of the left wheel is decremented when the actuated value is negative.[4] The robot is not aware of the

[4]Note that decreasing a negative wheel speed increases the absolute value of the speed.

drift: odometry is performed on the basis of *actuatedWheelSpeedR* and *actuatedWheelSpeedL*. The parameter μ_{error} represents the percentage by which the wheel speed is increased or decreased. The value of μ_{error} changes each time a robot grips an object. The new value is sampled from a Rayleigh distribution with parameter σ (lines 8 to 10). The same distribution is used to randomly initialize μ_{error}. We selected this distribution on the basis of the error points observed in Figure 6.10. The majority of the points are clustered in a certain area (values of X between 0.5 m and 2.0 m and Y between 0.0 m and 1.0 m); the remaining values are scattered around this area in a certain direction (i.e., left hand side with respect to the direction of motion of the robot). Due to its asymmetry, the Rayleigh distribution allows us to represent the pattern observable in Figure 6.10.

The value of the parameter σ characterizes the error of the robots: the higher its value, the higher the expected value of μ_{error}, and the larger the odometry error. The value of the parameter σ was fitted through a set of ad-hoc experiments. The default value of σ, indicated as $\bar{\sigma}$, is set to 0.0134. $\bar{\sigma}$ allows us to replicate in simulation a pattern similar to the one reported in Figure 6.10. To evaluate the impact of the odometry error on our system, in the experiments presented in Section 6.6 we test different values of the parameter σ. We express the value of σ used in an experiment as a fraction of $\bar{\sigma}$. Figure 6.11 reports error points (70 in each plot) collected in simulation for the different conditions tested in the experiments (excluding $\sigma = 0$, which generates no error).

Notice that the presented model does not represent explicitly real world phenomena (e.g. slippage, uneven terrain) or properties of the real marXbots (e.g. wheel misalignment, encoder resolution) that impact odometry. For our simulation this is acceptable, because our goal is to reproduce the observable behavior of the marXbot with a sufficient degree of accuracy.

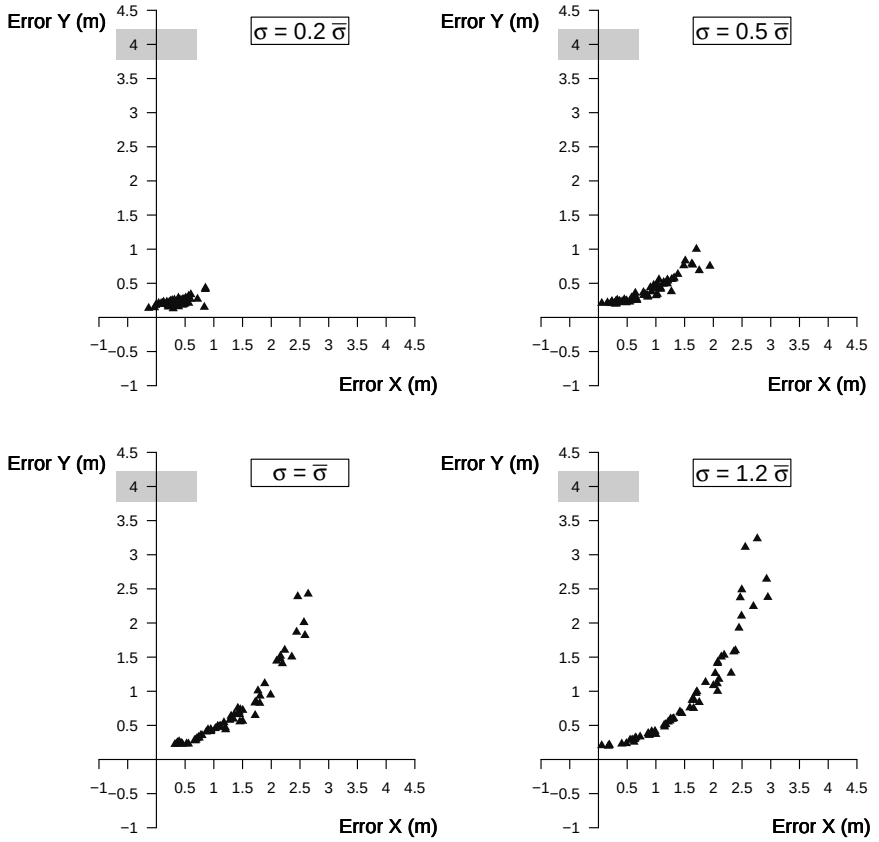

Figure 6.11: Odometry error samples collected in simulation for different values of σ.

6.5 Validation of the System

In this section, we present the experiments that we performed with the real robots and we compare their results to the ones obtained in equivalent simulations. At the moment of writing, there exist a number of multi-robot simulation softwares (see, for example, (Koenig and Howard, 2004; Michel, 2004; Carpin et al., 2007)). Despite this fact, a well-defined and universally accepted methodology for the validation of simulation models has not yet been developed.

Our goal here, is to perform a qualitative analysis and evaluate how closely the simulator reproduces trends and results observed in reality. We replicate in simulation the settings of the real-robot experiments. We compare the results obtained in simulation with the ones obtained in reality and we verify qualitatively that the simulation closely replicates the trends observable in real-world settings.

The experiments described in this section were originally proposed

in Pini et al. (2012b). In Pini et al. (2012b), the goal is to study the costs of task partitioning with direct sub-task interfacing. Therefore, the behavior of the robots is different for what concerns the way objects are transferred between sub-tasks. Once a robot has traveled a distance L from the point where it collected an object, it does not deposit the object on the ground, but it waits in place for another robot to which it can directly hand over the object. Apart from this difference in the behavior of the robots, the remaining experimental settings and behavioral characteristics of the robots are as described in this chapter.

Experimental Setup. The experiments are carried out in a 6.7 m by 4.5 m environment; the source is positioned at a distance $D = 4.0$ m from the nest. The swarm is composed of 6 robots; each experimental run lasts 30 minutes. Figure 6.12 reports two snapshots taken from a video recording of an experiment. The snapshot on top shows the initial configuration used in the experiments presented here: 3 marXbots are located in close proximity of the source and 3 marXbots in the center of the environment, at a distance $D/2 = 2.0$ m from the nest. Figure 6.12(bottom) reports the situation after 90 seconds.

We test two cases, for each we perform 6 real-robot experiments and 200 simulations. In one case, the robots perform foraging using task partitioning. In the rest of this section, we refer to this case by saying that the robots perform transportation using the *partition strategy*. In this case, the 3 robots starting at the center of the environment initially perform neighborhood search, centered around their starting position. The value of the partition length is fixed a priori to 2.0 m. A robot that receives an object from another robot delivers it to the nest (i.e., an object can be transferred only once). As a result, the transportation task is partitioned into two sub-tasks of (approximately) equal length. We compare the case in which the robots use the partition strategy to the case in which the robots perform transportation as an unpartitioned task. We refer to this second case saying that the robots perform transportation using the *non-partition strategy*. Differently from the previous case, the 3 robots in the center start by performing random walk. In case an object is deposited by mistake outside the nest by a robot, it is removed from the experimental area.[5] The complete data collected in the twelve experimental runs with real-robots and video recordings of two of such runs are available with the supplementary material (see Annex B).

[5]This situation can only happen in the real-robot experiments.

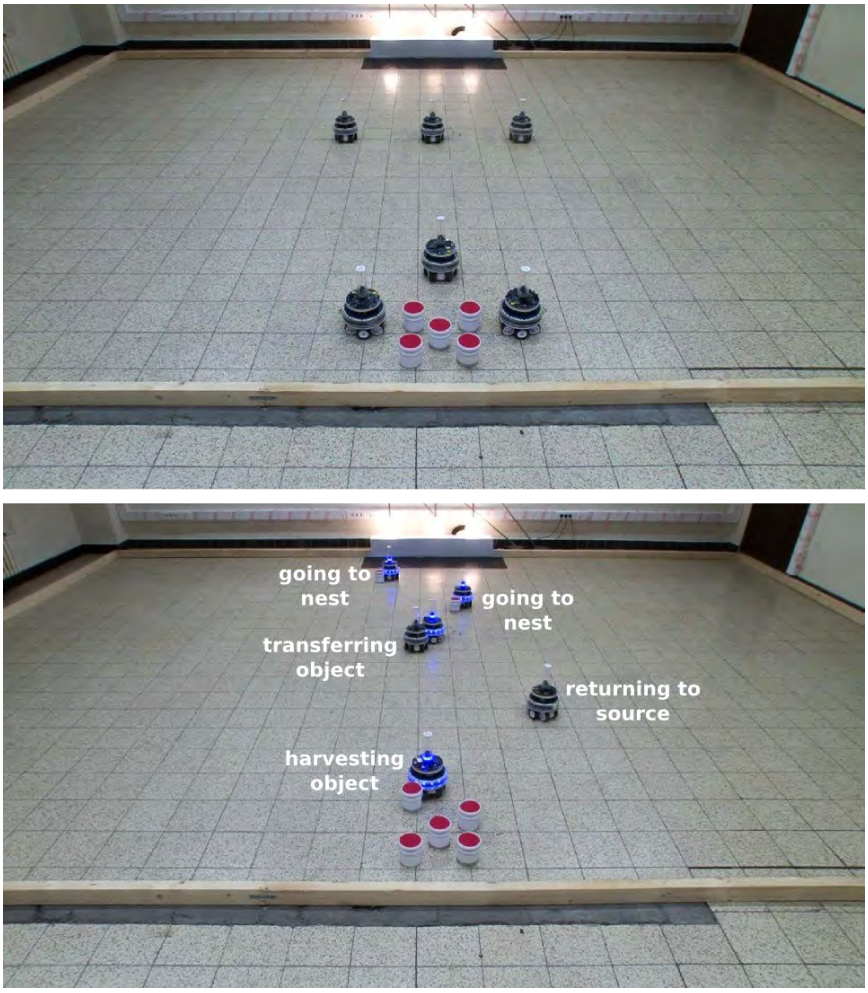

Figure 6.12: Snapshots of a real-robot experimental run in which the robots employ task partitioning. Top: initial configuration. 3 robots are positioned in proximity of the source, the rest at half the distance between source and nest. Bottom: situation after 90 seconds. Two robots transferred their object: one is returning to the source, the other is harvesting another object. The two robots that received an object are heading to the nest. The remaining two robots are in the process of transferring an object.

Results of the experiments. Figure 6.13 (left) reports the total number of objects transported to the nest by the robots using the two strategies for both the real-robots and the simulation-based experiments. Figure 6.13 (right) provides a summary of the actions performed by the robots. Each pair of bars reports the percentage of time, computed over all the experimental runs, that the robots spent performing each action for the non-partition strategy (white bars) and the partition

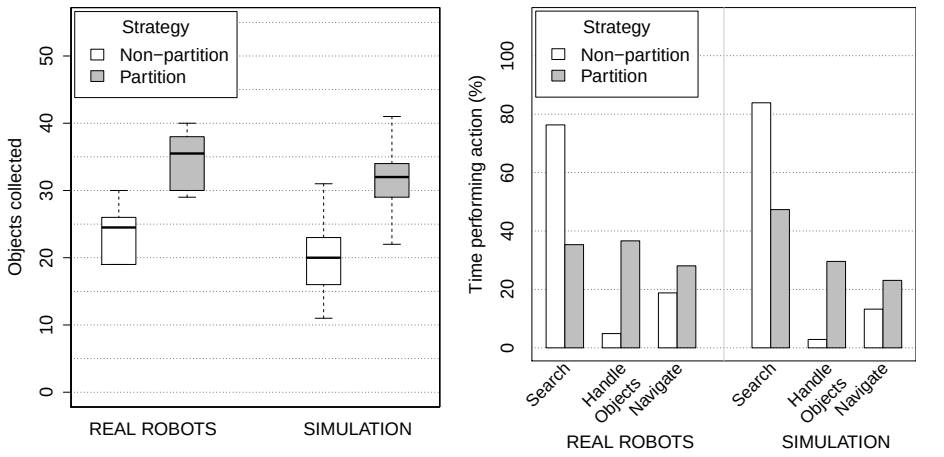

Figure 6.13: Total number of objects collected using the two strategies (left) and actions performed by the robots (right). In the right-hand side plot, each bar reports the percentage of time, computed across all the runs, the robots spent performing the corresponding action. The actions are grouped into three sets. *Search* accounts for the time spent performing random walk and neighborhood search. *Handle Objects* accounts for the time spent gripping objects and includes the time spent waiting and transferring objects in case the robots employ the partition strategy. *Navigate* accounts for the time spent going towards the nest with an object or trying to reach a position of the environment using the odometry estimate.

strategy (gray bars). The actions of the robots are sampled every second during the course of each run. The actions are grouped into three sets. *Search* includes the time the robots spent performing random walk and neighborhood search. *Handle Objects* includes the time the robots spent approaching and gripping objects. In case the robots employ the partition strategy, object handling also includes the time spent waiting for the robot that received the object and the time required to perform the transfer. *Navigate* accounts for the time the robots spent going to the nest while carrying an object, or trying to reach a position of the environment using the odometry estimate.

The results reported in Figure 6.13 indicate that there are numerical differences between the results obtained in simulation and reality. However, the simulation replicates the trends observed in reality for what concerns the performance of the swarm and the actions performed by the robots. In both cases, the swarm collects more objects when employing the partition strategy. Additionally, the robots spend more time handling objects when they use the partition strategy and searching in the environment when they use the non-partition strategy.

Table 6.1: Frequency at which the robots get lost in the real-robot and simulation experiments. For the robots employing the partition strategy, the frequency at which the robots get lost is reported also for the cases in which the robots estimated position is relative to the locations where objects are transferred. The table reports, for each measure, the 95% confidence interval on the value of the mean. The number of significant digits used to express frequencies depend on the total number of runs used to compute such frequencies (6 real-robot and 200 simulation runs for each strategy).

Measure	**Real robots**	**Simulation**
Non-partition strategy		
Get lost frequency - source	0.7 - 0.8	0.76 - 0.79
Partition Strategy		
Get lost frequency - source	0.0 - 0.2	0.13 - 0.15
Get lost frequency - transfer location	0.1 - 0.2	0.11 - 0.12

Therefore, task partitioning is costly because of its overheads due to object transfer, while performing transportation as an unpartitioned task is costly due to searching.

Table 6.1 reports, for both strategies, the frequency at which the robots get lost measured in simulation and in the real-robot experiments. The table reports, for each entry, the 95% confidence interval on the value of the mean. As for the foraging performance and the actions performed by the robots, we observe a numerical difference in the results comparing simulation and reality. However, we deem the difference to be sufficiently small for our purposes. A similar table, reporting additional measures not presented here, is available with the supplementary material (see Annex B).

