Université Libre de Bruxelles
Faculté des Sciences Appliquées
CoDE - Ingénierie de l'informatique et de la décision
IRIDIA - Institut de recherches interdisciplinaires
et de développements en intelligence artificielle

Incremental Social Learning in Swarm Intelligence Systems

Marco A. Montes de Oca Roldán

Promoteur:

Prof. Marco Dorigo

Directeur de Recherches du F.R.S.–FNRS
IRIDIA, CoDE, Université Libre de Bruxelles

Co-promoteur:

Dr. habil. Thomas Stützle

Chercheur Qualifié du F.R.S.–FNRS
IRIDIA, CoDE, Université Libre de Bruxelles

Thése présentée à la Faculté des Sciences Appliquées de l'Université Libre de Bruxelles en vue de l'obtention du titre de Docteur en Sciences de l'Ingenieur.

Année Académique 2010–2011

Abstract

A swarm intelligence system is a type of multiagent system with the following distinctive characteristics: (i) it is composed of a large number of agents, (ii) the agents that comprise the system are simple with respect to the complexity of the task the system is required to perform, (iii) its control relies on principles of decentralization and self-organization, and (iv) its constituent agents interact locally with one another and with their environment.

Interactions among agents, either direct or indirect through the environment in which they act, are fundamental for swarm intelligence to exist; however, there is a class of interactions, referred to as *interference*, that actually blocks or hinders the agents' goal-seeking behavior. For example, competition for space may reduce the mobility of robots in a swarm robotics system, or misleading information may spread through the system in a particle swarm optimization algorithm. One of the most visible effects of interference in a swarm intelligence system is the reduction of its efficiency. In other words, interference increases the time required by the system to reach a desired state. Thus, interference is a fundamental problem which negatively affects the viability of the swarm intelligence approach for solving important, practical problems.

We propose a framework called *incremental social learning* (ISL) as a solution to the aforementioned problem. It consists of two elements: (i) a growing population of agents, and (ii) a social learning mechanism. Initially, a system under the control of ISL consists of a small population of agents. These agents interact with one another and with their environment for some time before new agents are added to the system according to a predefined schedule. When a new agent is about to be added, it learns socially from a subset of the agents that have been part of the system for some time, and that, as a consequence, may have gathered useful information. The implementation of the social learning mechanism is application-dependent, but the goal is to transfer knowledge from a set of experienced agents that are already in the environment to the newly added agent. The process continues until one of the following criteria is met: (i) the maximum number of agents is reached, (ii) the assigned task is finished, or (iii) the system performs as desired. Starting with a small number of agents reduces interference because it reduces the number of interactions within the system, and thus, fast progress toward the desired state may be achieved. By learning socially, newly added agents acquire knowledge about their environment without incurring the costs of acquiring that knowledge individually. As a result, ISL can make a swarm intelligence system reach a desired state more rapidly.

We have successfully applied ISL to two very different swarm intelligence systems. We applied ISL to particle swarm optimization algorithms. The results of this study demonstrate that ISL substantially improves the performance of these kinds of algorithms. In fact, two of the resulting algorithms are competitive with state-of-the-art algorithms in the field. The second system to which we applied ISL exploits a collective decision-making mechanism based on an opinion formation model. This mechanism is also one of the original contributions presented in this dissertation. A swarm robotics system under the control of the proposed mechanism allows robots to choose from a set of two actions the action that is fastest to execute. In this case, when only a small proportion of the swarm is able to concurrently execute the alternative actions, ISL substantially improves the system's performance.

Acknowledgments

This dissertation represents the culmination of a process that started in late 2004, when I first contacted Marco Dorigo by email. From the second email on (he did not respond to the first email), I received his full support in practically everything I proposed to him. First, he supported my application for an Alβan scholarship, which was part of a European Union program of high-level scholarships for Latin America. His support was both academic and financial. Academically, he helped me define a project that would allow me to earn a Ph.D degree from the *Université Libre de Bruxelles.* Financially, Marco accepted to fund the remaining 33% of the costs of my initially 3-year long research project (for some reason, an Alβan scholarship did not fully fund the applicant). Eventually, I won the scholarship. While the application for the Alβan scholarship was being processed, we decided to apply for a CONACyT scholarship funded by the Mexican government. I remember being nervous about the fact that he had to sign documents in Spanish, which was a language he did not master. He did sign the documents nonetheless. I also won that scholarship. However, I declined it in order to accept Alβan scholarship No. E05D054889MX. That was the beginning of a journey that has had, and will continue to have, a tremendous impact on my professional and personal lives.

Here I am, five years and nine months after I arrived in Brussels, finally writing the last words of my dissertation. These last words have to be of gratitude because one never does anything truly by oneself. In other words, our actions are the result of past experiences that have been, to various extents, shaped by others (see the rest of this dissertation for a deeper discussion of this subject).

I want to thank Marco Dorigo, who played the role of my main advisor throughout these years. His academic and personal support were essential for the completion of my Ph.D. project. Also thanks to Marco, I was able to enjoy financial support from the ANTS and META-X projects, both of which were *Actions de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. I was also financially supported through the SWARMANOID project funded by the Future and Emerging Technologies programme (IST-FET) of the European Commission, under grant IST-022888, and by the VIRTUAL SWARMANOID project funded by the Fund for Scientific Research F.R.S.-FNRS of Belgium's French Community. I want to thank Thomas Stützle, my main co-advisor. Without Thomas, many of the ideas that are now presented in this dissertation would have been just ephemeral bits of inspiration that would have never seen the light of day. His constant support and encouragement were critical for the completion of my Ph.D. project. Thomas changed roles a number of times during these years. He started as my academic co-advisor. Then, he became my mentor. Now, he is also my friend. Mauro Birattari, my secondary co-advisor (just to give a name to his role), has always been critic of my work, and I thank him for that. Thanks to him, I learned that one should never be content with one's results. His deep knowledge about many fields allows him to take different perspectives from which to look at things. As a result, any work in which he is involved, is high-quality work. I wish I could have learned even more from him. I also want to thank Prof. Bernard Fortz, Dr. Rajan Filomeno Coelho, both from the *Université Libre de Bruxelles*, and Prof. Martin Middendorf from the *Universiät Leipzig*, for their comments and useful critique that helped me improve this dissertation.

My current and former fellow IRIDIAns deserve all my gratitude too. All of them, Christos Ampatzis, Doğan Aydın, Prasanna Balaprakash, Hugues Bersini, Navneet Bhalla, Saifullah bin Hussin, Matteo Borrotti, Manuele Brambilla, Arne Brutschy, Alexandre Campo, Sara Ceschia, Marco Chiarandini, Anders Christensen, Alain Coletta, Muriel Decreton, Antal Decugnière, Giacomo Di Tollo, Jérémie Dubois-Lacoste, Stefan Eppe, Eliseo Ferrante, Gianpiero Francesca, Marco Frison, Matteo Gagliolo, António Gaspar-Cunha, Roderich Groß, Álvaro Gutiérrez-Martín, Takashi Kawakami, Stefanie Kritzinger, Manuel López-Ibáñez, Tom Lenaerts, Renaud Lenne, Tianjun Liao, Max Manfrin, Mattia Manfroni, Amin Mantrach, Bruno Marchal, Xavier Martínez-González, Franco Mascia, Nithin Mathews, Michael Maur, Prospero C. Naval, Jr, Shervin Nouyan, Rehan O'Grady, Sabrina Oliveira, Paola Pellegrini, Christophe Philemotte, Carlo Pinciroli, Giovanni Pini, Carlotta Piscopo, Jessica Rivero Espinosa, Andrea Roli, Fabio Rossi, Ruben Ruiz, Jodelson Sabino, Utku Salihoglu, Francesco Sambo, Francisco C. Santos, Alexander Scheidler, Krzysztof Socha, Paolo Spanevello, Alessandro Stranieri, Cristina Teixeira, Vito Trianni, Elio Tuci, Ali Emre Turgut, Colin Twomey, Thijs Urlings, David Weiss, and Zhi Yuan helped me in different ways throughout these years.

Five groups of people have been especially important for me during my stay in Brussels. The first group, composed of Doğan Aydın, Mauro Birattari, Barbara Di Camillo, Marco Dorigo, Andries Engelbrecht, Eliseo Ferrante, Manuel López-Ibáñez, Tianjun Liao, Nithin Mathews, Michael Maur, Sabrina Oliveira, Paola Pellegrini, Jorge Peña, Carlo Pinciroli, Francesco Sambo, Alexander Scheidler, Paolo Spanevello, Thomas Stützle, Ken Van den Enden, and Zhi Yuan contributed to my professional development through many fruitful collaborations that resulted in a number of publications during my Ph.D. Prasanna Balaprakash, Navneet Bhalla, Manuele Brambilla, Eliseo Ferrante, Max Manfrin, Nithin Mathews, Sabrina Oliveira, Carlo Pinciroli, Francesco Sambo, Francisco C. Santos, and Elio Tuci comprise the second group, which is composed of close friends. I thank them for being an extended family and giving me personal support during the time I lived in Brussels. The third group is composed of César A. Marín, Gisela Zamayoa, Lilia Pérez Romero, and Yumi (a little furry friend). They are my best Mexican friends in Europe. The fourth group is my beloved family. I thank my mom, my brother, and my sister-in-law for their unwavering love and support. The fifth group is a singleton whose only member is Caitlin J. deWilde, who has mostly been in the United States, patiently waiting for me. I thank her for all her support and encouragement, and above all, I thank her for her love during the past two years.

Marco A. Montes de Oca
June 28, 2011
Brussels, Belgium

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The term *swarm intelligence* refers to the group-level intelligence that some groups of animals exhibit in nature (Bonabeau et al., 1999; Dorigo and Birattari, 2007; Garnier et al., 2007a). Famous examples of the swarm intelligence exhibited by some groups of animals are the ability of swarms of bees to choose the best site on which to build their nest (Seeley, 2010) or the ability of ant colonies to find the shortest path between their nest and a food source (Goss et al., 1989). A fundamental characteristic of a group exhibiting swarm intelligence is its ability to solve problems that the group's constituent members cannot solve individually. This fact has made scientists wonder whether it is possible to design problem-solving techniques or systems that use many, yet simple, constituent parts – referred to as *agents*[1]. A first wave of advances in swarm intelligence research led to the development of successful optimization techniques such as ant colony optimization (ACO) (Dorigo et al., 1991a,b; Dorigo, 1992; Dorigo et al., 1996; Dorigo and Di Caro, 1999; Bonabeau et al., 2000; Dorigo and Stützle, 2004; Dorigo, 2007) and particle swarm optimization (PSO) (Kennedy and Eberhart, 1995; Kennedy et al., 2001; Engelbrecht, 2005; Clerc, 2006; Poli et al., 2007; Dorigo et al., 2008). In this first wave of advances, swarm intelligence was also investigated in the context of multi-robot systems (Deneubourg et al., 1990b; Holland and Melhuish, 1999; Dorigo et al., 2004; Beni, 2005).

Most artificial swarm intelligence systems in existence today were inspired by natural swarms. For example, the foraging behavior of ants inspired the design of ACO (Dorigo and Stützle, 2004), and the flocking behavior of birds inspired the design of PSO (Kennedy and Eberhart, 1995). Likewise, in swarm robotics research it is possible to find complete research projects inspired by the way social insects, in particular ants, cooperate to solve problems (see e.g., Dorigo et al. (2004); Kernbach et al. (2008)). Despite the differences among these systems, their constituent agents share a common behavioral trait: they are usually *searching agents*, that is, they are agents that are continuously in search of a target state. What agents search for depends on the purpose of the system. For example, in ACO, the agents that form the swarm (called "colony" in the context of ACO) search for solutions to combinatorial optimization problems. In PSO, agents search for solutions to continuous optimization problems. In swarm robotics, the searching behavior of robots can be more elusive, but in many cases, it involves searching for a desired individual or collective state. For example, in the work of Turgut et al. (2008) or Trianni and Nolfi (2009), robots are continuously searching for a state that makes the swarm cohesive in space (flocking) or time (synchronization), respectively.

Swarm intelligence is the result of agents interacting with each other and with their environment. At the same time, however, sharing information and an environment with other agents produces negative interactions that we refer to as *interference*. This class of interactions blocks or hinders an agent's behavior. As a result of interference, the speed at which a swarm intelligence system reaches a desired state will be reduced. Importantly, interference will tend to increase with the size of the system as a result of the fact that

[1]Throughout this dissertation, we will use the word *agent* to generically refer to an entity, be it an animal or an artifact, such as a robot or a piece of software, capable of autonomous perception and action.

interference is a function of the number of interactions within a system. Thus, interference hinders the scalability of swarm intelligence systems.

Two examples will help us illustrate how interference reduces the performance of swarm intelligence systems. We first consider a PSO algorithm, in which a swarm of agents (called particles) exchange information with one another in order to bias their search toward the best points they find in the search space of a continuous optimization problem. Although cooperation is fundamental for the success of the algorithm, it is also a source of interference, especially during the first iterations of the algorithm. The mutual influence that particles exert on each other makes them move to regions that do not contain the optimal solution to the problem. If the swarm of particles is large, the number of objective function evaluations spent in this initial phase will also be large, and thus, the time needed by the swarm to start making progress toward good solutions will increase. As a second example, we consider a swarm robotics system in which robots have to search for a resource. Since the environment in which they move has finite dimensions, robots have to continuously avoid collisions with each other. If the swarm of robots is large, the space between robots may be such that robots spend most of their time and energy unproductively by avoiding collisions rather than completing their assigned tasks. The overall effect of interference in this example is also to slow down progress toward a desired state.

1.1 Objective

The main objective of the work presented in this dissertation is to reduce the effects of interference in swarm intelligence systems composed of multiple searching agents. Since interference manifests itself as an influence that slows down progress toward a desired state, reducing its effects helps a swarm intelligence system to reach a desired state more rapidly.

To meet the aforementioned objective, in this dissertation we introduce the *incremental social learning* (ISL) framework. This framework consists of two elements: (i) an initially small population of agents that grows over time, and (ii) a social learning process whereby new agents learn from more experienced ones. A small population of agents would reach a certain state more rapidly than a large population because of the reduced interference. However, it is possible that a small swarm cannot reach the desired state. For example, imagine a scenario in which too few robots cannot move a heavy object. We tackle this problem by adding agents to the swarm according to some predefined criterion. An agent that is added to the swarm learns from the agents that have been in the swarm for some time. This element of ISL is attractive because new agents acquire knowledge from more experienced ones without incurring the costs of acquiring that knowledge individually. Thus, ISL allows the new agents to save time that they can use to perform other tasks. After the inclusion of a new agent, the swarm needs to re-adapt to the new conditions; however, the agents that are part of the swarm do not need to start from scratch because some useful work would have already been completed.

1.2 Methodology

We considered two case studies of the application of the incremental social learning framework to swarm intelligence systems:

1. **Swarm intelligence for continuous optimization.** We considered PSO algorithms as a case study to measure the effectiveness of ISL. As a result, three PSO-based optimization algorithms are proposed. Two of these algorithms obtain results comparable with those obtained by other state-of-the-art continuous optimization algorithms. The development and analysis of these algorithms is presented in Chapter 4.

2. **Swarm intelligence for robotics.** As a second case study, we considered a swarm intelligence system in which robots perform a foraging task that involves collective

transport. In this task, robots need to choose one of two available paths to a storage room for transported objects. In this second case study, we first developed a collective decision-making mechanism that allows a swarm of robots to select the shortest path. Then, we instantiated the incremental social learning framework using the aforementioned decision-making mechanism as the searching algorithm used by the swarm. The collective decision-making mechanism and its combination with ISL are presented in Chapter 5.

In both case studies, the application of the incremental social learning framework resulted in a substantial improvement of the underlying system's performance. These successes should be taken as proof of concept. Our experiments are not formal proof that the approach will always produce positive results. However, some requirements that the underlying swarm intelligence system should satisfy in order to expect benefits from the application of ISL are proposed.

1.3 Contributions

In this dissertation, the following three contributions are presented:

1. **Incremental social learning framework**. This original framework aims to tackle interference in swarm intelligence systems. Since such systems are usually composed of a large number of interacting agents, interference can be a major problem because the effects of interference are stronger when a large population of agents is involved. The incremental social learning framework addresses this problem by making a swarm intelligence system start with a small population and by letting new agents learn from more experienced agents.

2. **High-performance PSO algorithms**. A number of high-performance PSO algorithms are proposed in this dissertation. Two of these algorithms are the result of the instantiation of the incremental social learning framework in the context of PSO algorithms. These algorithms are identified by the names IPSOLS and IPSOLS+. They are PSO algorithms with a growing population size in which individual and social learning are simulated through local search and biased initialization, respectively. The third algorithm, which is not based on the incremental social learning framework, is presented in Appendix A. This algorithm, called Frankenstein's PSO, is an integration of algorithmic components that were found to provide good performance in an extensive empirical evaluation of PSO algorithms.

3. **Self-organized collective decision-making mechanism for swarms of robots**. A self-organized collective-decision making mechanism with application to swarm robotics is proposed. Positive feedback and a consensus-building procedure are the key elements of this mechanism that allows a population of robots to select the fastest-to-execute action from a set of alternatives, thus improving the efficiency of the system. We apply the incremental social learning framework to this mechanism in order to make it more efficient in situations where a small fraction of the swarm can concurrently execute the available alternative actions.

1.4 Publications

A number of publications have been produced during the development of the research work presented in this dissertation. Many of these publications have been written in collaboration with colleagues under the supervision of Prof. Marco Dorigo and/or Dr. Thomas Stützle.

The publications associated with this dissertation are listed below. The majority of them deal with the incremental social learning framework and its applications. However, we have also listed publications that laid the ground for the development of the incremental social learning framework.

1.4.1 International Journals

1. Montes de Oca, M. A., Ferrante, E., Scheidler, A., Pinciroli, C., Birattari, M., and Dorigo, M. (2010b). Majority-rule opinion dynamics with differential latency: A mechanism for self-organized collective decision-making. Technical Report TR/IRIDIA/2010-023, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. [**Revision submitted to Swarm Intelligence**]

2. Montes de Oca, M. A., Aydın, D., and Stützle, T. (2011a). An incremental particle swarm for large-scale optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. *Soft Computing*. Forthcoming. DOI: 10.1007/s00500-010-0649-0

3. Montes de Oca, M. A., Stützle, T., Van den Enden, K., and Dorigo, M. (2011b). Incremental social learning in particle swarms. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 41(2):368–384

4. Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2009c). Frankenstein's PSO: A composite particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 13(5):1120–1132

5. Dorigo, M., Montes de Oca, M. A., and Engelbrecht, A. P. (2008). Particle swarm optimization. *Scholarpedia*, 3(11):1486

1.4.2 International Conferences, Workshops and Symposia

1. Liao, T., Montes de Oca, M. A., Aydın, D., Stützle, T., and Dorigo, M. (2011). An incremental ant colony algorithm with local search for continuous optimization. In Krasnogor, N. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*. ACM Press, New York. To appear. Preprint available at `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-005r002.pdf` [**Nominated for the best paper award in the Ant Colony Optimization and Swarm Intelligence track**]

2. Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2010c). Incremental social learning applied to a decentralized decision-making mechanism: Collective learning made faster. In Gupta, I., Hassas, S., and Rolia, J., editors, *Proceedings of the Fourth IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 243–252. IEEE Computer Society Press, Los Alamitos, CA

3. Montes de Oca, M. A., Ferrante, E., Mathews, N., Birattari, M., and Dorigo, M. (2010a). Opinion dynamics for decentralized decision-making in a robot swarm. In Dorigo, M. et al., editors, *LNCS 6234. Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, pages 251–262. Springer, Berlin, Germany [**Nominated for the best paper award**]

4. Yuan, Z., Montes de Oca, M. A., Stützle, T., and Birattari, M. (2010). Modern continuous optimization algorithms for tuning real and integer algorithm parameters. In Dorigo, M. et al., editors, *LNCS 6234. Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, pages 204–215. Springer, Berlin, Germany

5. Montes de Oca, M. A., Ferrante, E., Mathews, N., Birattari, M., and Dorigo, M. (2009a). Optimal collective decision-making through social influence and different action execution times. In Curran, D. and O'Riordan, C., editors, *Proceedings of the*

Workshop on Organisation, Cooperation and Emergence in Social Learning Agents of the European Conference on Artificial Life (ECAL 2009). No formal proceedings published

6. Montes de Oca, M. A., Van den Enden, K., and Stützle, T. (2008). Incremental particle swarm-guided local search for continuous optimization. In Blesa, M. J. et al., editors, *LNCS 5296. Proceedings of the International Workshop on Hybrid Metaheuristics (HM 2008)*, pages 72–86. Springer, Berlin, Germany

7. Montes de Oca, M. A. and Stützle, T. (2008b). Towards incremental social learning in optimization and multiagent systems. In Rand, W. et al., editors, *Workshop on Evolutionary Computation and Multiagent Systems Simulation of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 1939–1944. ACM Press, New York

8. Montes de Oca, M. A. and Stützle, T. (2008a). Convergence behavior of the fully informed particle swarm optimization algorithm. In Keijzer, M. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 71–78. ACM Press, New York [**Nominated for the best paper award in the Ant Colony Optimization, Swarm Intelligence, and Artificial Immune Systems track**]

9. Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2006a). A comparison of particle swarm optimization algorithms based on run-length distributions. In Dorigo, M. et al., editors, *LNCS 4150. Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2006)*, pages 1–12. Springer, Berlin, Germany

10. Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2006b). On the performance analysis of particle swarm optimisers. *AISB Quarterly*, 124:6–7

1.5 Structure

This dissertation consists of six chapters and one appendix. In Chapter 2, we provide relevant background information for the rest of the dissertation. In Chapter 3, we present the rationale and the algorithmic structure of the incremental social learning framework as well as a discussion of related work. The application of ISL to PSO algorithms is described in Chapter 4. First, we present a simple incremental PSO algorithm, called IPSO. Then, we present two high-performing PSO algorithms, called IPSOLS and IPSOLS+, that are derived from it. In Chapter 5, we present the application of ISL to a swarm robotics system. First, we describe the actual swarm robotics system the framework is applied to. Then, we describe the application of ISL to this system. Finally, in Chapter 6, we present the main conclusions of the research work documented in this dissertation. Appendix A is devoted to the description of *Frankenstein's PSO* algorithm, which is the result of an extensive experimentation with several PSO algorithms. Results of those experiments inspired in part some features of the ISL framework.