Figure 6.14 reports the empirical distribution of the time a robot takes to find the source after getting lost, for the non-partition strategy (left) and the partition strategy (right). The black line plots the data collected in simulation, the light-gray line the data collected in the real-robot experiments. The graphs show that, on average, in simulation the robots take more time to find the source after getting lost. This can explain the discrepancy between simulation and reality observed in Figure 6.13, in relation to the number of objects collected and actions performed by the robots. However, the trends followed by the curves in simulation and reality are very similar and once again we are satisfied by the behavior of the simulator. Note that the robots using the partition strategy take less time, on average, to find the source after they

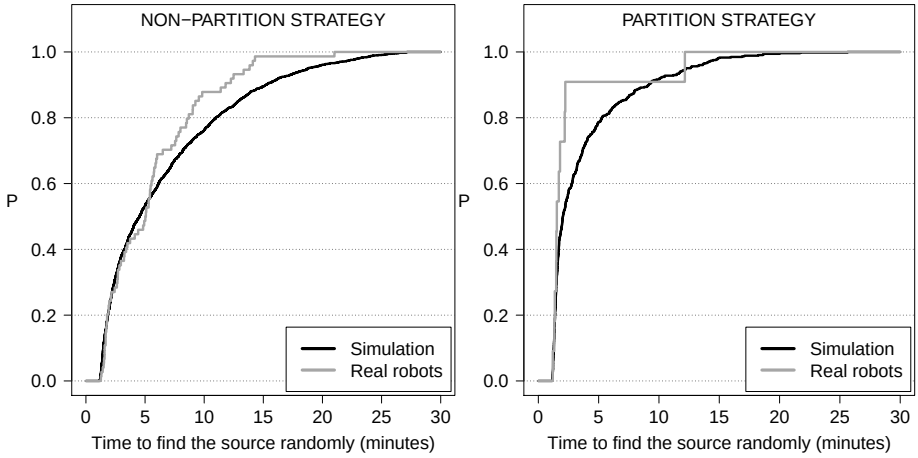

Figure 6.14: Empirical distribution (P) of the time required to find the source when searching randomly. The plotted data only includes the time needed to find the source after a neighborhood search failed (i.e., it does not include the time required to find the source for the first time). The left-hand side plot reports the data collected when the non-partition strategy is employed by the robots, the right-hand side plot the data collected when the partition strategy is employed.

got lost than the robots employing the non-partition strategy. This indicates that when the partition strategy is used, the robots get lost in locations that are closer to the source compared to the non-partition strategy.

To summarize, we observed some discrepancies between the numerical results obtained in simulation and real-world experiments. For the purpose of the work presented in this chapter, we consider the differences to be acceptable. The trends we observe in the real-robot experiments are reproduced in simulation and we are therefore confident that the simulation indeed captures the properties of the studied system.

6.6 Simulation Experiments and Results

In this section, we describe the simulation-based experiments that we carried out to test our approach for autonomous task partitioning in the system described. The experiments are divided into five sets, each aiming to test different aspects of the studied system. When not differently specified, the experimental parameters are set as follows (refer to Table 6.2). The experimental environment measures 6.7 m in width and 4.5 m in length (see Figure 6.7). Source and nest are positioned at

Table 6.2: Default experimental settings.

Parameter	**Value**
Environment size	$W = 6.7\,\text{m}$ by $L = 4.5\,\text{m}$
Source-to-nest distance	$D = 4\,\text{m}$
Duration of an experimental run	20 hours
Swarm size	4, 10, 20
Experimental runs per setting	20
Default error parameter σ	$\bar{\sigma} = 0.0134$

a distance $D = 4\,\text{m}$ from each other. Each experimental run lasts a total of 20 simulated hours. At the beginning of each run, the robots are positioned inside the nest, with a random orientation. We test three different swarm sizes: 4, 10, and 20 robots. For each experimental condition we run 20 randomly seeded simulations.

The rest of this section is organized as follows. In Section 6.6.1, we present a set of experiments that have the goal of evaluating the basic properties of the system, and of selecting the values of the parameters α and ε, used in the cost-based partitioning algorithm. In Section 6.6.2, we test the effect of the size of the environment on the studied system. In Section 6.6.3, we describe experiments in which we study different values of the source-to-nest distance D, corresponding to different lengths of the transportation task. In Section 6.6.4, we present experiments in which we study the effect of heterogeneity among the robots of the swarm. Finally, in Section 6.6.5, we evaluate a case in which the environmental conditions vary in time, and we discuss the trade-off between exploitation and exploration.

6.6.1 Basic Properties

The experiments described here have two main goals. The first goal is to assess the best way of partitioning the transportation task in relation to the size of the swarm and the accuracy of the odometry system. We perform experiments in which we test the reference algorithms with swarms of different sizes (4, 6, 8, 10, 15, and 20 robots) and for different values of σ (0.0, $0.2\bar{\sigma}$, $0.5\bar{\sigma}$, $\bar{\sigma}$, $1.2\bar{\sigma}$). The second goal of the experiments is to select a value for the parameters of the cost-based partitioning algorithm, and to compare the algorithm to the reference algorithms. We test different versions of the cost-based partitioning algorithm, that vary for what concerns the value of the parameter α, used to compute the cost estimates according to Equation (6.3), and ε

Table 6.3: Partition distance of the best performing fixed algorithm for different values of the odometry error parameter σ and different swarm sizes. NP indicates that the never-partition algorithm is the best performing.

	$\sigma = 0.0\bar{\sigma}$	$\sigma = 0.2\bar{\sigma}$	$\sigma = 0.5\bar{\sigma}$	$\sigma = \bar{\sigma}$	$\sigma = 1.2\bar{\sigma}$
4 robots	NP	NP	2.0 m	1.5 m	1.5 m
6 robots	NP	2.5 m	2.0 m	1.5 m	1.5 m
8 robots	NP	2.5 m	2.0 m	1.5 m	1.5 m
10 robots	NP	2.5 m	2.0 m	1.5 m	1.5 m
15 robots	2.0 m	2.0 m	1.5 m	1.5 m	1.5 m
20 robots	1.5 m	1.5 m	1.5 m	1.5 m	1.0 m

of the ε-Greedy algorithm. The parameter α is selected from the set $\{0.001, 0.1, 0.25, 0.9, 1.0\}$, while ε from the set $\{0.0, 0.05, 0.15, 0.25, 0.5\}$. The remaining experimental settings are as reported in Table 6.2.

The effect of the odmetry error and swarm size. The performance of the fixed algorithms in relation to the size of the swarm and the odometry error provides insights about the basic properties of the system. We refer to the *performance of an algorithm* as the total number of objects transported to the nest by the robots when that algorithm is employed.

Table 6.3 reports, for the different values of σ and the different swarm sizes, the partition length value of the best performing fixed algorithm in the corresponding setting. In the table, NP indicates that the never-partition algorithm is the best performing. The complete results of these experiments are available with the supplementary material (see Annex B). The results reported in Table 6.3 highlight two aspects. First, they confirm that the larger the odometry error, the more task partitioning becomes advantageous. In fact, for a given swarm size (i.e., a given table row), which fixed algorithm performs best varies in relation to the value of σ: the higher the value, the smaller the partition length value of the best performing fixed algorithm. In other words, if odometry is not accurate, the number of sub-tasks should be increased and their length decreased. Conversely, for small values of σ it is preferable to use few, long sub-tasks. These results confirm that task partitioning is beneficial to reduce the negative impact of the odometry error.

The second aspect highlighted by the experiments is that, given certain odometry error conditions (i.e., a given column in Table 6.3), in large swarms it is preferable to partition the given task into many,

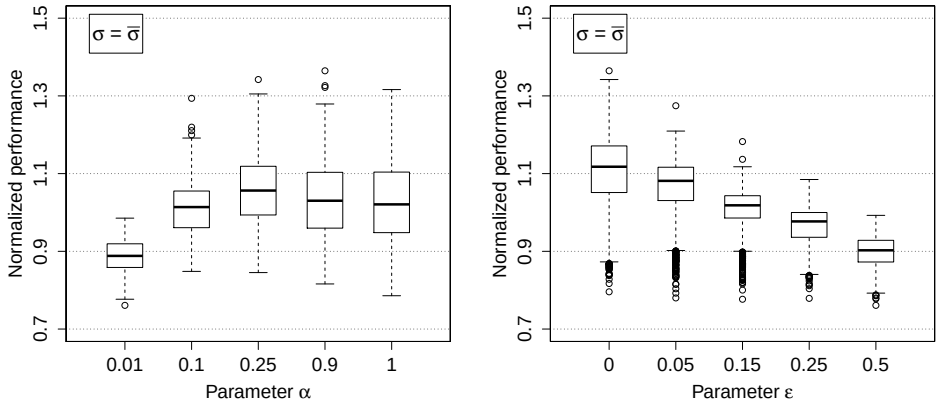

Figure 6.15: Effect of the parameter α (left) and ε (right), for $\sigma = \bar{\sigma}$. The data reported in the plots is computed as follows. We calculate the average performance P_N, computed over all the values of ε and α, for each swarm size N. The value P_N is used to normalize the performance values recorded for swarms of size N. Each box in the left-hand side plot aggregates the normalized performance for different swarm sizes and values of ε, for a given value of α. Analogously, in the right-hand side plot each box aggregates the normalized performance for different swarm sizes and values of α, for a given value of ε.

short sub-tasks. This is due to the fact that physical interference is higher in large swarms than in small ones. Task partitioning distributes the robots along the path between source and nest, thus diminishing interference among robots.

Parameter selection. As mentioned, in addition to evaluate the effect of the odometry error and of the swarm size, the experiments of the first set are also used to select the values of the parameters ε and α. Figure 6.15 summarizes the effect of the parameters ε and α on the performance of the cost-based partitioning algorithm, for the case in which $\sigma = \bar{\sigma}$. The complete results of the experiments are reported with the supplementary material (see Annex B).

The data reported in Figure 6.15 is computed as follows. First, we calculate the average performance P_N, computed over all the values of ε and α, for each swarm size N. Each box in the plots of Figure 6.15 aggregates the performance of swarms of different size, normalized using the corresponding value P_N.[6] The left-hand side plot of Figure 6.15 reports the normalized data for different values of α (i.e., each bar aggregates all the values of ε and swarm size N). Analogously, the plot

[6] P_N divides the performance.

on the right-hand side of Figure 6.15 reports the data for different values of ε. The plot on the left-hand side shows the overall effect of α, the one on the right-hand side the overall effect of ε. The plots show that the highest levels of performance are obtained for $\alpha = 0.25$ and $\varepsilon = 0$. We select these values and utilize them in all the experiments presented in this chapter, with the exception of the ones presented in Section 6.6.5.

The value 0 for the parameter ε corresponds to a purely exploiting version of the ε-Greedy algorithm: the algorithm always selects the element L_i associated to the minimal cost estimate $\hat{C}_i$. This is not surprising given the nature of the experiments. In fact, the only variations occurring in the system are those introduced by the robots depositing objects along the path from source to nest. The remaining conditions, such as the number of robots, the accuracy of the odometry system, the size of the environment, etc. do not vary in time. Consequently, no exploration is needed and a pure exploiting version of the ε-Greedy algorithm performs well. This result confirm what observed in the previous chapter, in which we pointed out that non-exploring versions of the algorithms lead to a higher performance in stationary conditions (see Section 5.5.2). Further considerations about the trade-off between exploration and exploitation are presented in Section 6.6.5.

Performance of the cost-based partitioning algorithm. Figure 6.16 reports the performance of the cost-based partitioning algorithm and of three reference algorithms for swarms of 4, 10, and 20 robots. The top plot reports the results for low error values ($\sigma = 0.2\bar{\sigma}$), the bottom plot for $\sigma = \bar{\sigma}$. For clarity, we did not report the performance of all the reference algorithms. For a given experimental setting (i.e., odometry error and swarm size), the fixed algorithm reported in the figure is the one performing the best in that setting. The performance of the fixed algorithm is an upper bound for the performance of a swarm that uses task partitioning. In general to reach such a performance the swarm needs prior knowledge about the environment. If the robots do not have such knowledge, they have to utilize other partitioning algorithms which, in general, perform worse than the fixed algorithm.

The results reported in Figure 6.16 highlight several aspects. The performance of all the algorithms increases with a decreasing odometry error. The smaller σ, the smaller the odometry error and the higher the frequency at which the robots find the source when performing neighborhood search. As the error increases, the performance of each algorithm decreases. The never-partition algorithm performs well when odometry is accurate and the swarm is composed of few robots. For

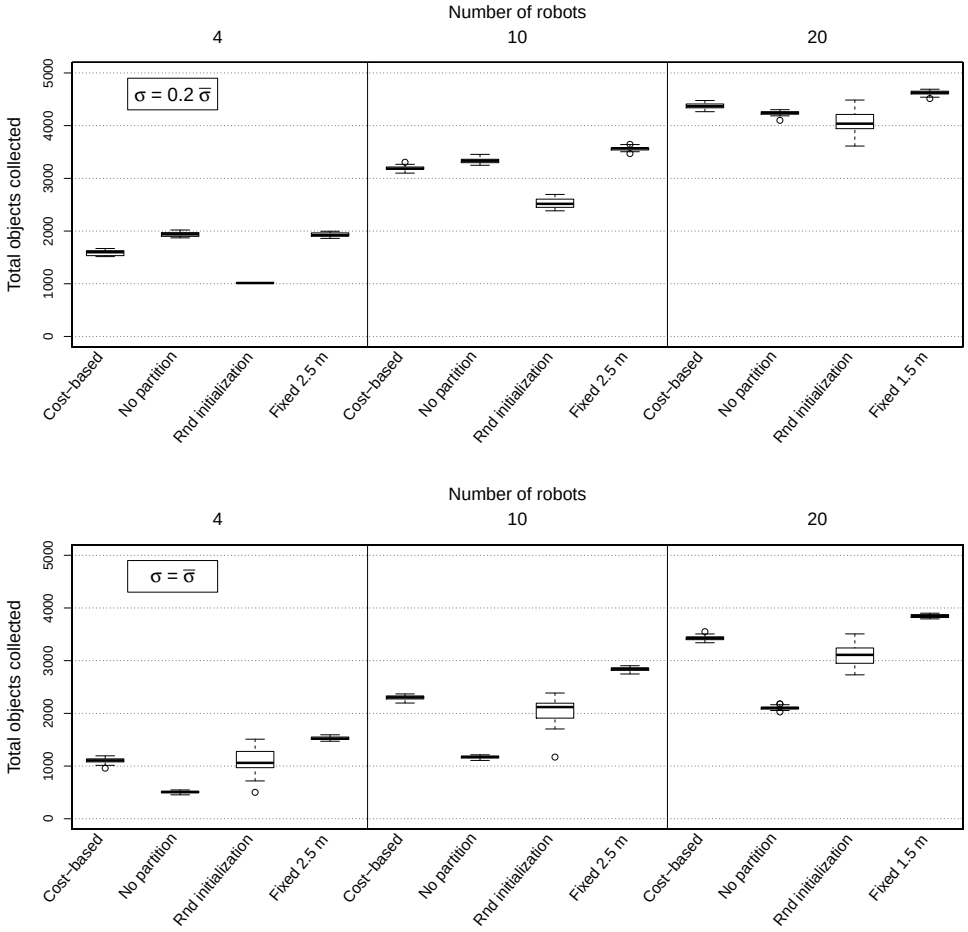

Figure 6.16: Objects collected by the swarm when different partitioning algorithms are employed. Each group of boxes reports data collected with a swarm of a given size (see top axis). The plot on top reports the data for $\sigma = 0.2\bar{\sigma}$, the plot at the bottom for $\sigma = \bar{\sigma}$. The fixed algorithms reported in figure are the ones performing the best in the corresponding setting. The performance of the fixed algorithm is an upper bound for the performance of a swarm that uses task partitioning.

increasing swarm sizes, the algorithms that utilize task partitioning become more advantageous. As already pointed out, this is due to interference: the robots employing the never-partition algorithm travel all the way from source to nest and interfere with each other's paths. The negative effect of interference on the algorithms increases for a decreasing odometry error: the robots get lost few times and the traffic along the path between source and nest is high. The cost-based partitioning algorithm performs well in the majority of the tested con-

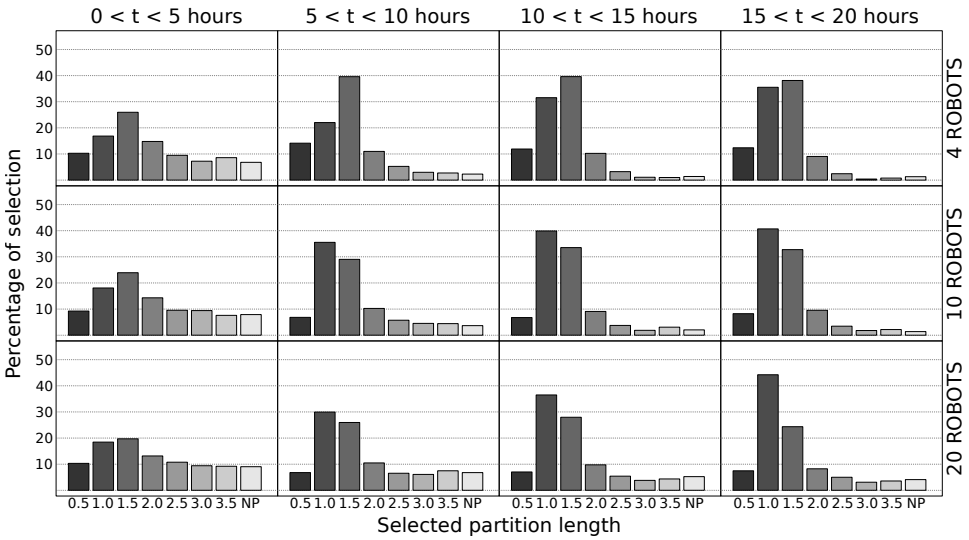

Figure 6.17: Partition length values selected in time by robots employing the cost-based partitioning algorithm, for the case $\sigma = \bar{\sigma}$. Each plot reports the percentage of selection of each value in a time window of 5 hours (see labels on top).

ditions, which indicates that the partitioning strategy utilized by the swarm suits the specific conditions in which the robots operate.

Partitioning strategy employed by the robots. Figure 6.17 reports the partition length values selected in time by robots employing the cost-based partitioning algorithm, for $\sigma = \bar{\sigma}$. The plot reports the percentage of times each value was selected by the robots. The percentages are computed across the 20 experimental runs. The sequence of plots in each row reports the data collected with swarms of a certain size, from top to bottom: 4, 10, and 20 robots. Each column reports the data relative to a time period of 5 hours, corresponding to a quarter of the duration of each experimental run.

Each sequence of plots shows that there is an initial phase (see plots in the first column) during which the robots sample the different values. In time (plots from left to right), the robots converge towards certain values of the partition length. The values selected the most by the robots in the final part of the experiment depend on the size of the swarm: the larger the swarm, the lower the partition length values selected by the robots (compare the plots in the last column). Therefore, the cost-based partitioning algorithm responds to a growing swarm size by reducing the length of the sub-tasks. The results obtained by the fixed algorithms demonstrate that this is advantageous

to diminish physical interference.

Analogously, Figure 6.18 reports the partition length values selected in time by robots employing cost-based partitioning algorithm, for $\sigma = 0.5\bar{\sigma}$. The plots confirm the trends observed in Figure 6.17: in large swarms the robots select lower values of the partition length. Comparing to the case reported in Figure 6.17 however, the overall choice of the robots is oriented towards higher values of the partition length. This is due to the lower error in the odometry system, compared to the previous case.

These results confirm that the robots employing the cost-based partitioning algorithm decide how to partition the transportation task autonomously, on the basis of the environmental conditions (odometry error and physical interference). This is an important result since the task partitioning mechanisms we propose are not explicitly built to take into account such conditions. Recall that the robots do not take direct measures of the frequency at which they get lost, the odometry error, or the perceived interference. Instead, they only estimate costs of performing tasks and sub-tasks. The strategy employed by the robots to partition the transportation task varies with the environmental conditions in a self-organized manner, due to the impact of such conditions on the cost estimates. In other systems, in which the factors impacting costs might be others, the very same task partitioning methods proposed here can be employed without modifications and, most importantly, without the need to know what these factors are.

Objects throughput. The plots reported in Figure 6.19 show the evolution of the throughput in time for swarms composed of 4 (Figure 6.19a), 10 (Figure 6.19b), and 20 (Figure 6.19c) robots. The throughput is measured as the number of objects delivered to the nest in one hour. The values reported in each plot are the medians computed over 20 experimental runs, for $\sigma = \bar{\sigma}$. The throughput of the cost-based partitioning algorithm and of three reference algorithms is reported in each plot.

The results reported in Figure 6.19 show that the throughput of the reference algorithms rapidly stabilizes around a given value. The throughput obtained by the robots employing the cost-based partitioning algorithm, on the other hand, grows slowly in time as an effect of the convergence of the robots to a certain value of partition length (refer to the results reported in Figure 6.17).

We speculate that the low growth speed is due to the fact that the actions performed by each robot influence the cost estimation process of the other robots. For example, a robot R that finds objects on the

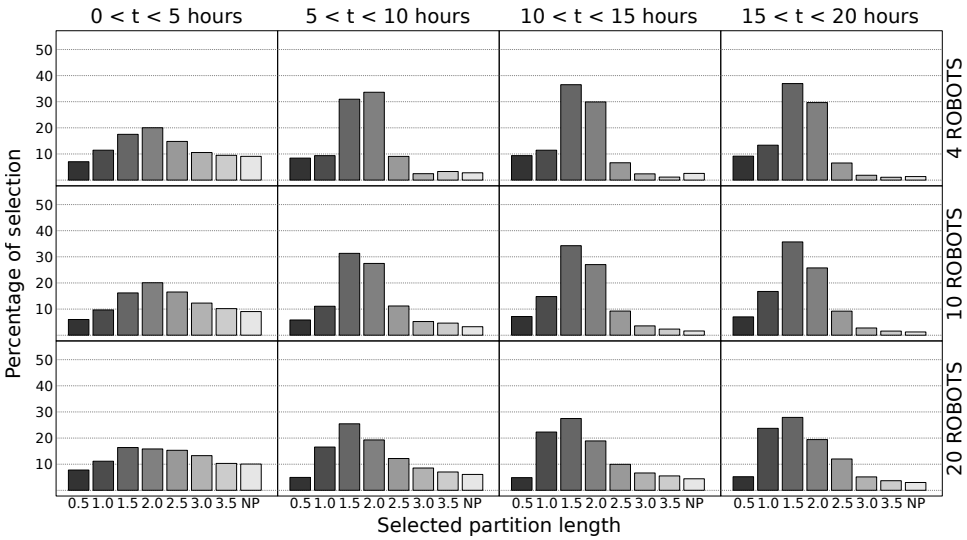

Figure 6.18: Partition length values selected in time by robots employing the cost-based partitioning algorithm, for the case $\sigma = 0.5\bar{\sigma}$. Each plot reports the percentage of selection of each value in a time window of 5 hours (see labels on top).

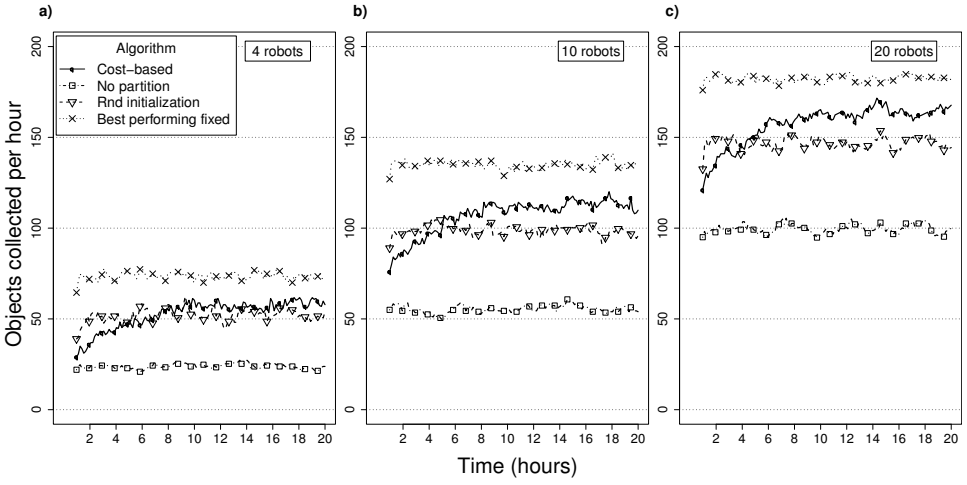

Figure 6.19: Evolution in time of the throughput obtained with different algorithms. The throughput is computed as objects delivered to the nest in an hour of experiment. Each curve reports the median value, computed over 20 experimental runs.

way between source and nest (i.e., not at the source location) needs another robot R' constantly delivering objects there. If the robot R' gets lost or selects a different partition length value, the robot R may not be able to find objects anymore. As a consequence, the robot R is

likely to assign a high cost to the selected value of the partition length, therefore penalizing its selection for the future. This influence on each other's selection process requires the robots to sample the environment for a longer time in order to rule out the side effect of cost estimation errors and to coordinate their actions. On the contrary, the selection of the partition length done by the reference algorithms is not based upon cost estimates. Therefore, the robots do not interfere with each other's selection process and the swarm rapidly reaches a steady throughput.

6.6.2 Size of the Environment

The goal of the experiments described here is to test the behavior of the system in environments of different size. In a large environment, the robots take longer, on average, to locate the source at the beginning of the experiment and in case they get lost (i.e., random walk is costly). We test two environments: a $W = 4.5\,\text{m}$ by $L = 4.5\,\text{m}$ environment (*small environment*) and a $W = 6.7\,\text{m}$ by $L = 10.0\,\text{m}$ environment (*large environment*). In both cases, the source-to-nest distance D is $4.0\,\text{m}$ and $\sigma = \bar{\sigma}$. The remaining experimental parameters are as reported in Table 6.2.

Figure 6.20 reports the performance of the cost-based partitioning algorithm and of three reference algorithms for different swarm sizes. The top plot in Figure 6.20 reports the results of the experiments performed in the small environment, the bottom plot in Figure 6.20 reports the results obtained in the large environment. The results of the experiments indicate that the cost-based partitioning algorithm performs well across environments of different sizes.

Figure 6.21 reports, for different swarm sizes and environments, the values of partition length selected by the robots in the final quarter of the experiment (last 5 hours). Plots in the same row refer to the same environment: small environment on top and large environment at the bottom. Plots in the same column report data for the same swarm size, from left to right: 4, 10, and 20 robots. The plots highlight a trend in the partition length values selected by the robots. For a given swarm size, the selected values are influenced by the size of the environment: the larger the environment, the lower the partition length values selected by the robots. This indicates that the cost-based partitioning algorithm identifies the high cost that derives from getting lost in larger environments and adapts the way the task is partitioned accordingly, by reducing the distance traveled by the robots and consequently the frequency at which they get lost. Analogously to what pointed out in the previous section, this is a significant result since the robots have no notion of the size of the environment in which they operate and yet

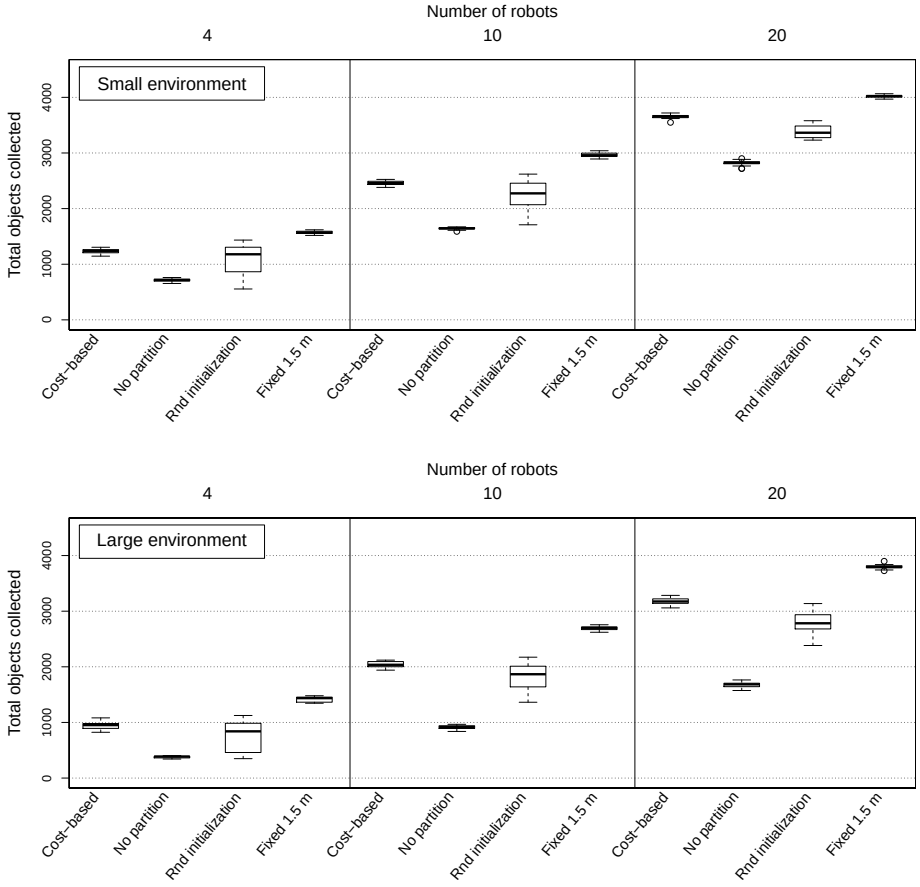

Figure 6.20: Objects collected by the swarm when different partitioning algorithms are employed. Each group of boxes reports data collected with a swarm of a given size (see top axis). The plot on top reports the data collected in the small environment, the plot at the bottom the data collected in the large environment.

the partitioning strategy employed by the swarm varies in relation to the environment in which foraging is performed.