Chapter 2

Background

In this chapter, we present some of the basic concepts of swarm intelligence and social learning, which are central to our work. In Section 2.1, we present the concept of swarm intelligence, and describe its principles and mechanisms. We also describe the most successful artificial swarm intelligence systems together with the natural phenomena that inspired their development. In Section 2.2, we present the concepts of individual and social learning, and describe the main mechanisms involved in social learning.

2.1 Swarm Intelligence

In nature, different kinds of animals tend to congregate in large numbers. For instance, European starlings can gather in thousands to form flocks (Carere et al., 2009), atlantic silversides form schools of hundreds of individuals (Partridge, 1982), and ants make colonies that range in size from a few dozen to millions of ants (Hölldobler and Wilson, 1990). When animals form these *swarms*, they are often able to solve problems that no single member could if it acted alone. From an external observer's point of view, it may appear as if the swarm possessed a certain level of intelligence that is well superior to that of any of its constituent members. This collective-level intelligence is called *swarm intelligence.*

The size and behavior of swarms have fascinated humans since antiquity. At times, swarms inspire fear. For example, it is written in the Bible that swarms of locusts plagued Egypt (Exodus:10.3–6). At other times, swarms inspire respect. An old Mesoamerican legend tells the story of how ants helped the gods feed all humans with cultivated maize (Nuttall, 1930). Both extremes of feelings, fear and awe, have motivated researchers to wonder whether it is possible to control swarms. On the one hand, controlling swarms would allow us to alleviate the effects of plagues, like those of locusts or termites (Buhl et al., 2006). On the other hand, controlling swarms would allow us to devise techniques that can be used to control man-made artifacts such as robots or software agents (Bonabeau et al., 1999). However, before we are able to control swarms, we need to understand their governing principles.

2.1.1 Principles and Mechanisms

Even though the characteristics of swarm-forming animals vary substantially, swarms exhibit behaviors that are in fact very similar. This similarity has pointed toward the existence of a set of general principles responsible for the emergence of swarm-level organization and intelligence (Buhl et al., 2006). The existence of these principles makes the design of artificial swarm intelligence systems possible. Thus, as a discipline, swarm intelligence has a twofold objective. First, it aims to understand the fundamental principles that are the responsible for the collective-level intelligence sometimes exhibited by large groups of animals. Second, it aims to define engineering methodologies for the design and construction of large groups of man-made entities that collectively solve practical problems (Dorigo and Birattari, 2007).

Researchers have made progress in the study of swarm intelligence and a set of principles and mechanisms that make it possible have been identified. The principles and mechanisms that we will describe have been found to operate in many animal societies, but especially in social insects groups (Bonabeau et al., 1999; Garnier et al., 2007a; Beekman et al., 2008).

Decentralization

The behavior exhibited by a swarm is not dictated by any central authority. The unfortunate name given to the reproductive member of an ant colony or a bee hive (i.e., a "queen") gives the impression that the organization observed at the collective level is the result of a hierarchical command structure. However, it is now well known that such a structure does not exist (Bonabeau, 1998; Garnier et al., 2007a). In a swarm, no single agent supervises the actions of, or issues orders to, other members of the swarm. The perception and interaction scope of a swarm member is local. Thus, the swarm's organization is the result of local interactions, both among the swarm members and between the swarm members and the environment.

Stigmergy

The theory of stigmergy (from the Greek roots *stigma*, which means mark, sign, or puncture, and *ergon*, which means action, labor, or work) was proposed by Grassé (1959) in the context of task coordination and nest construction regulation in colonies of termites. Grassé defined stigmergy as "the stimulation of the workers by the very performances they have achieved" (Grassé, 1959) p. 79. In other words, stigmergy refers to the coordination process that arises when an agent performs an action as a consequence of stimuli that are the result of another agent's – or possibly the same agent's – actions.

Stigmergy is key to explain how termites and other social insects are able to build structures and produce collective-level patterns that are orders of magnitude larger than a single individual, all without a central authority or global blueprint. For example, the construction of soil arches in termite nests starts when a termite fortuitously places a soil pellet on top of other pellets. This bigger soil structure stimulates termites to keep placing pellets on top. A self-reinforcing process then follows: the larger the structure, the stronger the attraction termites feel toward that structure to deposit soil pellets. Eventually an arch is built if two pillars happen to be at an appropriate distance. Another prominent example of how stigmergy enables swarm intelligence to occur is the ability of ants of some species to find the shortest path between their nest and a food source. While moving, ants deposit on the ground chemical substances called *pheromones*. These pheromones modify the environment and trigger a change in the behavior of ants. In particular, ants become attracted to areas of the environment marked with pheromones. This pheromone laying and following behavior induces a positive feedback process whereby areas with high concentration of pheromones become more and more attractive as more ants follow them (Pasteels et al., 1987; Goss et al., 1989; Deneubourg et al., 1990a). As a result, if there are several paths to the same food source, the colony is more likely to select the shortest path because ants will traverse it faster, and thus, it will have a higher pheromone concentration than longer ones.

Self-Organization

The theory of self-organization has found applications in such diverse fields as economics, urbanism, physics, chemistry, and biology (Haken, 2008). For example, it has been used to explain chemical reactions, such as the Belousov-Zhabotinsky reaction (Zhabotinsky, 2007), and the organization of cities (Portugali, 2000). In biology, it has been used to explain the external patterns on the skin or on the protective shells of some animals (Camazine et al., 2001), the movement of vertebrates in crowds (Couzin and Krause, 2003), and, most relevant for our discussion here, the behavior of social insects swarms (Bonabeau et al., 1997).

Self-organization is a term with different meanings in different contexts (Gershenson, 2007). In this dissertation, we adopt Camazine et al.'s definition:

Definition *Self-organization is a process in which [a] pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern.* (Camazine et al., 2001) p. 8.

With this definition, some forms of swarm intelligence can be considered to be the result of self-organization. For example, the ability of ant colonies to find the shortest path between their nest and a food source can be seen as a self-organization process. First, a pheromone trail that connects an ant colony nest to a food source is the pattern at the global level cited in Camazine et al.'s definition. Such a trail is the result of several ants reinforcing it every time they traverse it, that is, it is the result of multiple interactions among the system's components (the ants). Stigmergy is in this case the interaction mechanism. The pheromone-laying and pheromone-following behavior exhibited by ants serves as an interaction rule, which is triggered only when an ant perceives pheromones in its vicinity. Finally, the behavioral rules followed by ants do not make any reference to pheromone trails and do not encode desired goals such as finding shortest paths. The shortest path between an ant colony's nest and a food source is an emergent pattern.

Self-organization is itself the result of the interaction of several processes and elements. According to Camazine et al. (2001) and Moussaid et al. (2009), these processes and elements are the following:

1. Multiple direct or indirect interactions among the system's components. By definition, a self-organizing system is composed of a number of components whose behavior depends on the state of their immediate environment or on the information they possess. In such a setting, the system's components mutually influence each other because the behavior of one of them may affect the environment of, or the information perceived by, other components. If the system's components are able to communicate directly with each other, it is also possible to influence the behavior of these components via direct communication.

2. Presence of fluctuations. The components of a self-organizing system may be subject to external perturbations or may behave nondeterministically. As a result, there may be fluctuations in the system's state. For example, in the absence of pheromone trails, an ant chooses a walking direction at random, or an ant colony may suffer the sudden loss of several members due to the presence of predators or inclement weather conditions.

3. Positive feedback. Fluctuations, random or not, are often reinforced in self-organizing systems. The way termites construct pillars with soil pellets or the reinforcement of pheromone trails by ants are examples of positive feedback processes. Positive feedback is responsible for the appearance of new structures (e.g., pillars or pheromone trails) that in turn modify the behavior of the system.

4. Negative feedback. The self-reinforcing process brought about by positive feedback loops must be limited. It is impossible, for example, that the concentration of pheromones in an ant trail grows to infinity. In self-organizing systems this task is performed by a so-called negative feedback process. Negative feedback encompasses all limiting environmental factors and a system's internal regulation processes. In the

ant trails example, negative feedback includes pheromone evaporation, food depletion and satiation.

5. Bifurcations and multiple stable states. Self-organizing systems often show abrupt changes in their behavior without an abrupt change in the value of a control parameter. For example, the density of insects is often a parameter that affects the behavior of a swarm. Below a certain threshold, no swarm behavior is observed, whereas above it, a swarm behavior suddenly appears (Buhl et al., 2006). A self-organizing system will reach a stable state which depends on the initial conditions. Since self-organization is often triggered by random fluctuations, the stable state of a system may be just one of several available states.

Currently, there is a growing interest in developing methodologies for the design and control of self-organizing systems (see, for example, Gershenson (2007); Di Marzo Serugendo et al. (2004); Bruecker et al. (2005, 2006, 2007)). The knowledge gained in the process will certainly affect our ability to design and control swarm intelligence systems.

Other Mechanisms

Self-organization can account for many swarm intelligence behaviors, but they may also be the result of other mechanisms, either alone or in combination with a self-organizing process (Camazine et al., 2001; Johnson, 2009). Some of these mechanisms are leadership, blueprints, recipes, templates, or threshold-based responses (Bonabeau, 1998; Camazine et al., 2001; Garnier et al., 2007a). Leadership may play a role when some individuals are more experienced than others or simply when there are better informed individuals. This mechanism, as we will discuss in Chapter 3, is important in the framework proposed in this dissertation. Leadership plays an important role in large groups of moving animals as suggested by recent studies (Couzin et al., 2005). Blueprints are usually associated with the process of constructing a structure. They are representations of the desired structure; however, they do not specify how such a structure should be built. There is an ongoing debate as to whether blueprints are actually used by building animals; however, it is definitely possible to imagine man-made swarm intelligence systems in which agents use such a mechanism. Recipes are step-by-step directions to carry out a task. The execution of a recipe often ignores feedback from the execution process. This aspect of recipes is fundamental in order to distinguish them from stigmergic task execution, in which the execution of an action modifies the environment providing feedback to the acting animal or agent. A template is a kind of “preexisting pattern” in the environment that elicits a specific response from the members of a swarm, normally to actually build over them. For example, termites build a chamber around the body of the queen which produces a pheromone gradient that serves as a template (Bonabeau et al., 1998). Finally, in a threshold-based mechanism, an action is performed as a response to the strength of a stimulus. Threshold-based models have been used in the context of social insects to explain division of labor (Theraulaz et al., 1998), the mechanism whereby insects split responsibilities, as well as to explain collective phenomena in humans (Granovetter, 1978).