6.6.3 Distance to the Source

The goal of the set of experiments presented in this section is to study the effect of a different source-to-nest distance D (i.e., a different length of the overall task). The experiments are performed in the large environment ($W = 6.7\,\text{m}$ by $L = 10.0\,\text{m}$), with $\sigma = \bar{\sigma}$. The value of the source-to-nest distance D is set to 3.0 m and 6.0 m. The values of the remaining experimental parameters are as reported in Table 6.2.

Figure 6.22 reports the performance of the cost-based partitioning

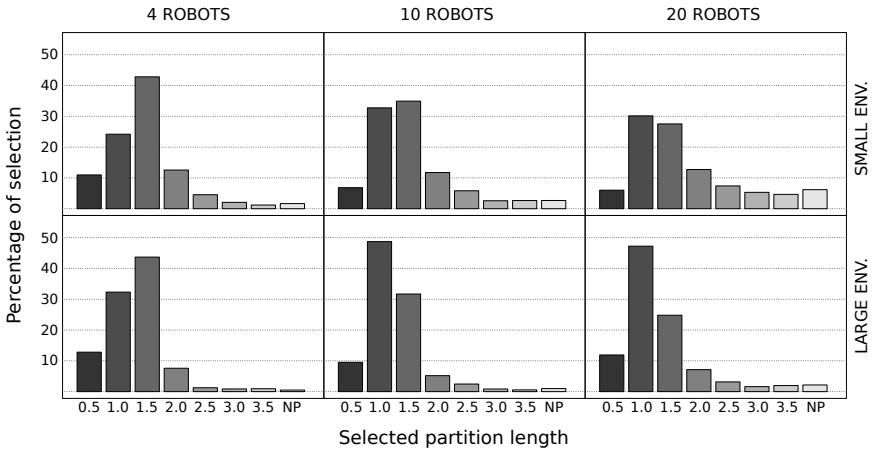

Figure 6.21: Partition length values selected in time by robots employing the cost-based partitioning algorithm. The plots report data collected for $\sigma = \bar{\sigma}$. Each plot reports the percentage of selection of each value in the final quarter of the experiments (last 5 hours), for a given swarm size and environment.

algorithm and three reference algorithms for different swarm sizes. The top plot displays the results for the case in which $D = 3.0\,\text{m}$, the bottom plot for the case in which $D = 6.0\,\text{m}$.

The performance obtained with a given algorithm for $D = 3.0\,\text{m}$ is higher than the corresponding performance for $D = 6.0\,\text{m}$. In the latter case, the objects must be transported for a longer distance and the throughput of the swarm is inferior. Additionally, in the case of the never-partition algorithm, the longer distance between source and nest increases the probability that a robot gets lost, thus lowering further the throughput. For the remaining algorithms, which utilize task partitioning, a longer distance corresponds to a higher number of sub-tasks and therefore higher overheads due to object transfer.

Figure 6.22 shows that, for $D = 3.0\,\text{m}$, the robots that use the cost-based partitioning algorithm perform well for all the tested swarm sizes. This is not the case for $D = 6.0\,\text{m}$: the performance of the cost-based partitioning algorithm is often close, or even inferior, to the performance of the random initialization algorithm.

Figure 6.23 reports the evolution of the throughput in time for swarms composed of 4 (Figure 6.23a), 10 (Figure 6.23b), and 20 (Figure 6.23c) robots. The figure reports the data for $D = 6.0\,\text{m}$. The plots show that in the final part of the experiment the throughput of the cost-based partitioning algorithm is higher than the throughput of the random initialization algorithm. However, the throughput grows

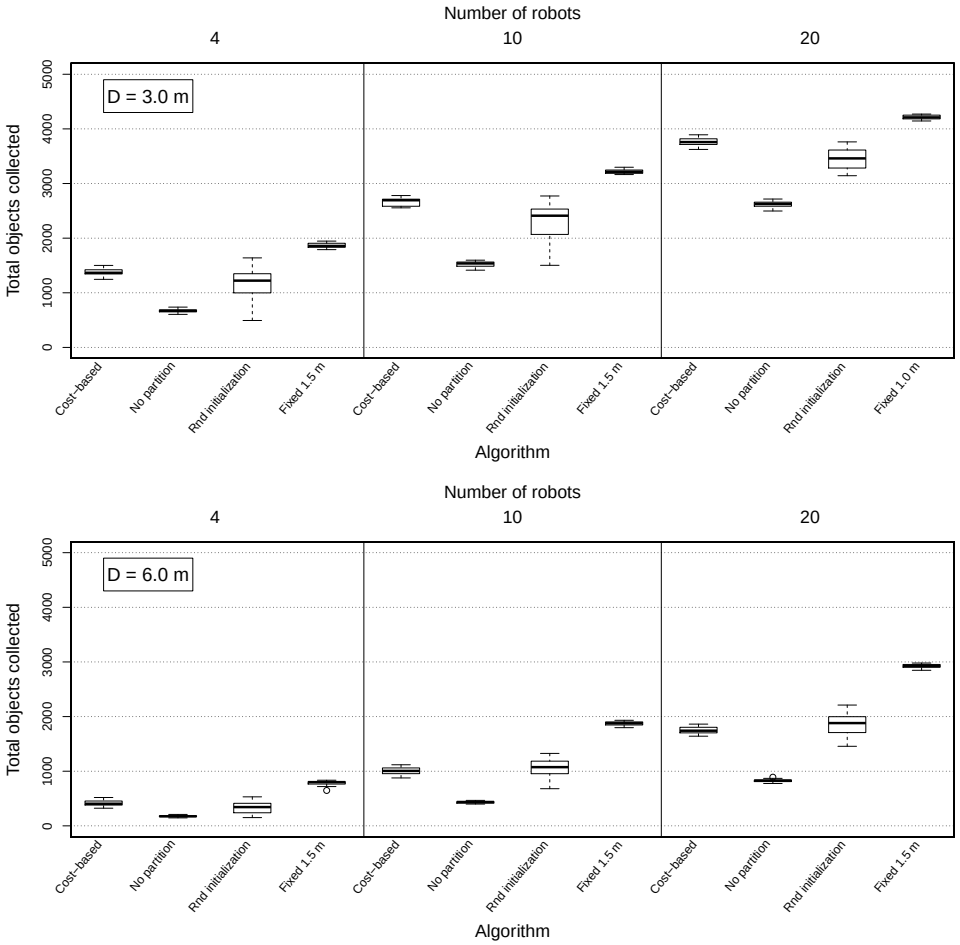

Figure 6.22: Objects collected by the swarm when different partitioning algorithms are employed. Each group of boxes reports data collected with a swarm of a given size (see top axis). The plot on top reports the data for $D = 3.0\,\mathrm{m}$, the plot at the bottom for $D = 6.0\,\mathrm{m}$.

very slowly in time and this motivates the relatively low performance observed in Figure 6.22.

6.6.4 Heterogeneity in the Robot Swarm

The goal of the experiments described in this section is to assess the impact of heterogeneity on the cost-based partitioning algorithm. Our focus is on heterogeneity impacting the efficiency in performing a task. To this end, we act upon the odometry system of the robots: the accuracy of the odometry system varies from robot to robot. Therefore, different robots might be more or less successful in reaching a position

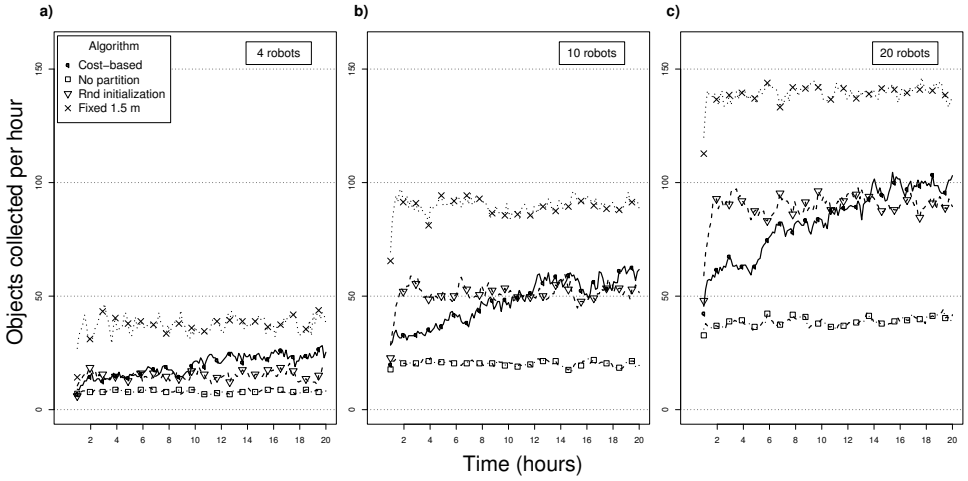

Figure 6.23: Evolution in time of the throughput obtained with different algorithms, for $D = 6.0\,\text{m}$. The throughput is computed as objects delivered to the nest in an hour of experiment. Each curve reports the median value, computed over 20 experimental runs.

of the environment using odometry. Heterogeneity in the odometry system of the robots is a realistic assumption. Indeed, in the real-robot experiments presented in Section 6.5, we observed that the success rate at finding the objects source varies from robot to robot.

Here, we simulate heterogeneity in the odometry system using different values of σ in the swarm. We divide the swarm into two groups, one in which $\sigma = 0.2\bar{\sigma}$ (we refer to robots of this group as *low-error*), the other in which $\sigma = \bar{\sigma}$ (*default-error* robots). We perform experiments using 10 robots, varying the percentage of low-error robots in the swarm. We test two conditions: one in which there are 2 low-error robots in the swarm and one in which the low-error robots are 8. The remaining experimental parameters take the values reported in Table 6.2.

Figure 6.24 reports the performance of the cost-based partitioning algorithm and of four reference algorithms. The plot on the left-hand side reports the case in which the low-error robots are 2, the plot on the right-hand side the case in which the low-error robots are 8. The results reported in Figure 6.24 indicate that the cost-based partitioning algorithm performs well in the tested conditions, showing that it is robust with respect to heterogeneity within the swarm.

Figure 6.25 reports the partition length values selected by the robots employing the cost-based partitioning algorithm in the final quarter of experiment (last 5 hours). Each plot reports the percentage of times,

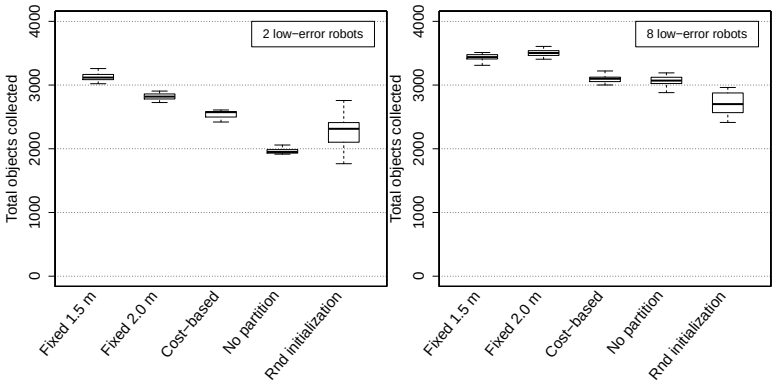

Figure 6.24: Experiments on heterogeneity: objects collected by the swarm for different partitioning algorithms. The plot on the left-hand side reports the data for the case in which the low-error robots are two, the plot on the right-hand side for the case in which their number is eight.

computed over 20 experimental runs, each value was selected by the robots. In the figure, plots in the same column refer to the same number of low-error robots in the swarm: 2 robots in the plots on the left-hand side and 8 robots in the plots on the right-hand side. The top plots report the partition length values selected by the low-error robots, the bottom plots the values selected by the default-error robots.

The results reported in Figure 6.25 show that the partition length values selected by the robots vary in relation to their odometry accuracy. The low-error robots select higher partition length values compared to the values selected by the default-error robots. In other words, the way a robot partitions each task depends on the characteristics of the robot itself. Once more this result is not obtained through an explicit mechanism that, for example, relates the selected partition length to the odometry accuracy of a robot. All the robots are homogeneous for what concerns the mechanism implemeting task partitioning. Different partitioning strategies for different robots result from the dependency of the costs on the odometry accuracy of a given robot.

6.6.5 Adaptivity to Variable Conditions

In all the experiments presented so far, the cost-based partitioning algorithm obtains its best performance with a purely exploiting version of the ε-Greedy algorithm (i.e., for $\varepsilon = 0$). This is due to the fact that, in all the experiments, we test static conditions. The only variations occurring in the environment are introduced by the robots depositing objects. However, exploration may result useful when properties of

Figure 6.25: Experiments on heterogeneity: partition length values selected by the robots over time, when the cost-based partitioning algorithm is employed. Each plot reports the percentage of selection of each value in the final quarter of experiment (last 5 hours). The plots in the first row report the values selected by the low-error robots, the plots in the second row the values selected by the default-error robots. The plots in the first column report the data for the case in which the number of low-error robots is two, the plots in the second column report the data for the case in which their number is eight.

the environment vary in time. Given the way the ε-Greedy works, exploration may be beneficial only if one of the costs $\hat{C}_i$ decreases and renders the associated element L_i advantageous over the one that previously had the lowest cost associated.

Suppose, for example, that the environmental conditions initially favor low values of the partition length and that the robots identified low values as a good choice. Suppose that a change occurs in the environment, that renders favorable higher values of the partition length. If the effect of the change is to reduce the cost associated to high values of partition length, without increasing the cost associated to low values, a purely exploiting version of the ε-Greedy algorithm is unable to detect the change. In fact, as the robots constantly select low values of the partition length, initially identified as the best option, they never sample again high values and cannot detect that these values are now preferable. This is analogous to what observed in the experiments presented in the previous chapter (Section 5.5.2) in which we vary the interfacing time at the cache. A variation of the interfacing time from a low to a high value is detected by the robots, since they initially

use the interface and therefore they can directly perceive the variation. Conversely, the robots have problems detecting a variation from a high to a low value: the interface is initially not used and the robots cannot directly perceive the variation of the interfacing time.

To test an analogous situation, we perform an experiment in which we introduce a variation in the environmental conditions. The variation consists in incrementing the distance at which the robots perceive the objects. A broader perception diminishes the probability that a robot gets lost: objects can be perceived from farther away and therefore odometry errors have a smaller impact.

We perform the experiments with swarms of 10 and 20 robots. We vary the perception range of the robots from the default value 0.5 m to 1.2 m, when the experimental run reaches half-time. The duration of a run is extended to 40 simulated hours, to allow the cost-based partitioning algorithm to identify good partition length values before the perception change is introduced. We test different versions of the cost-based partitioning algorithm, that vary with respect to the value of the parameter ε. The tested values for the parameter ε are $\{0, 0.3, 0.6\}$, corresponding to progressively increasing degrees of exploration in the algorithm. The odometry error parameter is set to $\sigma = 0.5\bar{\sigma}$. We selected this value as it is the one that highlights the most the effects of exploration and exploitation on the cost-based partitioning algorithm.

Figure 6.26 reports the performance of the cost-based partitioning algorithm and of three reference algorithms for swarms composed of 10 (top) and 20 robots (bottom). In each plot, the group of boxes on the left-hand side reports the data collected in the first half of the experiment, before the perception range is modified. The group of boxes on the right-hand side of each plot reports the data collected in the second half of the experiment, after the perception range has been modified.

The results show that initially the best performing algorithm is the fixed algorithm with a partition length of 2.0 m. Among the different versions of the cost-based partitioning algorithm, the best performing one is the non-exploring ($\varepsilon = 0$). The other versions, which include some exploration, perform worse.

In the second half of experiment, the situation is different: the change in the perception range of the robots makes the never-partition algorithm preferable to the fixed algorithm. Notice however that, for a swarm composed of 20 robots, the relative performance gain of the never-partition algorithm over the fixed algorithm is limited, compared to the case of a swarm of 10 robots. This is again an effect of interference, which is the dominant factor in large swarms.

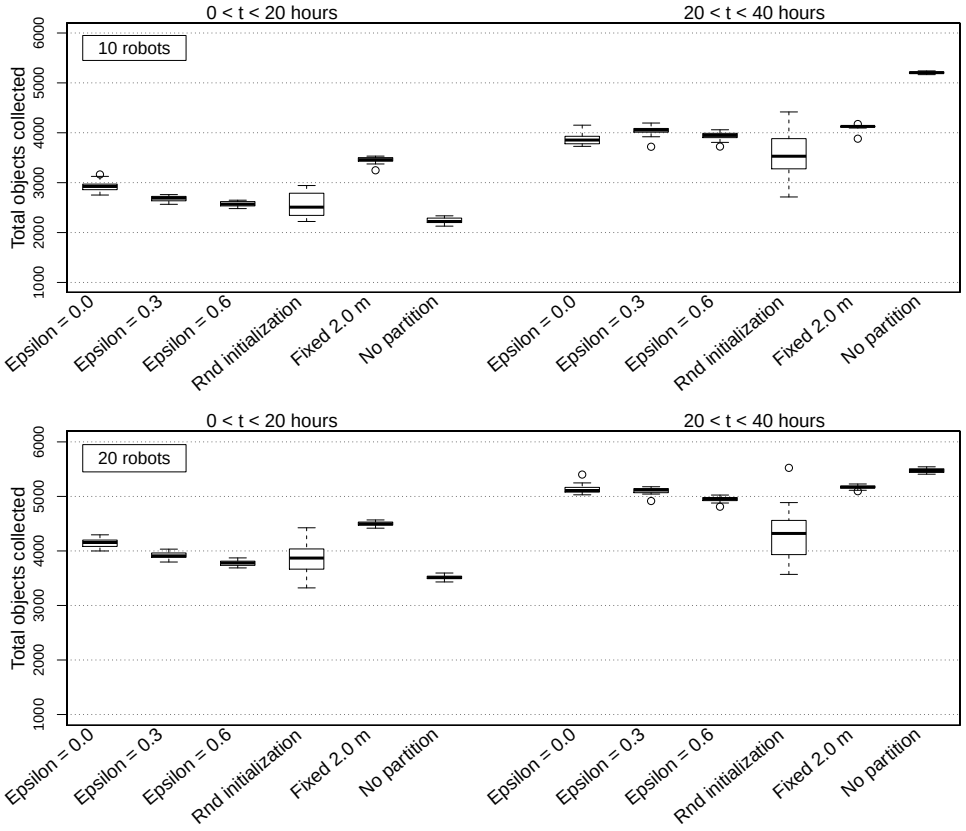

Figure 6.26: Objects collected by the swarm when different partitioning algorithms are employed. In each plot, the first group of boxes refers to the first half of experiment, the second group to the second half. The plot on top reports the results for a swarm of 10 robots, the plot at the bottom for a swarm of 20 robots.

The results obtained by the different versions of the cost-based partitioning algorithm indicate that exploration is beneficial. In the case of a swarm composed of 10 robots, the cost-based partitioning algorithm with $\varepsilon = 0.3$ performs better that the non-exploring version. In the case of a swarm of 20 robots, the performance reached by the two versions is comparable. Exploration is beneficial up to a certain point: for $\varepsilon = 0.6$, performance is always inferior than for $\varepsilon = 0.3$, and therefore exploration and exploitation should be balanced carefully.

Figure 6.27 reports the partition length values selected in time by a swarm of 10 robots. Each row of plots corresponds to a value of ε. The plots in the first column report the frequency at which each partition length value was selected in the second quarter of the experiment (from $t = 10$ to $t = 20$ hours). The plots in the second column reports

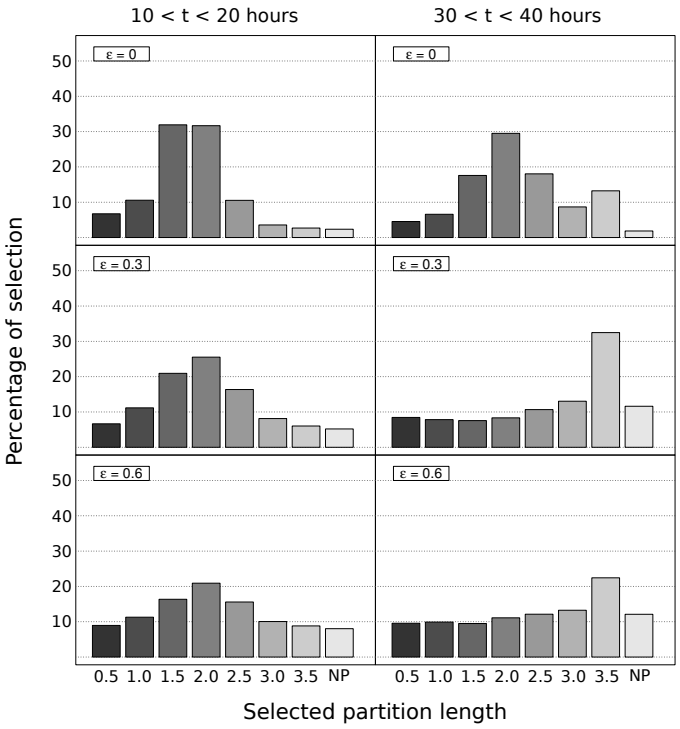

Figure 6.27: Partition length values selected by robots employing different versions of the cost-based partitioning algorithm. The plots in the first column report the percentage of selection of each value in the second quarter of experiment. The plots in the second column report analogous data collected in the final quarter. Each row of plots refers to a different version of the cost-based partitioning algorithm (increasing exploration from top to bottom).

analogous data, collected in the last quarter of the experiment (from $t = 30$ to $t = 40$ hours). The plots highlight different behaviors of the cost-based partitioning algorithm, depending on the value of ε.

The plots in the first column show that all the versions the cost-based partitioning algorithm mostly select values around 2.0 m. The effect of an increased exploration can be observed in the plots (from top to bottom): the peaks flatten and the selected values distribute more evenly. This behavior affects performance negatively, as shown in the left part of Figure 6.26.

The effect of exploration is also visible in the second column of plots reported in Figure 6.27. In case of $\varepsilon = 0.3$ and $\varepsilon = 0.6$, the peak moves from the value of 2.0 m to the value of 3.5 m. This indicates that the cost-based partitioning algorithm detects the change occurring in the environment and reacts selecting a new partitioning

strategy. The non-exploring version of the cost-based partitioning algorithm, on the other hand, continues to select values around 2.0 m. However, the value 3.5 m is selected more often compared to the second quarter of experiment (Figure 6.27 top-left). This indicates that a form of exploration is present in the studied system independently of the value of ε, most likely due to stochasticity in the exploration time when the robots get lost, which introduces variations in the cost estimates. This stochasticity may not be present in other contexts and therefore explicit exploration is an option to consider, in general. The degree of exploration must be selected carefully, since exploration may hinder the performance of the system. For example, the plots for $\varepsilon = 0.6$ (Figure 6.27 last row) show that when exploration is too high, the behavior of the cost-based partitioning algorithm approaches a random behavior. A dominant choice can still be observed, but the cost-based partitioning algorithm does not exploit it efficiently and the performance is affected negatively.

6.7 Summary

In this chapter, we studied the second fundamental building block for implementing complex partitioning strategies: the capability of deciding the amount of work to contribute with a sub-task, which is equivalent to deciding the size of the sub-task. As a testbed, we considered a foraging scenario in which the robots use odometry to estimate their position relatively to the source.

We studied the system and an algorithm based on our approach in simulation-based experiments, under different conditions in terms of magnitude of the odometry error, size of the environment, and length of the transportation task. The results of the experiments confirm the viability of our approach to tackle the studied problem. A remarkable result is that the partitioning strategy employed by the swarm varies across the tested conditions, resulting appropriate in relation to aspects such as physical interference, odometry accuracy, and the size of the environment. None of these aspects is taken into account explicitly by the robots at the time of making decisions, which are instead based on the generic concept of cost.

We presented experiments carried out with real robots that demonstrate that task partitioning can be utilized to contrast the negative effect of odometry errors and to enhance the localization capabilities of the robots. Together with the ones presented in Fontan and Matarić (1996) and Goldberg and Matarić (2002), these experiments are the only ones in which task partitioning is applied to a real-robot scenario.

As a final remark, we pointed out that in certain situations the swarm is slow in converging towards a certain partitioning strategy and this hinders the performance. As we pointed out, this is due to the fact that the interfaces are movable and therefore the choices made by a robot have a strong impact on the work of other robots. In the final chapter of this dissertation we discuss the issue and propose explicit coordination through communication as a possible solution. However, the experiments show that, even without explicit coordination among the robots, the swarm converges with time to an appropriate partition strategy. This indicates that our approach can also be applied in swarms of minimalistic robots, in which communication might not be an option.

Chapter 7

Conclusions

In this final chapter we summarize the main contributions of the research work presented in this dissertation and discuss possible directions for future research.

7.1 Contributions

Swarm robotics is a promising approach for the implementation and control of robotic systems. Swarm robotics systems are best suited for applications that require miniaturization, redundancy and scalability, and that entail danger or risk of failure of individual robots. Examples of applications with such characteristics are space and underwater exploration, mine clearance, surveillance, and military applications. In these contexts, often humans cannot intervene and therefore the robots are required to be robust and flexible in their behavior, in order to autonomously adapt to unforeseen situations and cope with dynamic environments.

The research work presented in this dissertation is driven by the pursuit of flexibility also at the level of the definition of tasks. We envision swarms in which the robots autonomously define how to partition the mission to be accomplished into a set of separate units of work. In this way, not only the behavior of the robots, but also the way a mission is carried out can be adapted to unforeseen situations and changing environments.

In this dissertation, we present research work that makes the first steps in this direction by proposing an approach for autonomous task partitioning in swarms of robots. We focus our study on foraging, which is an abstraction of real-world applications such as the ones mentioned above. However, task partitioning entails benefits that are desirable in many contexts and therefore its application is not limited to foraging.

With this in mind, we based our approach upon the generic concepts of amount of work and cost. Building the task partitioning methods upon these concepts allowed us to:

1. Decouple the task partitioning process and the behavior of the robots;
2. Drive this process towards a generic goal (i.e., reducing costs) and not a specific one (e.g., reducing physical interference among robots);
3. Implement a framework that has the potential to be applied to other contexts.

In our study we identified two decisions that the robots must be able to make. The first consists in deciding whether to use a fixed interface or to bypass it; the second in deciding the amount of work to contribute with a sub-task. These two decisions are the fundamental building blocks for implementing arbitrarily complex partitioning solutions. We thoroughly studied these decisions and we proposed algorithms, based upon our approach, that robots can use to make the decisions autonomously. We performed experiments to test our algorithms and their properties. The results of the experiments demonstrate that our approach is a viable solution to obtain task partitioning in swarms of autonomous robots.

7.2 Future Work

The work presented in this dissertation contributes significantly to the study of task partitioning in swarm robotics. However, our vision of swarms of robots capable of autonomously partitioning any task is still far from being a reality. Many questions remain unanswered and several research directions are still to be explored.

As mentioned in Section 3.1, we focus our study on a subset of possible task partitioning problems. In particular, we study the case in which tasks are partitioned into sequences of interdependent sub-tasks. In general, tasks can also be partitioned into sub-tasks that are not sequentially dependent: the overall task still requires that all the sub-tasks are completed, however, the completion order is not important. To be applied to such contexts, our approach would require modifications. Costs would still be at the basis of the decisions made by the robots, however the different relation between the sub-tasks would impact the role of the interfaces. The methods and algorithms presented in this dissertation largely rely on an implicit information exchange

between sub-tasks, which takes place at the interfaces. If the coupling between sub-tasks is loose, such as in the cases in which there is no sequential dependency, the interfaces do not favor such an exchange of information and the decision process of the robots that leads to the definition of the partitioning strategy is likely to be compromised.

We believe that explicit communication is a solution to tackle this issue. The robots could use communication to compensate for the lack of information at the interfaces. The robots would estimate the cost of performing sub-tasks exactly as they do in the approach we propose and communicate this information to the other robots. The information gathered by different robots would therefore spread within the swarm and it would be used to obtain the overall coordination required to perform the sub-tasks efficiently.