2.1.2 Artificial Swarm Intelligence Systems

The design and construction of artificial swarm intelligence systems have been heavily inspired by the behavior of natural swarms. The first efforts toward the development of artificial swarm intelligence systems began in the 1990s with pioneering works in robotics, data mining, and optimization. In fact, these domains are still the application areas of most artificial swarm intelligence systems (Dorigo and Birattari, 2007).

In the remainder of this section, we describe some of the most successful swarm intelligence systems devised to date.

Ant Colony Optimization

The ants' pheromone trail laying and trail following behavior described in Section 2.1.1 inspired the development of ant colony optimization (ACO) (Dorigo et al., 1991a,b; Dorigo, 1992; Dorigo et al., 1996; Dorigo and Di Caro, 1999; Bonabeau et al., 2000; Dorigo and Stützle, 2004; Dorigo, 2007). Some aspects of the real behavior of ants that allows them to find shortest paths in nature are simulated in ACO algorithms in order to tackle optimization problems. In nature, real ants form pheromone trails; in ACO, artificial ants construct candidate solutions to the problem instance under consideration. Solution construction is a stochastic process biased by artificial pheromone trails and possibly by available heuristic information based on the input data of the instance being solved. Pheromones are simulated as numerical information associated with appropriately defined solution components. A positive feedback process implemented by iterative modifications of the artificial pheromone trails is key for all ACO algorithms. In ACO algorithms, pheromone trails can be thought of as a function of the ants' search experience. The goal of positive feedback is to bias the colony towards the most promising solutions.

The ACO metaheuristic (Dorigo and Di Caro, 1999; Dorigo et al., 1999) is an algorithmic framework that allows the implementation of the aforementioned ideas for the approximate solution of optimization problems. Such a framework needs to be instantiated into an algorithm in order to tackle a specific problem. The framework is flexible enough to accommodate specialized problem-solving techniques.

ACO is commonly used to solve combinatorial optimization problems. A formal definition of a combinatorial optimization problem is given next.

Definition *A combinatorial optimization problem is modeled by the tuple (S, f, Ω), where:*

- *S is the set of candidate solutions defined over a finite set of discrete decision variables $\mathbb{X}$. S is referred to as the search space of the problem being tackled;*
- *$f : S \to \mathbb{R}$ is an objective function to be minimized;*[1]
- *Ω is a (possibly empty) set of constraints among the decision variables.*

A decision variable $X_i \in \mathbb{X}$, with $i = 1, \ldots, n$, is said to be instantiated when a value v_i^j that belongs to its domain $D_i = \left\{v_i^1, \ldots, v_i^{|D_i|}\right\}$ is assigned to it. A solution $s \in S$ is called feasible *if each decision variable has been instantiated satisfying all constraints in the set Ω. Solving the optimization problem requires finding a solution s^* such that $f(s^*) \leq f(s)\ \forall s \in S$, while satisfying all constraints in Ω.*

Three high-level procedures compose ACO (see Algorithm 1):

- **ConstructSolutions**. This procedure implements the artificial ants' incremental construction of candidate solutions.

  In ACO, an instantiated decision variable $X_i \leftarrow v_i^j$ is called a solution component $c_{ij} \in C$, where C denotes the set of solution components. A pheromone trail value τ_{ij} is associated with each component $c_{ij} \in C$.

  A solution construction starts from an initially empty partial solution s^p. At each construction step, it is extended by appending to it a feasible solution component from the set of feasible neighbors $N(s^p) \subseteq C$ that satisfies the constraints in Ω. The choice of a solution component is guided by a stochastic decision policy, which is biased by both the pheromone trail and the heuristic values associated with c_{ij}. The exact rules for the probabilistic choice of solution components vary across different ACO variants. The rule proposed in the Ant System algorithm (Dorigo et al., 1996) is the best known rule:

[1]Note that minimizing the value of an objective function f is the same as maximizing the value of $-f$; hence, every optimization problem can be described as a minimization problem.

Algorithm 1 Basic structure of an ant colony optimization algorithm

```
repeat
  ConstructSolutions
  DaemonActions /* Optional */
  UpdatePheromones
until Stopping criterion is satisfied
```

$$p_{c_{ij}|s^p} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum\limits_{c_{il} \in N(s^p)} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}\,, \tag{2.1}$$

where τ_{ij} and η_{ij} are, respectively, the pheromone value and the heuristic value associated with the component c_{ij}. The parameters $\alpha > 0$ and $\beta > 0$ determine the relative importance of pheromone versus heuristic information.

- **DeamonActions**. This procedure, although optional, is important when state-of-the-art results are sought (Dorigo and Stützle, 2004). It allows for the execution of problem-specific operations, such as the use of local search procedures, or of centralized actions that cannot be performed by artificial ants. It is usually executed before the update of pheromone values so that ants bias their search toward high quality solutions.

- **UpdatePheromones**. This procedure updates the pheromone trail values associated with the solution components in the set C. The modification of the pheromone trail values is composed of two stages: (i) *pheromone evaporation*, which decreases the pheromone values of all components by a constant factor ρ (called *evaporation rate*) in order to avoid premature convergence, and (ii) *pheromone deposit*, which increases the pheromone trail values associated with components of a set of promising solutions S_{upd}. The general form of the pheromone update rule is as follows:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \sum_{s \in S_{upd} | c_{ij} \in s} F(s)\,, \tag{2.2}$$

  where $\rho \in (0, 1]$ is the evaporation rate, and $F : S \to \mathbb{R}^+$ is a function such that $f(s) < f(s') \Rightarrow F(s) \geq F(s'),\ \forall\ s \neq s' \in S$. $F(\cdot)$ is called the fitness function. Different definitions for the set S_{upd} exist. Two common choices are $S_{upd} = s_{bsf}$, and $S_{upd} = s_{ib}$, where s_{bsf} is the best-so-far solution, that is, the best solution found since the start of the algorithm, and s_{ib} is the best solution of the current iteration. The specific implementation of the pheromone update mechanism differs across ACO variants (Dorigo et al., 1991a,b, 1996; Dorigo and Gambardella, 1997; Gambardella and Dorigo, 1996; Stützle and Hoos, 2000).

Many ACO algorithms have been proposed. Some of them aim to solve specific problems, and others have a more general purpose. In Table 2.1, we list some of the most representative ACO algorithms proposed to date.

Particle Swarm Optimization

Particle swarm optimization (PSO) (Kennedy and Eberhart, 1995; Eberhart and Kennedy, 1995; Kennedy et al., 2001; Engelbrecht, 2005; Clerc, 2006; Poli et al., 2007; Dorigo et al., 2008) is a population-based stochastic optimization technique primarily used to tackle continuous optimization problems. A continuous optimization problem is defined as follows:

Table 2.1: Representative ACO works.

ACO algorithm	*Main references*
Ant System (AS)	(Dorigo et al., 1991b; Dorigo, 1992; Dorigo et al., 1996)
Elitist AS	(Dorigo et al., 1991b; Dorigo, 1992; Dorigo et al., 1996)
Ant-Q	(Gambardella and Dorigo, 1995)
Ant Colony System (ACS)	(Dorigo and Gambardella, 1997; Gambardella and Dorigo, 1996)
$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS)	(Stützle and Hoos, 1996, 1997, 2000)
Rank-based AS	(Bullnheimer et al., 1999)
ANTS	(Maniezzo, 1998, 1999)
Best-worst AS	(Cordón et al., 2002, 2000)
Population-based ACO	(Guntsch and Middendorf, 2002)
Beam-ACO	(Blum, 2004, 2005)

Definition *Given a set $\Theta \subseteq \mathbb{R}^n$ and an objective function $f : \Theta \to \mathbb{R}$, the continuous optimization problem consists in finding at least one member of the set*

$$\Theta^* = \arg\min_{\boldsymbol{\theta}\in\Theta} f(\boldsymbol{\theta}) = \{\boldsymbol{\theta}^* \in \Theta \colon f(\boldsymbol{\theta}^*) \leq f(\boldsymbol{\theta}), \quad \forall \boldsymbol{\theta} \in \Theta\} \,.$$

The set Θ is referred to as the feasible solution space or as the search space of function f. If $\Theta = \mathbb{R}^n$, then the problem is called an unconstrained continuous optimization problem. Otherwise, the problem is called a constrained continuous optimization problem.

PSO has roots in computer graphics, social psychology, and natural swarm intelligence. Within the computer graphics field, the first antecedents of PSO can be traced back to the work of Reeves (1983), who proposed particle systems to model objects that are dynamic and cannot be easily represented by polygons or surfaces. Examples of such objects are fire, smoke, water and clouds. In these systems, particles are independent of each other and their movements are governed by a set of rules. A few years later, Reynolds (1987) used a particle system to simulate the collective behavior of a flock of birds. In a similar kind of simulation, Heppner and Grenander (1990) included a roost that was attractive to the simulated birds. Reynolds's and Heppner and Grenander's models inspired the set of rules that were later used in the original PSO algorithm (Kennedy and Eberhart, 1995). According to Kennedy (2006), social psychology research, in particular the theory of social impact (Latané, 1981; Nowak et al., 1990), was another source of inspiration in the development of the first particle swarm optimization algorithm (see Chapter 3 for more information).

PSO is a *direct search* method, which means that it works only with ordinal relations between objective function values and does not use the actual values to model, directly or indirectly, higher order properties of the objective function. In a PSO algorithm, simple agents, called *particles*, move in the solution space of an n-dimensional objective function f (see definition above). There are three vectors associated with a particle i at time step t: its position vector $\mathbf{x}_i^t$, which represents a candidate solution, its velocity vector $\mathbf{v}_i^t$, representing the particle's search direction, and its *personal best* vector $\mathbf{pb}_i^t$, which denotes the particle's best position attained by particle i since the beginning of the algorithm's execution.