Another dimension that remains unexplored is the study of task partitioning in contexts in which tasks and sub-tasks require group execution. In this dissertation we focus on foraging and the tasks consist in transporting objects. Each robot is capable of transporting an object by itself and therefore tasks and sub-tasks are performed individually. Other situations in swarm robotics require the robots to cooperate actively in order to perform a task. For example, foraging for objects that are too heavy for a single robot to carry requires transportation to be performed collectively by a group of robots. Our approach entails that the robots make decisions individually; the partitioning strategy is the result of the combination of individual decisions. Therefore, in cases in which the tasks and sub-tasks require a group of individuals to work together, the decision process must involve all the robots of the group. One option is that one of the robots makes decisions and imposes its choice to the other robots of the group. The alternative would be to resort to a collective decision process, such as the mechanisms proposed in Campo (2011). In addition to the decision process, the way costs are estimated would require a modification as well. The estimation should be done collectively to exploit the information gathered by the different members of the group. Also in this case, the use of explicit communication is a suitable solution to extend our approach.

Annexes

Annex A

ARGoS 1 Versus ARGoS 2

In this annex, we compare the results of the experiments presented in Section 5.4 with the results published in Pini et al. (2011). The contents of Section 5.4, are largely based on our published article. However, the results presented in the section are obtained with a newer version of ARGoS (henceforth ARGoS 2) with respect to the results presented in the published article (obtained with ARGoS 1).

We decided to replicate the experiments for a matter of coherence: all the results of the simulation-based experiments presented in this dissertation are obtained with ARGoS 2. Repeating the experiments gave us the opportunity to collect more data and to study properties of the system that were neglected in Pini et al. (2011). Additionally, we improved the implementation of the controllers of the robots, in particular for what concerns navigation. The results obtained with ARGoS 2, presented in Section 5.4, confirm the ones published in the article.

In Section A.1, we point out the implementation differences between the environments in which the robots perform foraging. In Section A.2, we highlight minor differences in the experiments (i.e., in their parameters) and include the results originally published in Pini et al. (2011).

A.1 Implementation Differences

The main implementation difference between the experiments reported in Section 5.4 and the ones presented in Pini et al. (2011) concerns the environment in which the robots perform foraging.

Figure A.1 depicts the environment utilized in the two versions of the experiments: on the left-hand side the environment utilized in the experiments presented in Section 5.4; on the right-hand side the environment utilized in Pini et al. (2011). A first difference is that the

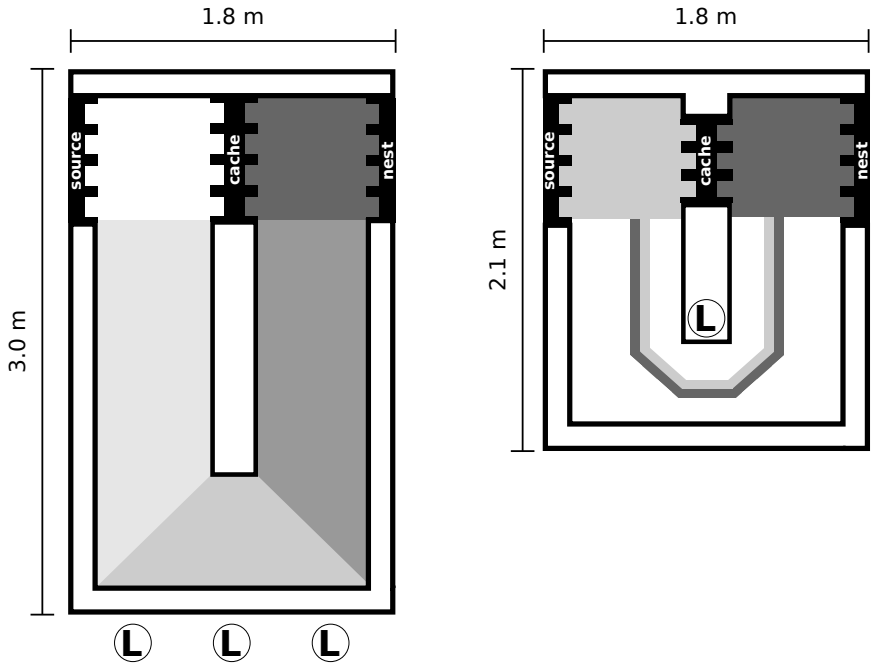

Figure A.1: Comparison of the environments in which the robots perform foraging. Left: environment used in the experiments described in Section 5.4. Right: environment used in the experiments presented in Pini et al. (2011).

floor of the corridor is painted differently. In the environment represented on the right-hand side of Figure A.1, a path marks the floor and it is followed by the robots to navigate through the corridor. In the new version of the experiments, we substitute the path with three areas of different color (see Figure A.1 left). The robots can navigate in the corridor by maintaining a certain heading with respect to the direction of the lights depending on the perceived color of the ground. This solution renders the navigation of the corridor more efficient. In fact, the previous implementation forced the robots to remain on the narrow strip marked by the colored path. The new implementation, on the other hand, allows the robots to use the whole width of the corridor. The result is that, while navigating in the corridor, the robots can move faster and avoid each other more easily.

In addition to the color of the ground, the number and the position of the lights is different across the implementations. In the new implementation of the experiments, there are three lights (instead of one) and they have been positioned outside the perimeter of the environment. This improves the navigability of the corridor, in this case in

Table A.1: Implementation differences between the environment used in the experiments presented in Section 5.4 and the one used in Pini et al. (2011).

Parameter	**Section 5.4**	**Pini et al. (2011)**
Corridor floor	3 colored areas	2 colors path
Lights	3, outside the perimeter	1, inside the perimeter
Cache	4 TAMs per side	3 TAMs per side
Default swarm size	14 robots	10 robots
Environment size	1.8 m by 3.0 m	1.8 m by 2.1 m

its curved portion (bottom part in Figure A.1).

The size of the cache is also different across the implementations. In the new version of the experiments, the cache is composed of 4 TAMs on each side, to match the number of TAMs at the source and the nest. In the previous version of the experiments, the cache was implemented using 3 TAMs on each side. Given that we increased the number of slots available in the cache, we also increased the default swarm size from 10 to 14 robots, to maintain a similar ratio between the number of robots and the number of slots in the cache.

Finally, the size of the environments is different: the previous version consists of a 1.8 m by 2.1 m environment, while the new version is 1.8 m by 3.0 m. Therefore, we increased the length of the corridor. The reason is that, due to the improvements in navigation, the robots navigate through the corridor faster and therefore we had to increase its length so that the cache remains preferable to the corridor for low values of the interfacing time Π. Table A.1 summarizes the differences across the two versions of the experiments.

A.2 Published Experiments and Results

In this section, we focus on the experiments performed with ARGoS 1 and presented in Pini et al. (2011) and we discuss the differences with respect to the corresponding ones described in Section 5.4. The experiments differ in two aspects. The main difference is the location of the robots at the beginning of the experiment. In the experiments described in Pini et al. (2011), the robots are initially positioned in the area containing the nest. An exception are the cases in which the robots use the always-partition algorithm: in this case, half of the swarm is initially positioned on each side of the cache. In the experiments described in Section 5.4, the initialization of the positions of the robots is the same independently of the algorithm utilized: half of

the swarm is positioned on each side of the cache at the beginning of each experiment. In this way, all the algorithms are compared using the same initial conditions and therefore we removed a potential bias.

The second difference between the experiments is in the number of repetitions executed for each experimental setting. In the experiments presented in Section 5.4, we perform 25 repetitions for each condition, while the published results are obtained with 50 repetitions for each condition.

As mentioned in Section 5.4, the experiments are divided into three sets: evaluation of the performance, test of adaptivity, and study of scalability. In the following we report for each set of experiments the main differences in the experimental setup (if any) and the corresponding results that have been published in Pini et al. (2011).

A.2.1 Performance Evaluation

The goal of the experiments presented in this section is to compare the AdHoc algorithm with the two reference algorithms: the never-partition and the always-partition algorithms. We compare the algorithms for different values of the cache interfacing time Π. The values of Π do not match the ones utilized in the experiments presented in Section 5.4.2. Due to different implementations of the environments and the controllers of the robots, the relative benefit of using the cache over the corridor for a given value of Π is different. The values of Π utilized in Pini et al. (2011) are 0, 25, 50, 75, 100, 150, and 200 seconds.

Figure A.2 reports the average number of objects delivered to the nest per robot for different values of Π and for each algorithm. The results agree with the corresponding ones reported in Figure 5.7 (see page 71): for small values of Π the always-partition algorithm performs better than the others, for high values of Π the never-partition algorithm is preferable. The AdHoc algorithm performs well across the spectrum of the values of Π.

Figure A.3 provides a summary of the strategy employed by the robots of the swarm when using the AdHoc algorithm, for the different values of Π. Each bar reports the percentage of times that each action was performed by the robots; the reported values are averages computed across 50 runs. Also in this case, the results agree with the corresponding ones reported in Section 5.4.2 (see Figure 5.8, page 72): the robots select the corridor with a higher frequency for increasing values of Π.

A difference between the results is in the frequency at which the robots abandon using the cache. In the experiments carried out with ARGoS 2, the robots abandon less often. This is a consequence of

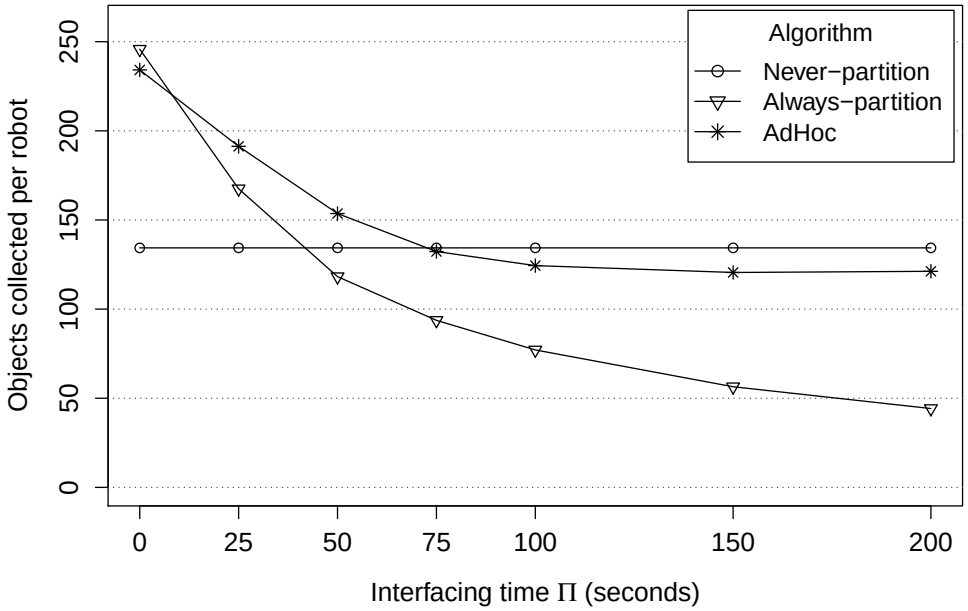

Figure A.2: Average number of objects delivered to the nest per robot at the end of the experiment for different values of the interfacing time Π.

improvements made to the controller of the robots, that render the robots faster in entering the TAMs. The first version of the controller was very sensitive to obstacles and in many cases the robots were taking a long time to enter a TAM, perceived as an obstacle. Consequently, in those cases there was a high probability that the usage of the cache was abandoned by the robot. When we reimplemented the system, we identified the sensitiveness to obstacles and the poor navigability of the corridor as two weaknesses and decided to intervene and remove them.

Figure A.4 reports the time taken by the robots to use the corridor ($t_H + t_S$) and the cache (t_P and t_D) employing the different algorithms, when the interfacing time is 0 s (left), 25 s (center), and 50 s (right). Also in this case, the results published in Pini et al. (2011) agree with the results reported in Section 5.4 (see Figure 5.9, page 73). The plots in Figure 5.9 confirm the improvement of the navigation capabilities of the robots: the time taken to enter the cache TAMs is lower compared to Figure A.4.

To summarize we can conclude that there is agreement between the results presented in Section 5.4.2 and the corresponding ones published in Pini et al. (2011). The changes we made to the implementations of the controllers and of the environment render the results numerically different, but the conclusions that can be drawn remain the same.

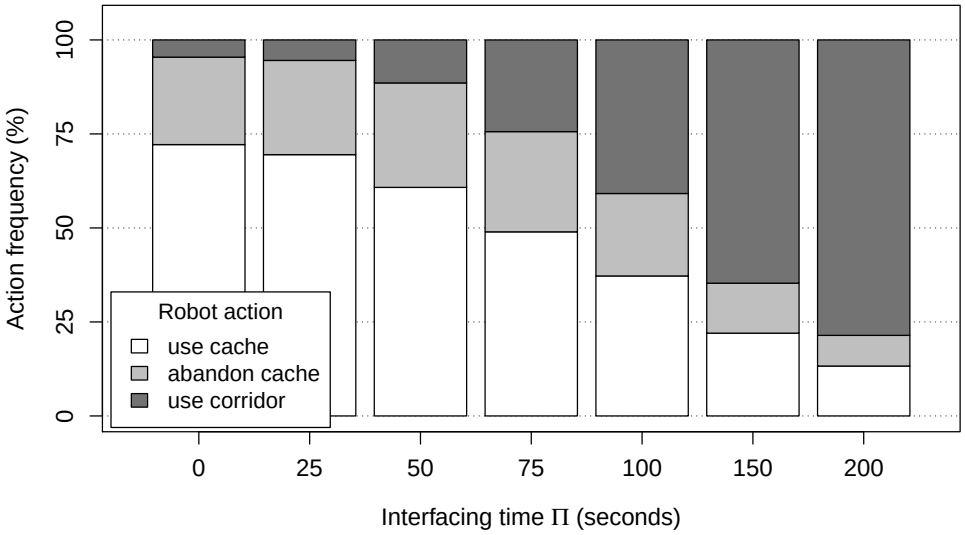

Figure A.3: Actions performed by the robots employing the AdHoc algorithm, for different values of the interfacing time Π. Each bar reports, for a given value of Π, the percentage of times an action was performed. The actions reported are: selection and actual usage of the cache (white), selection of the cache and abandon (light gray), and selection of the corridor (dark gray). The percentage of times the robots chose to employ the cache is the sum of the values reported by the white and the light gray bars. The values reported are the averages, computed across 50 experimental runs.