The rules that determine the particles' movement are the core of any PSO algorithm. These rules determine from which other particles a certain particle i should get information, and how that information should be exploited. The set of particles from which particle i may obtain information is referred to as particle i's *neighborhood* and is denoted by $\mathcal{N}_i$. However, particle i's *informers*, denoted by $\mathcal{I}_i$ with $\mathcal{I}_i \subseteq \mathcal{N}_i$, are the particles from which

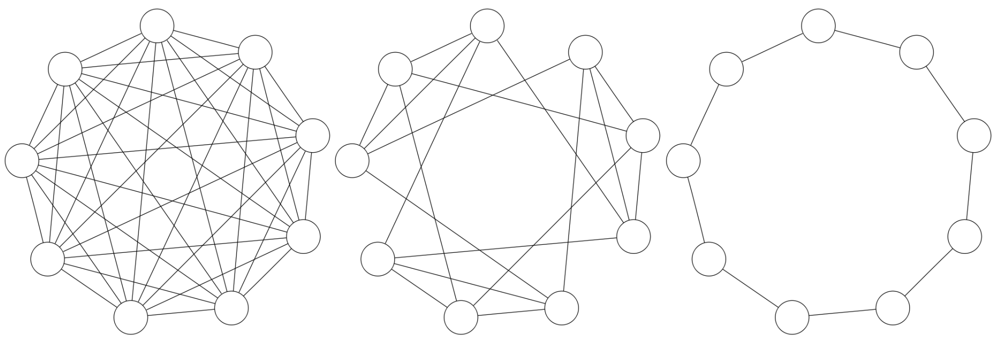


Figure 2.1: Example population topologies. The leftmost picture depicts a fully connected topology, that is, $\mathcal{N}_i$ is composed of all the particles in the swarm (self-links are not drawn for simplicity). The picture in the center depicts a so-called von Neumann topology, in which $|\mathcal{N}_i| = 4 \; \forall i$. The rightmost picture depicts a ring topology in which each particle is neighbor to two other particles.

it actually obtains information. The sets $\mathcal{N}_i$ can be visualized as a graph called *population topology* (see Figure 2.1). The *model of influence* defines the mechanism to form $\mathcal{I}_i$ from $\mathcal{N}_i$. Finally, a particle's velocity-update rule determines how to compute the particle's next position using information from its informers.

In the standard PSO algorithm (Bratton and Kennedy, 2007), for example, the aforementioned factors are instantiated as follows: (i) fully-connected graphs or rings (respectively known as *gbest* and *lbest* models in PSO parlance) as population topologies, (ii) a *best-of-neighborhood* model of influence such that only the best particle in the neighborhood and the particle itself are taken as informers, and (iii) an update rule for the jth component of the ith particle's velocity and position vectors given by

$$v_{i,j}^{t+1} = wv_{i,j}^{t} + \varphi_1 U_1 \left(pb_{i,j}^{t} - x_{i,j}^{t}\right) + \varphi_2 U_2 \left(lb_{i,j}^{t} - x_{i,j}^{t}\right) , \tag{2.3}$$

and

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1} , \tag{2.4}$$

where w is a parameter called *inertia weight* (Shi and Eberhart, 1998a), φ_1 and φ_2 are parameters called *acceleration coefficients*, U_1 and U_2 are uniformly distributed pseudorandom numbers in the range $[0, 1)$ that are generated for each particle for each coordinate at each iteration. A particle's velocity in each coordinate j is usually constrained within the range $[-v_{\max}, v_{\max}]$. Finally, the vector $\boldsymbol{lb}_i^t$ is the best solution in particle i's neighborhood $\mathcal{N}_i$, that is:

$$\boldsymbol{lb}_i^t = \arg\min_{j \in \mathcal{N}_i} f(\boldsymbol{pb}_j^t) . \tag{2.5}$$

The basic structure of a PSO algorithm is shown in Algorithm 2. In the procedure *InitializeSwarm*, a certain number of particles are created and placed uniformly at random in the problem's search space. Each particle's velocity is initialized to zero or a small random value (Dorigo et al., 2008). In this procedure, the population topology is also initialized. In the procedure *EvaluateSwarm*, each particle's position is evaluated using the problem's objective function. If a particle finds a position that is better than its personal best solution, it updates its memory. Otherwise, it remains unchanged. In the procedure *UpdatePositions*, all particles are moved using Eqs. 2.3 and 2.4. The procedures *EvaluateSwarm* and *UpdatePositions* are executed iteratively until the stopping criterion is satisfied.

Different settings for the population topology, the model of influence, or the velocity-update rule give rise to different PSO algorithms. Two-dimensional lattices, small-world networks or random graphs are among the possible choices for replacing the standard fully-connected or ring graphs as population topologies (Kennedy, 1999; Kennedy and

Algorithm 2 Basic structure of a particle swarm optimization algorithm

```
InitializeSwarm
repeat
  EvaluateSwarm
  UpdatePositions
until Stopping criterion is satisfied
```

Table 2.2: Representative PSO works

Investigated Aspect	*Main references*
Acceleration Coefficients	Kennedy (1997); Ratnaweera et al. (2004); Chatterjee et al. (2007); Chaturvedi et al. (2009)
Inertia Weight	Shi and Eberhart (1998a,b, 1999, 2001); Eberhart and Shi (2001); Zheng et al. (2003a,b); Chatterjee and Siarry (2006)
Model of Influence	Mendes et al. (2004); Jordan et al. (2008); Montes de Oca and Stützle (2008a)
Population Size	van den Bergh and Engelbrecht (2001); Lanzarini et al. (2008); Coelho and de Oliveira (2008); Chen and Zhao (2009)
Population Topology	Kennedy (1999); Suganthan (1999); Janson and Middendorf (2003, 2005); Mohais et al. (2005); Kennedy and Mendes (2006)
Theoretical Aspects	Ozcan and Mohan (1999); Clerc and Kennedy (2002); Trelea (2003); Kadirkamanathan et al. (2006); Poli (2007, 2009); Fernández Martínez and García Gonzalo (2009); Ghosh et al. (2011)
Velocity-Update Rule	Kennedy (2003); Blackwell and Branke (2006); Mendes and Kennedy (2007); dos Santos Coelho (2008)

Mendes, 2002). Likewise, alternatives to the *best-of-neighborhood* model of influence can be implemented. The most salient example is the *fully-informed* model, in which a particle is informed by all of its neighbors (Mendes et al., 2004; Mendes, 2004). In Table 2.2 we list a number of works in which one or more of the three aforementioned factors are investigated.

Swarm Robotics

Robotics has been pivotal in the development of the swarm intelligence field. In fact, it was in a robotics paper that the term swarm intelligence was first used (Beni and Wang, 1993; Beni, 2005). Swarm intelligence applied to the multi-robot domain is called *swarm robotics* (Dorigo and Şahin, 2004; Şahin, 2005; Bayindir and Şahin, 2007). It is sometimes defined as "the study of how [a] large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment." (Şahin, 2005) (p. 12). This definition is very similar to that of the engineering branch of the swarm intelligence field (Dorigo and Birattari, 2007). The particularity of swarm robotics is the embodiment of robots. In one of the first works in the field, Deneubourg et al. (1990b) used the term "ant-like" to describe the robots they used in one of the first experiments in the history of the swarm robotics field. At the same time, Deneubourg et al. reinforced the link of the field with one of its major sources of inspiration: social insects. Deneubourg et al. also showed that swarm robotics could be used as a scientific tool to test hypotheses about the mechanisms involved in swarm organization in animals— cf. Webb (2000). For this and other reasons, swarm robotics research, unlike ACO and PSO research, does not focus solely on applications.

In swarm robotics, some mechanisms involved in robot control and the benchmark tasks robots solve, have been inspired by studies of real swarm-forming animals. For example, Deneubourg et al. (1990b), Holland and Melhuish (1999), Wilson et al. (2004), and Melhuish et al. (2006) studied swarms of robots performing spatial sorting inspired by the brood sorting behavior of ants; Theraulaz and Bonabeau (1995) and Grushin and Reggia (2008) studied structure building mechanisms inspired by wasps and other social insects; Kube and Bonabeau (2000) and Groß and Dorigo (2008) reproduced with robots the cooperative transport abilities of ants; and Mondada et al. (2004) and O'Grady et al. (2010b) draw inspiration from social insect assemblages (Anderson et al., 2002) to devise control algorithms that allow swarms of robots to perform collective tasks.

Research in swarm robotics is not only focused on tasks that can be solved collectively by robots. There are also practical problems that need to be tackled in a swarm robotics system. For example, robots that are part of a swarm may need to know when one of their peers stops working properly, or they may need to know how many robots compose the swarm. Some of these problems have been tackled using nature-inspired as well as purely engineered approaches. For instance, Christensen et al. (2009) proposed a distributed mechanism for robot fault detection within a swarm that was inspired by models of firefly synchronization. Using a similar approach, Brambilla et al. (2009) built on the work of Holland et al. (1999) to design a mechanism that allows individual robots to reliably estimate the size of the group that they belong to. Energy supply within a swarm is another practical problem that needs to be dealt with. Batteries have a limited capacity, thus, robots have a limited lifetime. If the robots lifetime is short, a swarm of robots is of little practical use. To tackle this problem, some researchers, for example Witkowski (2007), Melhuish and Kubo (2007), and Schloler and Ngo (2008) have proposed energy sharing mechanisms inspired by trophallaxis, that is, the direct exchange of food between animals (Hölldobler and Wilson, 1990). By sharing charge with one another, some robots can continuously operate while other robots get their batteries recharged.

One application area for which swarm robotics is particularly appealing is the construction of two- and three-dimensional structures (Stewart and Russell, 2006; Werfel and Nagpal, 2008; Mellinger et al., 2010). In this application area, most of the basic collective behaviors inspired by animals can be integrated into a single complex task. For example, robots need to aggregate, find construction materials, sort them, transport them from one place to another (most likely, cooperatively), and finally, coordinate their actions in order to actually build the desired structure.

Other Swarm Intelligence Systems

ACO, PSO, and swarm robotics have undoubtedly been the most popular swarm intelligence systems to date. However, other systems exist and deserve being mentioned.

A family of swarm intelligence systems is used to perform data clustering. The goal of any clustering algorithm is to partition a set of data or objects into clusters (groups, subsets, classes) so that elements belonging to the same cluster are as similar as possible and elements that belong to different clusters are as dissimilar as possible (Höppner et al., 1999). Some of these swarm intelligence systems for data clustering focus on optimization, and thus, use ACO, or PSO to tackle the problem (Martens et al., 2011). Other systems, however, are inspired by the brood sorting behavior of some ant species. These systems are called ant-based clustering algorithms (Lumer and Faieta, 1994; Handl et al., 2005; Handl and Meyer, 2007).