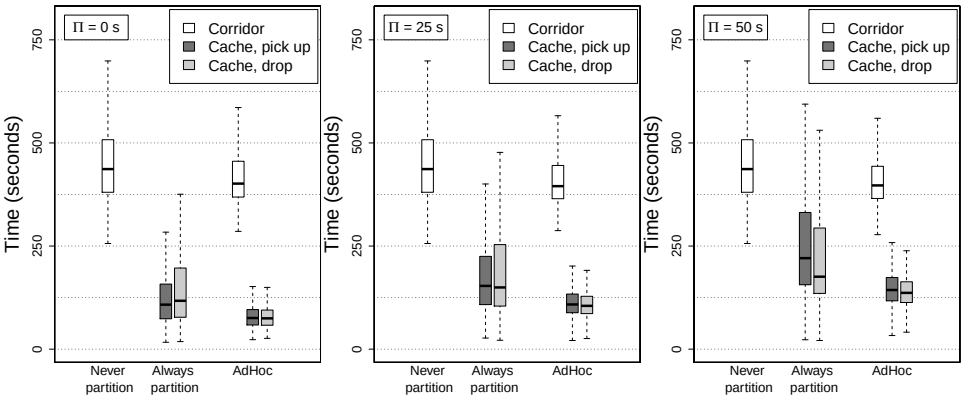

Figure A.4: Time taken to use the corridor and the cache for the three algorithms. The graphs report the data for $\Pi = 0\,\mathrm{s}$ (left), $\Pi = 25\,\mathrm{s}$ (center), and $\Pi = 50\,\mathrm{s}$ (right).

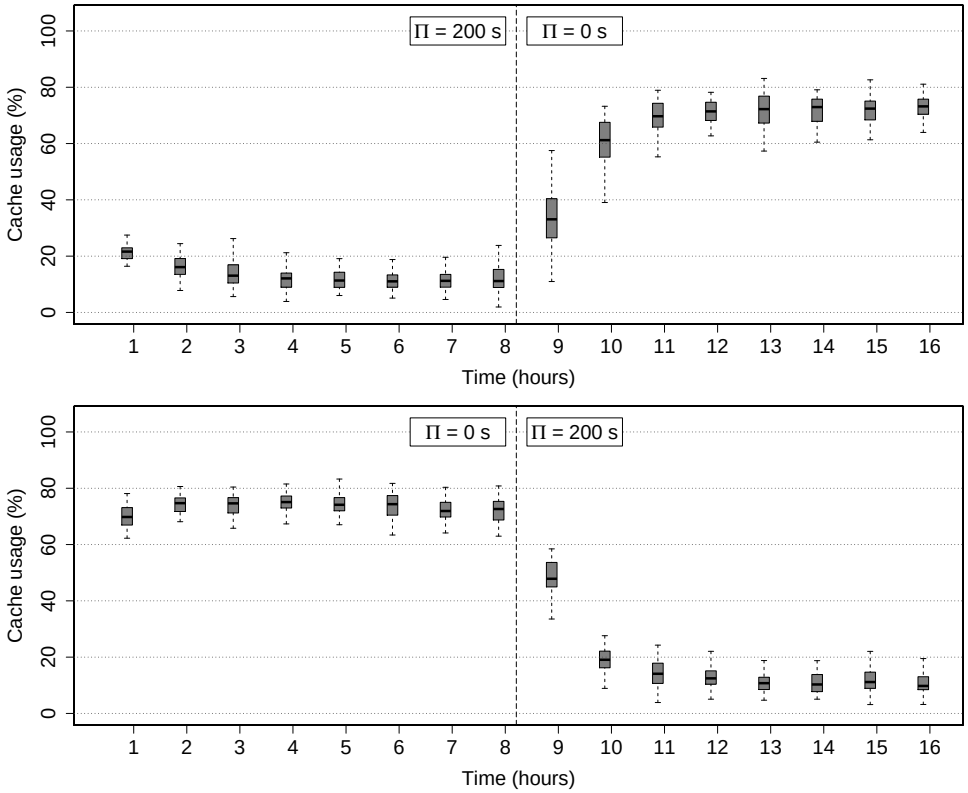

Figure A.5: Percentage of usage of the cache in time when the interfacing time Π changes from 200 s to 0 s (top) and from 0 s to 200 s (bottom). The vertical dashed line marks the instant at which the value of Π changes. Each box reports the percentage of usage of the cache in the hour preceding the time reported on the X axis.

A.2.2 Adaptivity to Changes

The goal of the experiments presented in this section is to test the adaptiveness of the AdHoc algorithm in response to variations of the environmental conditions. Differently from the experiments presented in Section 5.4.2 of this dissertation (page 74), we change the value of the interfacing time only once (instead of three times) during the course of the experiment. We change the interfacing time Π when the experiment reaches its half. We consider two cases: one in which we change Π from 200 s to 0 s and another in which we change it from 0 s to 200 s.

Figure A.5 reports the results of the two experiments. The graph on top plots the data for the case in which the initial value of Π is 200 s; the graph at the bottom for the case in which its initial value

is 0 s. Each graph reports the data collected in the corresponding 50 experimental runs. In the graphs of Figure A.5, the total time frame of the experiment has been divided into windows of 60 minutes. Each box in the plot reports the percentage of usage of the cache in the time window preceding the value indicated on the X axis. Analogously to what is reported in Figure 5.11 (page 76), the strategy employed by the swarm varies in relation to the value of Π. In both cases, at half the experiment time, one of the two strategies is employed more frequently than the other, indicating that the swarm identifies it as the best strategy. The results at the end of both experiments show that the swarm reacts to the change in the environmental conditions. In fact, the strategy of the swarm changes after the value of Π has been modified. The swarm converges to a new strategy more suited to the new value of Π.

A.2.3 Scalability

In this section, we report on the results of experiments testing the scalability of the three algorithms, for two values of the interfacing time: $\Pi = 0\,\text{s}$ and $\Pi = 25\,\text{s}$. Figure A.6 reports the results of the experiments for $\Pi = 0\,\text{s}$ (top) and $\Pi = 25\,\text{s}$ (bottom). Notice that Figure 5.12 (page 78) reports on the Y axis the average total number of objects delivered to the nest by the swarm, while Figure A.6 (published in Pini et al. (2011)) reports the average number of objects delivered per robot. For comparison, we report in Figure A.7 the data plotted in Figure 5.12a (obtained with ARGoS 2), but expressed on a per-robot basis, as in Figure A.6. Among the plots in Figure 5.12, we selected the one that reports data for $\Pi = 5\,\text{s}$, since it can be considered a "low" value of the interfacing time in the new implementation of the system. Analogously, the values $\Pi = 0\,\text{s}$ and $\Pi = 25\,\text{s}$ were low values in the original implementation of the experiments.

A comparison between Figure A.6 and Figure A.7 highlights a different behavior of the always-partition algorithm across the two implementations of the experiments. In the experiments performed with ARGoS 2, the number of objects delivered to the nest per robot monotonically decreases for an increasing swarm size. In the experiments performed with ARGoS 1, the number of objects delivered to the nest per robot initially increases with the swarm size and it has a peak for a swarm of 8 robots. We believe this discrepancy is also a consequence of the different behavior of the robots when entering the TAMs. As mentioned in Section A.2.1, the previous implementation of the controller was very sensitive to the presence of obstacles, and the robots were sometimes taking a long time to enter a TAM. A robot taking a

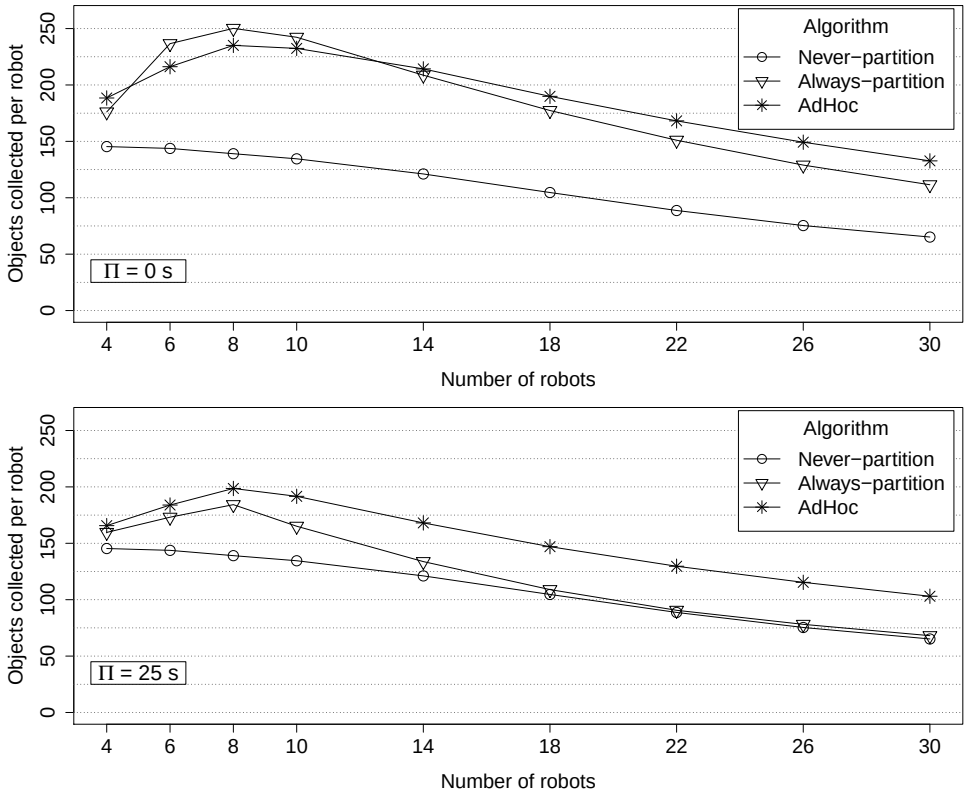

Figure A.6: Results of the scalability experiment for $\Pi = 0\,\mathrm{s}$ (top) and $\Pi = 25\,\mathrm{s}$ (bottom). The plots report, for each algorithm, the number of objects collected by swarms of different size.

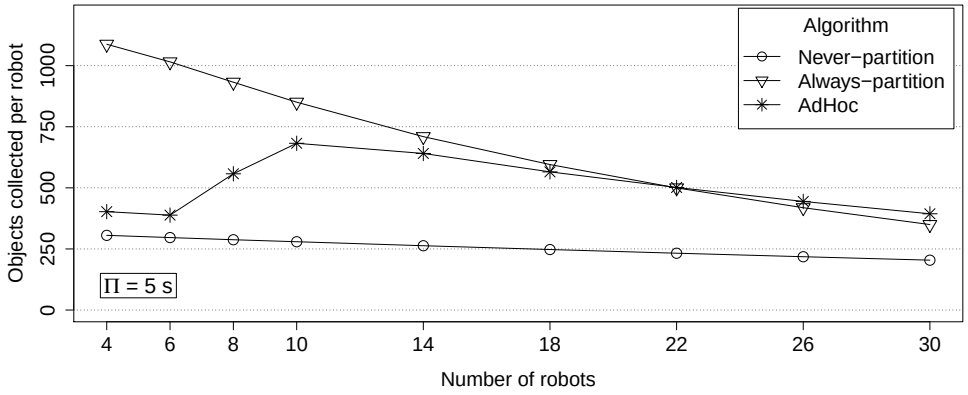

Figure A.7: Results of the scalability experiment performed with ARGoS 2. The plot reports, for different swarm sizes, the average number of objects delivered to the nest per robot. The plot reports the data for $\Pi = 5\,\mathrm{s}$.

long time to enter a cache TAM has a stronger negative impact in small swarms than it has in large ones. Consider, for example, the extreme case in which there are two robots only (one on each side of the cache). If the robot that drops objects in the cache takes a long time to enter a TAM, the robot on the other side is prevented from using the cache since it is likely to be empty. This behavior impacts the performance negatively. Analogous situations are less frequent if the the swarm is composed of more robots. In fact, even if one of the robots takes a long time to enter a TAM, the remaining robots can still utilize the cache and it is less likely that the cache gets full or empty.

The behavior of the never-partition and of the AdHoc algorithms is instead the same across the two implementations. The per-robot performance of the never-partition algorithm monotonically decreases for an increasing swarm size, due to growing physical interference. The individual performance of the robots employing the AdHoc algorithm initially increases with the swarm size, it reaches a peak, and then progressively decreases for an increasing swarm size.

Annex B

Supplementary Material

This thesis includes supplementary material that is available in a CD-ROM, provided as an annex (see final page). The same material is available online.[1] This material can be cited as:

> Pini, G. (2013). Towards autonomous task partitioning in swarm robotics: Experiments with foraging robots, supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2013-001

The CD-ROM contains the following material:

Supplementary material for Chapter 5

- Video: Implementation of the cache using TAMs
- Experiments: AdHoc algorithm, parameters selection
- Experiments: Drop objects abandoning statistics
- Experiments: Tested algorithms, parameters selection
- Experiments: Cache usage in time, single runs

Supplementary material for Chapter 6

- Video: Trajectory of a marXbot
- Experiments: Real-robot experimental runs results
- Video: Real-robot run without task partitioning
- Video: Real-robot run with task partitioning
- Experiments: Real-robot and simulation comparison
- Experiments: Effect of odometry noise and swarm size
- Experiments: Cost-based partitioning algorithm, parameters

[1] `http://iridia.ulb.ac.be/supp/IridiaSupp2013-001`

Bibliography

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. and Franks, N. R. (2001). Teams in animal societies. *Behavioral Ecology*, 12(5):534–540.

Anderson, C. and Ratnieks, F. L. W. (1999). Task partitioning in insect societies ii: Use of queueing delay information in recruitment. *The American Naturalist*, 154(5):536–548.

Arnan, X., Ferrandiz-Rovira, M., Pladevall, C., and 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., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256.

Auer, P. and Ortner, R. (2010). UCB revisited: improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1–2):55–65.

Beni, G. and Wang, J. (1989). Swarm intelligence in cellular robotic systems. In *NATO Advanced Workshop on Robotics and Biological Systems*, volume 102, Tuscany, Italy.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Science of Complexity. Oxford University Press, New York, NY.

Bonabeau, E., Theraulaz, G., and Deneubourg, J.-L. (1996). Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proc. Roy. Soc. London B*, 263(1376):1565–1569.

Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., and Mondada, F. (2010). The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, pages 4187–4193. IEEE Press.

Borenstein, J. (1994). The CLAPPER: A dual-drive mobile robot with internal correction of dead-reckoning errors. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation (ICRA 1994)*, volume 4, pages 3085–3090. IEEE Computer Society Press, Los Alamitos, CA.

Borenstein, J. and Liqiang, F. (1996). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880.

Bovet, D. P. and Cesati, M. (2005). *Understanding the Linux Kernel*. O'Reilly Media.

Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41.

Brutschy, A., Pini, G., Baiboun, N., Decugnière, A., and Birattari, M. (2010). The IRIDIA-TAM: A device for task abstraction for the e-puck robot. Technical Report TR/IRIDIA/2010-015, IRIDIA, ULB.

Brutschy, A., Pini, G., and Decugnière, A. (2012a). Grippable objects for the foot-bot. Technical Report TR/IRIDIA/2012-001, IRIDIA, ULB.

Brutschy, A., Pini, G., Pinciroli, C., Birattari, M., and Dorigo, M. (2012b). Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems*.

Brutschy, A., Tran, N.-L., Baiboun, N., Frison, M., Pini, G., Roli, A., Dorigo, M., and Birattari, M. (2011). Costs and benefits of behavioral specialization. In *Towards Autonomous Robotic Systems - 12th Annual Conference (TAROS 2011)*, volume 6856 of *Lecture Notes in Computer Science*, pages 90–101. Springer, Berlin, Germany.

Brutschy, A., Tran, N.-L., Baiboun, N., Frison, M., Pini, G., Roli, A., Dorigo, M., and Birattari, M. (2012c). Costs and benefits of behavioral specialization. *Robotics and Autonomous Systems*, 60(11):1408–1420.

Campo, A. (2011). *On the Design of Self-Organized Decision Making in Robot Swarms*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

Campo, A. and Dorigo, M. (2007). Efficient multi-foraging in swarm robotics. In Capcarrere, M., Freitas, A. A., Bentley, P. J., Johnson, C. G., and Timmis, J., editors, *Advances in Artificial Life: Proceedings of the 8th European Conference on Artificial Life (ECAL 2005)*, volume 4648 of *Lecture Notes in Artificial Intelligence*, pages 696–705. Springer, Berlin, Germany.

Campos, M., Bonabeu, E., Theraulaz, G., and Deneubourg, J.-L. (2000). Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 8(2):83–95.

Cao, Y. U., Fukunaga, A. S., and Kahng, A. B. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):1–23.

Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. (2007). USARSim: a robot simulator for research and education. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2007)*, pages 1400–1405, Piscataway, NJ. IEEE Press.

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, Learning, and Games*. Cambridge University Press, Cambridge, UK.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms, Second Edition*. MIT Press, Cambridge, MA.

Dahl, T. S., Matarić, M. J., and Sukhatme, G. S. (2009). Multi-robot task allocation through vacancy chain scheduling. *Robotics and Autonomous Systems*, 57:674–687.

Dorigo, M. and Birattari, M. (2007). Swarm intelligence. *Scholarpedia*, 2(9):1462.

Dorigo, M., Birattari, M., O'Grady, R., Gambardella, L. M., Mondada, F., Floreano, D., Nolfi, S., Baaboura, T., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A. L., Decugnière, A., Di Caro, G., Ducatelle, F., Ferrante, E., Martinez Gonzales, J., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, M., Pinciroli, C., Pini, G., Rétornaz, P., Rey, F., Roberts, J., Rochat, F., Sperati, V., Stirling, T., Stranieri, A.,

Stützle, T., Trianni, V., Tuci, E., Turgut, A. E., and Vaussard, F. (2011). Swarmanoid, the movie. In Aha, D. and Jhala, A., editors, *AAAI-11, AI Video Competition*. AAAI Press. Winner of the "AAAI-2011 Best AI Video Award".

Dorigo, M. and Şahin, E. (2004). Guest editorial. Special issue: Swarm robotics. *Autonomous Robots*, 17(2–3):111–113.

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., Di Caro, G., Ducatelle, F., Ferrante, E., Förster, A., Gonzales, J. M., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, 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., and Vaussard, F. (2013). Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*. In press.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. MIT Press, Cambridge, MA.

Drogoul, A. and Ferber, J. (1992). From Tom Thumb to the Dockers: Some experiments with foraging robots. In Meyer, J.-A., Herbert, L. R., and Stewart, W. W., editors, *Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 2*, pages 451–459. MIT Press, Cambridge, MA.

Ducatelle, F., Di Caro, G., Pinciroli, C., Mondada, F., and Gambardella, L. M. (2011). Communication assisted navigation in robotic swarms: Self-organization and cooperation. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 4981–4988. IEEE Computer Society Press, Los Alamitos, CA.

Ennals, R., Sharp, R., and Mycroft, A. (2005). Task partitioning for multi-core network processors. In *Compiler Construction*, volume 3443/2005 of *Lecture Notes in Computer Science*, pages 76–90. Springer, Berlin,Germany.

Feng, L., Borenstein, J., and Everett, H. R. (1994). *"Where am I" Sensors and Methods for Autonomous Mobile Robot Positioning*. University of Michigan Press, Ann Arbor, MI.

Fontan, M. S. and Matarić, M. J. (1996). A study of territoriality: The role of critical mass in adaptive task division. In Maes, P., Matarić,

M. J., Meyer, J.-A., Pollack, J., and Wilson, S., editors, *Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 4*, pages 553–561. MIT Press, Cambridge, MA.

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

Frison, M., Tran, N.-L., Baiboun, N., Brutschy, A., Pini, G., Roli, A., Dorigo, M., and Birattari, M. (2010). Self-organized task partitioning in a swarm of robots. In Dorigo, M., Birattari, M., Di Caro, G., Doursat, R., Engelbrecht, A. P., Floreano, D., Gambardella, L. M., Groß, R. Şahin, E., Stützle, T., and Sayama, H., editors, *Proceedings of the 7th International Conference on Swarm Intelligence (ANTS 2010)*, volume 6234 of *Lecture Notes in Computer Science*, pages 287–298. Springer, Berlin, Germany.

Garnier, S., Gautrais, J., and Theraulaz, G. (2007). The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31.

Goldberg, D. (2001). *Evaluating the Dynamics of Agent-Environment Interaction*. PhD thesis, University of Southern California, Los Angeles, CA.

Goldberg, D. and Matarić, M. J. (2002). Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. In Balch, T. and Parker, L. E., editors, *Robot Teams: From Diversity to Polymorphism*, pages 315–344. A K Peters/CRC Press.

Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J. M. (1989). Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581.

Grabowski, R., Navarro-Serment, L. E., Paredis, C. J. J., and Khosla, P. K. (2000). Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, (8):293–308.

Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., and Magdalena, L. (2009). Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 3111–3116, Piscataway, NJ. IEEE Press.

Gutiérrez, A., Campo, A., Monasterio-Huelin, F., Magdalena, L., and Dorigo, M. (2010). Collective decision-making based on social odometry. *Neural Computing & Applications*, 19(6):807–823.

Handl, J., Knowles, J., and Dorigo, M. (2006). Ant-based clustering and topographic mapping. *Artificial Life*, 12(1):35–61.

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

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

Helbing, D., Molnár, P., Farkas, I. J., and Bolay, K. (2001). Self-organizing pedestrian movement. *Environment and Planning B: Planning and Design*, 28:361–383.

Hicks, R. W. I. and Hall, E. L. (2000). Survey of robot lawn mowers. In *Proceedings of SPIE 4197, Intelligent Robots and Computer Vision XIX: Algorithms, Techniques, and Active Vision*, pages 262–269. SPIE, Bellingham, WA.

Hongo, T., Arakawa, H., Sugimoto, G., Tange, K., and Yamamoto, Y. (1987). An automatic guidance system of a self-controlled vehicle. *IEEE Transactions on Industrial Electronics*, IE-34(1):5–10.

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

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

Jeanne, R. L. (1991). *The Social Biology of Wasps*, chapter 11, pages 389–425. Cornell University Press, Ithaca, NY.

Johansson, R. and Saffiotti, A. (2009). Navigating by stigmergy: A realization on an RFID floor for minimalistic robots. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 245–252. IEEE Press, Piscataway, NJ.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME, ser. D, Journal Of Basic Engineering*, 82(1):35–45.

Kalra, N. and Martinoli, A. (2006). A comparative study of market-based and threshold-based task allocation. In *Distributed Autonomous Robotic Systems 7*, pages 91–102. Springer, Berlin, Germany.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948. IEEE Service Center, Piscataway, NJ.

Koenig, N. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, pages 2149–2154. IEEE Press.

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

Kurazume, R. and Hirose, S. (2000). An experimental study of a cooperative positioning system. *Autonomous Robots*, 1(8):43–52.

Labella, T. H., Dorigo, M., and Deneubourg, J.-L. (2006). Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):4–25.

Lein, A. and Vaughan, R. T. (2008). Adaptive multi-robot bucket brigade foraging. In Bullock, S., Noble, J., Watson, R., and Bedau, M. A., editors, *Artificial Life XI: Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems*, pages 337–342. MIT Press, Cambridge, MA.

Lein, A. and Vaughan, R. T. (2009). Adapting to non-uniform resource distributions in robotic swarm foraging through work-site relocation. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 601–606, Piscataway, NJ. IEEE Press.

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

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

Lumer, E. and Faieta, B. (1994). Diversity and adaptation in populations of clustering ants. In Meyer, J.-A. and Wilson, S. W., editors, *Proceedings of the 3rd International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3*, pages 501–508. MIT Press, Cambridge, MA.

Magnenat, S., Rétornaz, P., Bonani, M., Longchamp, V., and Mondada, F. (2011). ASEBA: A modular architecture for event-based control of complex robots. *IEEE/ASME Transactions on Mechatronics*, 16(2):321–329.

Mathews, N., Christensen, A. L., O'Grady, R., Rétornaz, P., Bonani, M., Mondada, F., and Dorigo, M. (2011). Enhanced directional self-assembly based on active recruitment and guidance. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 4762–4769. IEEE Computer Society Press, Los Alamitos, CA.

Mayet, R., Roberz, J., Schmickl, T., and Crailsheim, K. (2010). Antbots: A feasible visual emulation of pheromone trails for swarm robots. In Dorigo, M., Birattari, M., Di Caro, G., Doursat, R., Engelbrecht, A. P., Floreano, D., Gambardella, L., Groß, R. Şahin, E., Stützle, T., and Sayama, H., editors, *Proceedings of the 7th International Conference on Swarm Intelligence (ANTS 2010)*, volume 6234 of *Lecture Notes in Computer Science*, pages 84–94. Springer, Berlin, Germany.

Michel, O. (2004). Cyberbotics Ltd.—Webots: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65. IPCB-Instituto Politécnico de Castelo Branco.

Mondada, F., Pettinaro, C. G., Guignard, A., Keww, I., Floreano, D., Deneubourg, J.-L., Nolfi, S., Gambardella, L. M., and Dorigo, M. (2004). Swarm-Bot: A new distributed robotic concept. *Autonomous Robots*, 17:193–221.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press/Bradford Books, Cambridge, MA.

Nouyan, S., Groß, R., Bonani, M., Mondada, F., and Dorigo, M. (2009). Teamwork in self-organized robot colonies. *IEEE Transactions on Evolutionary Computation*, 13(4):695–711.

Orians, G. H. and Pearson, N. E. (1979). *On the theory of central place foraging*, pages 155–177. Ohio State University Press, Columbus, OH.

Østergaard, E. H., Sukhatme, G. S., and Matarić, M. J. (2001). Emergent bucket brigading: A simple mechanisms for improving performance in multi-robot constrained-space foraging tasks. In Andre, E., Sen, S., Frasson, C., and Jörg, P. M., editors, *Proceedings of the 5th International Conference on Autonomous Agents*, pages 29–30. ACM Press, New York.

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

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

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Stirling, T., Gutiérrez, A., Gambardella, L. M., and Dorigo, M. (2011). ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 5027–5034. IEEE Computer Society Press, Los Alamitos, CA.

Pini, G., Brutschy, A., Birattari, M., and Dorigo, M. (2009a). Interference reduction through task partitioning in a robotic swarm. In Filipe, J., Andrade-Cetto, J., and Ferrier, J.-L., editors, *Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2009)*, pages 52–59. INSTICC Press, Setùbal, Portugal.

Pini, G., Brutschy, A., Birattari, M., and Dorigo, M. (2009b). Task partitioning in swarms of robots: Reducing performance losses due to interference at shared resources. In Cetto, J. A., Filipe, J., and Ferrier, J.-L., editors, *Informatics in Control, Automation and Robotics*, volume 85 of *Lecture Notes in Electrical Engineering*, pages 217–228. Springer, Berlin, Germany.

Pini, G., Brutschy, A., Francesca, G., Dorigo, M., and Birattari, M. (2012a). Multi-armed bandit formulation of the task partitioning problem in swarm robotics. In Dorigo, M., Birattari, M., Blum, C., Christensen, A. L., Engelbrecht, A. P., Groß, R., and Stützle, T., editors, *Proceedings of the 8th International Conference on Swarm Intelligence (ANTS 2012)*, volume 7461 of *Lecture Notes in Computer Science*, pages 109–120. Springer, Berlin, Germany.

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

Pini, G., Brutschy, A., Pinciroli, C., Dorigo, M., and Birattari, M. (2013a). Autonomous task partitioning in robot foraging: An approach based on cost estimation. *Adaptive Behavior*, 21(2):117–135.

Pini, G., Brutschy, A., Scheidler, A., Dorigo, M., and 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., and Birattari, M. (2013b). Task partitioning in a robot swarm: A study on the effect of communication. *Swarm Intelligence*.

Pini, G. and Tuci, E. (2008). On the design of neuro-controllers for individual and social learning behaviour in autonomous robots: An evolutionary approach. *Connection Science Journal*, 20(2–3):211–230.

Pini, G., Tuci, E., and Dorigo, M. (2007). Evolution of social and individual learning in autonomous robots. In *Proceedings of the 1st Workshop on Social Learning in Embodied Agents (SLEA), CD-ROM*.

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

Rekleitis, I., Dudek, G., and Milios, E. (2001). Multi-robot collaboration for robust exploration. *Annals of Mathematics and Artificial Intelligence*, (31):7–40.

Rooks, B. (2001). Robots reach the home floor. *Industrial robot: An international Journal*, (28):27–28.

Russel, S. J. and Norvig, P. (2009). *Artificial Intelligence: a Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of the SAB 2004 Workshop on Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, Berlin, Germany.

Seeley, T. D. (1995). *The Wisdom of the Hive*. Harvard University Press, Cambridge, MA.

Shell, D. A. and Matarić, M. J. (2006). On foraging strategies for large-scale multi-robot systems. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, pages 2717–2723, Pitscataway, NJ. IEEE Press.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning, an Introduction*. MIT Press, Cambridge, MA.

Vasconcelos, H. L. and Cherrett, J. M. (1996). The effect of wilting on the selection of leaves by the leaf-cutting ant *Atta laevigata*. *Entomologia Experimentalis et Applicata*, 78(2):215–220.

Vaughan, R. T., Stoy, K., Sukhatme, G. S., and Matarić, M. J. (2002). LOST: Localization-space trails for robot teams. *IEEE Transactions on Robotics and Automation*, 18(5):796–812.

Werger, B. B. and Matarić, M. J. (1996). Robotic "food" chains: Externalization of state and program for minimal-agent foraging. In Maes, P., Matarić, M. J., Meyer, J. A., Pollack, J., and Wilson, S. W., editors, *Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 4*, pages 625–634, Cambridge, MA. MIT Press.

Winfield, A. F. T. (2009). Foraging robots. In Meyers, R. A., editor, *Encyclopedia of Complexity and System Science*, pages 3682–3700. Springer, Berlin, Germany.