Ant-based clustering algorithms are related to experiments in swarm robotics. Deneubourg et al. (1990b) made robots execute the following rules: pick up an object if it is relatively isolated, and put down an object if there are other objects around. As a result, the robots created "heaps" of objects in the environment. Lumer and Faieta (1994) implemented in software a similar system in which agents move over a toroidal square grid on which there are objects representing data items. Agents pick up an object with high probability if it is not surrounded by other similar objects. By the same token, agents put down objects on any free location surrounded by similar objects to the one they are carrying. As a result, groups of similar data items are created. In other words, the algorithm performs data clustering. A number of improvements of the basic technique have followed (see the work of Handl and Meyer (2007) for one of the latest surveys of the topic).

A family of swarm intelligence algorithms, inspired by the behavior of bees, is attracting the attention of researchers in the field (see the work of Karaboga and Akay (2009) for a recent review). One of the algorithms that belong to this category is called Bee Colony Optimization (BCO) (Teodorović, 2009). This algorithm is typically used to tackle combinatorial optimization problems. BCO consists of two procedures that are executed iteratively. In the first procedure, artificial bees build partial candidate solutions. In the second procedure, the artificial bees "meet" in order to recruit other bees to search in the area in proximity to the best found partial solutions. These two procedures roughly mimic the behavior of scout bees looking for rich food sources and of the waggle dance of bees, which is aimed at recruiting other bees from the nest. Another bee-inspired algorithm, the Artificial Bee Colony (ABC) algorithm (Karaboga and Basturk, 2007), is used for tackling continuous optimization problems. In ABC, the position of the bees represent candidate solutions to the problem. The algorithm works through the interaction of three kinds of artificial bees. Bees can be play three roles. They can be "employed", "onlookers", or "scouts." An employed bee exploits a promising region. In other words, the bee carries out a sort of local search. Onlooker bees search around promising regions based on their quality. Onlooker bees can compare the quality of different regions in the search space, thus they perform a more global search than employed bees. Finally, scout bees perform random search, which enables them to discover new promising regions in the search space.

2.2 Social Learning

Social and individual learning are terms that are often used vaguely, meaning different things in different contexts. For the purpose of this dissertation, it is therefore important to clearly define the meaning of these two concepts and their relationship.

Individual (or asocial) learning is the process whereby an agent benefits from experience to become better adapted to its environment (Rescorla, 1988). The exact meaning of "experience" and "adaptation" depends on the context in which the term "learning" is used. In any case, learning implies a change in an agent's behavior from the moment in which it interacts with its environment, or gains "experience", and the moment in which its level of "adaptation" to its environment is measured or observed. In Chapters 4 and 5, we will explicitly define these terms in the context of the two case studies presented in this dissertation.

From a machine learning perspective, learning is finding an association between inputs and some output. Inputs can have many forms, from abstract data, to actual information gathered through electronic sensors. An agent's output can be, for example, actions that change the agent's environment, or an abstract concept, such as a category identifier. The association between inputs and output changes during the lifetime of the learning agent. This association represents the agent's "experience" discussed in the previous paragraph. The purpose of associating inputs with outputs is to maximize some performance measure. A better score using a performance measure means that the agent is "better adapted" to its environment. There are roughly three categories of learning problems (Birattari, 2009): supervised, reinforcement, and unsupervised. In supervised learning (Aha et al., 1991), a *supervisor* provides examples of the desired input-output associations. In this case, a learning agent tries to minimize the differences between its own responses and the desired ones. Reinforcement learning (Kaebling et al., 1996) is based on rewards given to a learning agent when it performs actions that lead to a certain environment state. In this case, a learning agent tries to maximize the collected rewards. Unsupervised learning (Jain et al.,

1999) does not require any examples or rewards. In this case, a learning agent tries to identify input patterns that trigger similar outputs or responses.

There are more definitions of social learning than of individual learning. Fortunately, Heyes (1994) provides a definition onto which one can map many working definitions existing in the literature:

Definition *The term 'social learning' refers to learning that is influenced by observation of, or interaction with, another animal (typically a conspecific) or its products [...]. The complementary set is commonly known as 'individual learning'.* (Heyes, 1994) p. 207.

Heyes's definition is general enough to encompass the definitions of Biederman et al. (1993) who refer to social learning as learning from the observation of others' behavior, and Caldwell and Millen (2009) who use the term social learning as learning from the interaction with others. Other authors prefer to use Heyes's full definition (Brown and Laland, 2003; Caldwell and Millen, 2009; Rendell et al., 2010b,a, 2011).

Social learning in animals has been studied since the 19th century (Galef Jr., 1990). In humans, social learning started to be seriously studied around the 1970s with the work of Bandura (1977) and other psychologists. Similarly to other theories of behavior, social learning in humans and animals has been studied from a mechanistic as well as from a functional point of view. Ethologists and psychologists take a mechanistic perspective in order to determine the mechanisms and strategies that animals use to learn from others. Biologists and scientists from other disciplines, including economics, study social learning from a functional perspective in order to answer the question of why and under which circumstances social learning is useful.

2.2.1 Social Learning Mechanisms and Strategies

Social learning mechanisms (how an agent may learn from others) and strategies (when and from whom should an agent learn socially) are the subject matter of the mechanistic approach to the study of social learning. In the following paragraphs, we will briefly define some of the most commonly studied social learning mechanisms and strategies.

Mechanisms

Imitation, emulation, enhancement, conditioning, facilitation and mimicking are social learning mechanisms. They are not learning phenomena themselves, but they may lead to learning (Heyes et al., 2000). Imitation and emulation involve copying. When an observer imitates, it copies the actions of a demonstrator with the goal of reproducing the actions' effects; when an observer emulates, it uses its own actions to reproduce the results produced by a demonstrator's actions (Heyes, 1994; Caldwell and Millen, 2009; Cakmak et al., 2010). Imitation has been traditionally assumed to be the main mechanism through which animals learn socially (Galef Jr., 1990). However, imitation is a relatively complex process that implies that the copying animal is able to take the perspective of the demonstrating animal. Thus, to explain social learning in animals that are considered to have limited cognitive abilities, such as insects, simpler mechanisms have been sought. One such mechanism is called social enhancement (Franz and Matthews, 2010). Some authors distinguish between two forms of social enhancement: stimulus enhancement and local enhancement. Stimulus enhancement occurs when an agent calls the attention of another one to a particular object, increasing the likelihood that the observer interacts with that object (or with objects with similar physical features) in the future, regardless of the objects' location (Heyes, 1994; Bonnie and de Waal, 2007; Franz and Matthews, 2010). Local enhancement occurs when an agent is attracted to the location where a certain behavior was observed (Galef Jr., 1990; Heyes, 1994; Franz and Matthews, 2010). Social enhancement makes some features of the environment more salient than others. As a result, the observer may save time and effort exploring the environment in order to find interesting objects or locations. Social enhancement imposes lower cognitive capabilities

Table 2.3: Examples of "When" and "From whom" components of a social learning strategy.

When	From whom
When established behavior is unproductive	From majority
When asocial learning is costly	From successful individuals
When uncertain	From good social learners
When dissatisfied	From related individuals (kin)
When environment is stable	From familiar individuals
	From older individuals

on animals than imitation or emulation do. Conditioning in a social context means that an animal learns an association between two stimuli as a result of observing the reaction of a demonstrator to a stimulus (Heyes, 1994). Social facilitation occurs when an animal manifests a behavior more (or less) strongly in the presence of another passive animal of the same species (Zajonc, 1965; Guérin, 1993; Heyes et al., 2000). Social facilitation is considered a social learning mechanism because the influence of another animal may increase or decrease the responsiveness of the observer to its environment, and thus, may change the observer's learning ability. Mimicking is similar to imitation in that the observer copies the actions of a demonstrator. However, when mimicking, the observer is not trying to get the same results as the demonstrator; it simply performs the actions without regard to the actions' goals (Tomasello, 2004). Mimicking could be seen as a socially mediated action exploration mechanism.

Strategies

Functional studies of social learning (see Section 2.2.2) suggest that agents should not learn socially all the time. Instead, these studies conclude that agents should selectively choose between individual and social learning depending on the characteristics of their environment. The strategy used by an agent to decide when and from whom to learn is called a *social learning strategy* (Laland, 2004; Galef Jr., 2009).

Social learning strategies have been studied mostly theoretically within a functional framework to determine which ones are more likely to offer advantages under predefined circumstances (Laland, 2004). Examples of social learning strategies can be built from the components listed in Table 2.3, which lists some plausible "when" and "from whom" components of a social learning strategy. This list was proposed by Laland (2004) and later adapted by Galef Jr. (2009).

In experiments with animals, some scientists have reported what probably is the execution of certain social learning strategies. For example, a copy-when-uncertain social strategy could explain the behavior of Norway rats in an experiment designed by Galef Jr. (1996) in which Norway rats had to choose between two completely novel foods. In such an uncertain situation, the rats preferred the foods that had been consumed by other rats (detected through breath odor) instead of trying any of them with equal probability, which would have been the case if they had been learning individually.

The study of social learning strategies is still in its infancy, but some important efforts are being made in order to discover strategies robust to different environmental conditions. For example, Rendell et al. (2010a) organized a computer-based tournament aimed at discovering effective social learning strategies under a wide range of environmental conditions. In total, 104 strategies were submitted and the final outcome of the tournament has given researchers useful insight into what makes a social learning strategy successful. The strategy that won the tournament favored social learning almost all the time. The reason, Rendell et al. conclude, is that since agents frequently demonstrated the highest-payoff behavior, social learners could observe and copy only promising behaviors. In effect, through the demonstration of good behaviors, agents were filtering out mediocre behaviors that could not be spread through social learning.

2.2.2 Functional Value of Social Learning

The functional approach to the study of social learning aims at understanding the conditions under which social learning evolved and what the adaptive value of social learning is. This approach is of interest to scientists of many disciplines. For example, biologists wonder how variable can an environment be so that social learning evolves (Wakano et al., 2004). Sociologists consider that social learning is at the root of culture but since social learning is a product of evolution, they wonder how cultural and genetic evolution interact (Cavalli-Sforza and Feldman, 1983; Boyd and Richerson, 1985; Flinn, 1997). Economists wonder what is the effect of social learning in the decisions economic agents make and its consequences for the population as a whole (Ellison and Fudenberg, 1995; Chamley, 2004). Computer scientists and engineers are interested in exploiting social learning in the design and use of software and robots (Thomaz, 2006; Nehaniv and Dautenhahn, 2007; Cakmak et al., 2010). We belong to this last class of researchers. As it will be discussed in more detail in Chapter 3, the work presented in this dissertation takes a functional approach toward the application of social learning ideas.

The adaptive value of social learning has been studied mainly through mathematical and computational models. Almost all models assume that social learning is a convenient way to acquire adaptive behavior because it allows the social learning agent to save time and energy that it would otherwise spend learning individually (Laland, 2004). There are also other advantages associated with social learning, such as reducing the risk of exposure to predators or lowering the chances of getting poisoned as a result of trying unknown foods (Galef Jr., 2009). Consequently, it would be reasonable to assume that a population composed of social learning agents would have a higher average fitness than a population composed of only individual learning agents. As it turns out, this reasoning is flawed as shown by Rogers (1988). He demonstrated that social learning agents have an advantage only when individual learning agents are present. This insight motivates research on social learning strategies as we saw above.

A family of social learning models is aimed at investigating the degree to which an environment can change so that social learning is useful (Bergman and Feldman, 1995; Wakano et al., 2004; Laland and Kendal, 2003; Galef Jr., 2009). These models study the relative advantage that reliance on social and individual learning as well as genetically encoded behavior offers to an agent in the presence of a changing environment. As a result of years of theoretical work, it is now well established that when the environment does not change, or when it changes too frequently, a genetically encoded behavior prevails. In the first case, it is assumed that there is a cost associated to learning. Thus, a genetically encoded behavior provides everything an agent needs at a lower cost. In the second case, there is no possibility of learning and thus, again for economic reasons, a genetically encoded behavior prevails. At high rates of change that still allow for some predictability of the environment, individual learning lets an agent have up-to-date information whereas social learning can potentially be harmful since outdated information can pass from one agent to another. At intermediate rates of change social learning flourishes more than individual learning because it is a cheaper way of obtaining adaptive information. Note that social learning models and their implications are subject to change because their predictions have been subjected to limited empirical tests (Laland and Kendal, 2003). As recently shown by (Rendell et al., 2010a), a population of agents might still rely on social learning even in a frequently changing environment simply because demonstrators will tend to adapt their own behavior to the new circumstances and thus, they can still pass useful information to others.

Other models have been devised in order to study the spread of behavior through social learning (Laland and Kendal, 2003; Cavalli-Sforza and Feldman, 1981). The goal of these models is to find a "signature" of social learning in the curves that represent the proportion of individuals in a population adopting a particular behavior. Unfortunately, these models do not consider simple explanations that could account for the adoption patterns observed (Laland and Kendal, 2003). Finally, there are models aimed at understanding whether culture (the cumulative effect of social learning) and natural evolution interact (Feldman and Laland, 1996; Laland and Kendal, 2003). The basic assumption here is that an animal's genotype may determine what it learns, and that learned behavior affects, in turn, the selection pressure on that genotype.

Chapter 3

Incremental Social Learning

In this chapter, we present the incremental social learning (ISL) framework. First, we describe the problem of interference in multiagent systems. Then, we explain how interference is addressed by the ISL framework and present the framework's algorithmic structure. We finish with a brief discussion of related work. Work specifically related to each instantiation of the ISL framework in our case studies is discussed in Chapters 4 and 5.

3.1 Interference

There are different kinds of interactions among agents in multiagent systems. Depending on the effect of such interactions, they can be labeled as "positive", "negative", or "neutral" interactions (Gershenson, 2007). Positive interactions facilitate the accomplishment of an assigned task. For example, in a collective transport task, robots form teams in order to transport objects that are too difficult for a single robot to move (Kube and Bonabeau, 2000; Tuci et al., 2006). Negative interactions, also called *interference*[1] (Matarić, 1997), *friction* (Gershenson, 2007), or *repulsive and competitive interactions* (Helbing and Vicsek, 1999), are those that block or hinder the functioning of the system's constituent agents. As a result, interference decreases the performance of a multiagent system. For instance, in an ant-based clustering algorithm (see Section 2.1.2) agents can undo the actions of other agents, which increases the time needed by the algorithm to find a satisfactory final clustering. A neutral interaction does not affect the system's dynamics in such a way that it benefits or harms progress toward the completion of an assigned task. Deciding whether an interaction is positive, negative, or neutral depends on the time scale used to measure the interaction's effects. For example, an interaction that involves two robots performing a collision avoidance behavior can be labeled as a negative interaction in the short term because time is spent unproductively. However, if the time horizon of the task the robots are performing is significantly longer than the time frame of a collision avoidance maneuver, then the overall effect of such an interaction may be negligible. In this case, such interaction can be labeled as neutral.

Interference is one of the main challenges to overcome during the design and operation of systems composed of many agents (Gershenson, 2007). For example, Kennedy and Eberhart, the designers of the first PSO algorithm, pondered different candidate particle interaction rules before proposing the rules that we now know (see Eqs. 2.3 and 2.4). Their ultimate goal was to design rules that promoted positive interactions between particles. In the final design, particles cooperate, that is, they engage in positive interactions, by exchanging information with one another about the best solution to an optimization problem that each particle finds during its lifetime. At the same time, however, such an exchange of information can "distract" particles and make them search in regions of a problem's search space that seem promising but that in fact do not contain the optimal solution

[1]In this dissertation, we use the term interference to refer to the set of negative interactions that occur within multiagent systems, including swarm intelligence systems.

that the particles are looking for. The net effect of such interactions is that particles may spend objective function evaluations unproductively. This effect intensifies as the size of the particle swarm increases.

Directly measuring interference is difficult. First, one can determine whether the effects of an interaction, or a set of interactions, are beneficial or not only after the task has been performed. Second, as we mentioned before, an interaction may be positive, negative or neutral, depending on the time scale used to measure its effect. In this dissertation, we advocate for qualifying interactions based on their effects in the long term. We do so because it is only at a time scale similar to the time a system needs to perform a task that labeling interactions is relevant for practical purposes. Third, the nature of the interactions themselves poses a challenge. In some systems, agents interact directly on a one-to-one or one-to-some basis, such as in PSO algorithms. In other systems, such as ACO algorithms, agents interact indirectly through the environment and there may be extended periods of time between the moment an agent acts and the moment another agent is affected by those actions. With these restrictions, interference can only be measured indirectly through observation of the system's performance. Despite these difficulties, two measures can be used to indirectly gauge interference: (i) the time needed by the system to reach a desired or target state, or (ii) the amount of work performed in a certain amount of time. If one compares two systems, we expect the system with higher interference to make progress toward a desired state more slowly than the system with lower interference. As a result, if one let two systems run for the same amount of time, the system with larger interference would perform less work than the system with lower interference.

There are two properties of systems composed of many agents that are in direct relation with interference:

1. Interference increases with the number of agents in the system. This effect is the result of the increased number of interactions within the system. The larger the number of agents that comprise the system, the higher the probability of a negative interaction occurring.

2. Interference tends to decrease over time. At one extreme of the spectrum, one can find a system in which interactions between agents are completely random or not purposeful. In such a case, it is expected that agents cannot coordinate and thus, cannot perform useful work. Thus, we expect interference to remain at a constant level over time. At the other extreme of the spectrum, one finds well-behaved systems consisting of a number of agents whose interaction rules are designed in order to make agents coordinate with each other. Initially, we expect a high-level of interference because agents would not have enough knowledge about their current environment. However, over time, the behavioral rules of these agents would exploit any gained knowledge in order to make progress toward the completion of the assigned task. Thus, we expect that in cases like these, interference decreases over time, because the other alternatives would be a random behavior or a pathological system in which interference increases.

By making use of these two properties, it is possible to control, to a certain extent, the levels of interference in a multiagent system. The incremental social learning framework, which will be described next, is based on this observation.

3.2 The Incremental Social Learning Framework

Our goal with the incremental social learning (ISL) framework is to reduce the effects of interference in swarm intelligence systems. ISL is a framework because it offers a conceptual structure that does not prescribe a specific implementation of the ideas on which it relies. Each instantiation of the framework will benefit from knowledge about the specific application domain, and therefore, specific properties of the framework should be analyzed in an application-dependent context.

Algorithm 3 Incremental social learning framework

```
Input: Agent addition criteria, stopping criteria
 1: /* Initialization */
 2: $t \leftarrow 0$
 3: Initialize environment $\mathbf{E}^t$
 4: Initialize population of agents $\mathbf{X}^t$
 5:
 6: /* Main loop */
 7: while Stopping criteria not met do
 8:   if Agent addition criteria is not met then
 9:     default($\mathbf{X}^t, \mathbf{E}^t$) /* Default system */
10:   else
11:     Create new agent $a_{new}$
12:     slearn($a_{new}, \mathbf{X}^t$) /* Social learning */
13:     $\mathbf{X}^{t+1} \leftarrow \mathbf{X}^t \cup \{a_{new}\}$
14:   end if
15:   $\mathbf{E}^{t+1} \leftarrow$ update($\mathbf{E}^t$) /* Update environment */
16:   $t \leftarrow t + 1$
17: end while
```

The ISL framework consists of two elements that manipulate and exploit the two properties mentioned in Section 3.1. The first element of the framework directly affects the interference levels within a system by manipulating the number of interactions among the system's constituent agents. Such a control is achieved by varying the number of agents in the system. The strategy for controlling the size of the agent population exploits the second property, that is, that interference tends to decrease over time. The system starts with a small population that grows at a rate determined by agent addition criteria specified by the user. Two phenomena with opposite effects occur while the system is under the control of the ISL framework. On the one hand, interference increases as a result of adding new agents to the swarm (first property described in Section 3.1). On the other hand, interference decreases because the system operates while the population grows (second property described in Section 3.1).

The second element of the framework is social learning. This element is present before a new agent freely interacts with its peers. Social learning is used so that the new agent does not produce extra interference due to its lack of knowledge about the environment. Leadership, a swarm intelligence mechanism (see Chapter 2), is present in the framework in the process of selecting a subset of agents from which the new agent learns. The best strategy to select such a set depends on the specific application. However, even in the case in which a random agent is chosen as a "model" to learn from, knowledge transfer occurs because the selected agent will have more experience than the new agent that is about to be added. As stated in Chapter 2, we take a functional approach to the use of social learning concepts. We do not pay attention to the mechanisms used by the agents to learn from each other. Instead, we are interested in the effects that social learning has on the agents and on the system.

The two elements that compose ISL are executed iteratively as shown in Algorithm 3.

In a typical implementation of the ISL framework, an initial population of agents is created and initialized (line 4). The size of the initial population depends on the specific application domain. In any case, the size of this initial population should be small in order to reduce interference to the lowest level possible. A loop structure allows the interspersed execution of the underlying system and the creation and initialization of new agents (line 7). This loop is executed until some user-specified stopping criteria are met. Stopping criteria can be specific to the application or related to the ISL framework. For example, the framework may stop when the task assigned to the swarm intelligence system is completed or when a maximum number of agents are reached. While executing the main loop, agent addition criteria, which are also supplied by the user, are repeatedly evaluated (line 8). The criteria can range from a predefined schedule to conditions based on statistics of the system's progress. If the agent addition criteria are not met, the set of agents work normally, that is, the underlying swarm intelligence system is executed. In line 9, such an event is denoted by a call to the procedure default($\mathbf{X}^t, \mathbf{E}^t$). If the agent addition criteria are satisfied, a new agent is created (line 11). In contrast to a default initialization such as the one in line 4, this new agent is initialized with information extracted from a subset of the currently active population (line 12). Such an initialization is denoted by a call to the procedure slearn($a_{new}, \mathbf{X}^t$). This procedure is responsible for the selection of the agents from which the new agent will learn, and for the actual implementation of the social learning mechanism. Once the new agent is properly initialized, it becomes part of the system (line 13). In line 15, we explicitly update the environment. However, in a real implementation, the environment may be continuously updated as a result of the system's operation.

In most swarm intelligence systems, the population of agents is large and homogeneous, that is, it is composed of agents that follow exactly the same behavioral rules. Thus, any knowledge acquired by an agent is likely to be useful for another one. The social learning mechanism used in an instantiation of the ISL framework should allow the transfer of knowledge from one agent to the other. In some cases, it is possible to have access to the full state of the agent that serves as a "model" to be imitated, and thus, the social learning mechanism is simple. In other cases, access to the model agent's state may be limited and a more sophisticated mechanism is required. In most cases, the result of the social learning mechanism will not be simply a copy of the model agent's state, but a biased initialization toward it. Copying is not always a good idea because what may work very well for an agent in a system composed of n agents may not work well in a system of $n + 1$ agents.

3.3 Related Work

The ISL framework and many works in the field of multiagent systems (Wooldridge, 2009) share a common goal: interference reduction. The means used by these works and the ISL framework to achieve this goal differ. In traditional multiagent systems, interference is a problem that has been tackled indirectly through the careful design of interaction protocols that consider all the possible events that the agents can possibly experience (Shoham and Tennenholtz, 1995; Gmytrasiewicz and Durfee, 2000). Examples of protocols designed in such a way are the following: Contract Net (Smith, 1980), coalition formation algorithms (Shehory and Kraus, 1998), or the protocols used for negotiation in agent-mediated electronic commerce applications (He et al., 2003). Tackling interference has required a significant effort on the part of the multiagent systems community. These efforts could be grouped into categories such as methodologies, standards, or communication protocols. Early on in the development of the field of multiagent systems, researchers recognized that for analyzing and designing multiagent systems, new methodologies were required. Well-known methodologies that are the result of work in this direction are MaSE (Deloach et al., 2001) and the Gaia methodology (Zambonelli et al., 2003). Through these methodologies, interactions between agents are identified and carefully designed. Standards have been proposed to allow interoperability of agents developed by different parties. The best known organization dedicated to establish specifications for multiagent systems is the Foundation for Intelligent Physical Agents (FIPA)[2] (O'Brien and Nicol, 1998). A sign that interactions are one of the main issues in the design of multiagent systems is that the core FIPA specification is the one related to agent communication. Methodologies and standards call for a common communication language between the agents that comprise a system. As a result, some agent languages have been proposed. For example, languages such as KQML (Finin et al., 1994), or FIPA-ACL (IEEE Foundation for Intelligent Physical Agents, 2011) have

[2]http://www.fipa.org

explicit specifications that let agents exchange knowledge with each other.

A complete review of the literature in the field of multiagent systems that deals with interference, either directly or indirectly, is out of the scope of this dissertation. However, we can say that there are strong differences between practically all previous works in the field of multiagent systems and the ISL framework. First, the size of the systems that can be designed with a traditional approach is limited to just a few and very sophisticated agents. Moreover, when taking a traditional approach, one is necessarily assuming that the number of agents is constant over time. This assumption is needed because with traditional approaches, each agent plays a specific role in the system, and adding or removing an agent would require the designer to re-program all or at least some of the agents that comprise the system.

In contrast, in the ISL framework we assume that the agents are very similar, if not identical, to each other. As a result, since each agent does not play a specific role, it is possible to assume that the number of agents can change over time and that the total number of agents can be very large. Thus, even though the framework may work for small systems, we are proposing the framework to be primarily used with systems composed of a large number of agents. Hence, we expect the ISL framework to have a larger impact on the design and operation of swarm intelligence systems than on the design and operation of small multiagent systems.

The other body of literature that is related to the ISL framework is the one in which social learning or related concepts are used in the context of multiagent systems and swarm intelligence systems. Two main categories of works can be distinguished: (i) those that study social learning using a multiagent system as a tool, and (ii) those that exploit social learning as a tool for developing better performing systems. The ISL framework belongs to this second category of works. Until recently, the first category was the most active of the two. Simulations of social systems in computers began in the 1950s (Conte et al., 1998) and have continued gaining popularity. This increased popularity is evidenced by the fact that there are now scholarly journals, such as the Journal of Artificial Societies and Social Simulation (JASS)[3], devoted to the topic. Areas of interest in this category range from the study of the usefulness of social learning under different environmental conditions (Annunziato and Pierucci, 2003; Noble and Franks, 2003; van der Post and Hogeweg, 2004; Priesterjahn and Eberling, 2008) to the evolution of language and culture (Divina and Vogt, 2006; Vogt, 2006). The second category of works has being attracting the attention of a growing community. Social learning as a mechanism to improve the performance of systems composed of many agents has been investigated in the context of robotics (Matarić, 1997; Pini and Tuci, 2008; Cakmak et al., 2010), multiagent systems (Kopp and Graeser, 2006; García-Pardo et al., 2010), and neural computation (Jang and Cho, 2002).

In the swarm intelligence field, social learning concepts have been associated with PSO algorithms almost since they were first proposed. Kennedy (2006) explains how the development of the first PSO algorithm was heavily influenced by Latané's social impact theory (Latané, 1981). This theory argues that an individual changes its psychological state to a degree that is a function of the strength, immediacy, and the number of other individuals. In the context of PSO algorithms, this theory was another source of inspiration for the rules that govern the movement of particles. Although swarm intelligence is based on the idea that the actions of one agent can affect the behavior of another agent, for instance, via stigmergy (see Section 2.1), social learning has been overlooked by researchers in the field. We hope that this dissertation makes social learning research more visible to the swarm intelligence community, and that the community of scientists studying social learning in animals becomes aware of the potential of swarm intelligence as a hypothesis and application testing field. We hope that the mutual exchange of ideas will serve to enrich both fields.

In the next two chapters, we will describe the case studies designed to test the effectiveness of the ISL framework. Previous work specifically related to the instantiation of the ISL framework in the context of each case study is presented in the corresponding chapter.

[3]`http://jasss.soc.surrey.ac.uk`

Chapter 4

Incremental Social Learning Applied to Particle Swarms

In this chapter, we describe the first of the two case studies that we use to evaluate the effectiveness of the ISL framework. In the instantiation of the ISL framework described in this chapter, a PSO algorithm serves as the underlying swarm intelligence system. Three different algorithms are presented. The first two algorithms are straightforward instantiations of the ISL framework. The third algorithm is the result of a more elaborate design process in which automatic tuning plays an important role.

In Section 2.2, we said that the meanings of "experience" and "adaptation" need to be explicitly defined in order to understand the role of social learning in a specific instantiation of the ISL framework. In the case study presented in this chapter, these terms are intimately related to the purpose of particles in a PSO algorithm. The term "experience" is the memory that each particle maintains about its search history, that is, the best solution found since the beginning of the algorithm's run. The term "adaptation" is interpreted as the actual quality of that best-found solution. Therefore, in the context of the case study presented in this chapter, "learning" is interpreted as a process through which a particle's memory is used in order to find better solutions to the optimization problem at hand. The social learning procedure used throughout this chapter (see Section 4.1) is consistent with this interpretation.

As mentioned in Section 2.1.2, PSO is a direct search method. Thus, no assumptions are made regarding the features of the problems PSO is applied to. In other words, our experiments are carried out under the assumption that the objective function's derivatives are not available and that only direct function evaluations can be performed. In this context, interference in PSO algorithms is seen as a trade-off between solution quality and the number of function evaluations used. This trade-off is greatly affected by the size of the population: When a limited number of function evaluations are allowed, small populations obtain the best results. In contrast, when solution quality is the most important aspect, large populations usually work better (van den Bergh and Engelbrecht, 2001) (see also Appendix A). Thus, the analysis of the benefits due to the use of the ISL framework is based on the solution quality obtained after a certain number of function evaluations or the number of function evaluations needed to find a solution of a certain quality.

4.1 Incremental Particle Swarm Optimizer

The first instantiation of the ISL framework in the context of PSO algorithms is an algorithm with a growing population size that we call incremental particle swarm optimizer (IPSO). IPSO is based on the *constricted* PSO algorithm, which strictly speaking is a particular setting of numerical parameters of the standard PSO algorithm. However, this setting has become so popular since it was proposed by Clerc and Kennedy (2002) that we refer to it as a variant.

Clerc and Kennedy (2002) modified Eq. 2.3 and introduced a constant called *constriction factor*. The goal of this modification was to avoid the unlimited growth of the particles' velocity that may occur with certain parameter settings (Clerc and Kennedy, 2002). In the constricted PSO variant, the velocity update rule for particle i's j-th component is

$$v_{i,j}^{t+1} = \chi\left(v_{i,j}^{t} + \varphi_1 U_1 (pb_{i,j}^{t} - x_{i,j}^{t}) + \varphi_2 U_2 (lb_{i,j}^{t} - x_{i,j}^{t})\right) , \tag{4.1}$$

where χ is the constriction factor. The value taken by the constriction factor is based on the following relation: $\chi = 2/\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|$, where $\varphi = \varphi_1 + \varphi_2$ and $\varphi > 4$. Note that Eqs. 2.3 and 4.1 can be equivalent if the values of w and φ_1 and φ_2 are set appropriately.

In the ISL framework, every time a new agent is added to the population, it learns socially from a subset of the more experienced agents. In IPSO, every time a new particle is added, it is initialized using information from particles that have already been part of the swarm for some time. This social learning mechanism is implemented as an initialization rule that moves a new particle from an initial randomly generated position in the problem's search space to one that is closer to the position of a particle that serves as a "model" to imitate (hereafter referred to as *model particle*). The initialization rule used in IPSO, as applied to a new particle's j-th dimension, is as follows:

$$x'_{new,j} = x_{new,j} + U \cdot (p_{model,j} - x_{new,j}), \tag{4.2}$$

where $x'_{new,j}$ is the new particle's updated position, $x_{new,j}$ is the new particle's original random position, $p_{model,j}$ is the model particle's previous best position, and U is a uniformly distributed random number in the range $[0, 1)$. Once the rule is applied for each dimension, the new particle's previous best position is initialized to the point $\boldsymbol{x}'_{new}$ and its velocity is set to zero. The random number U is the same for all dimensions in order to ensure that the new particle's updated previous best position will lie somewhere along the direct attraction vector $\boldsymbol{p}_{model} - \boldsymbol{x}_{new}$. Using independent random numbers for each dimension would reduce the strength of the bias induced by the initialization rule because the resulting attraction vector would be rotated and scaled with respect to the direct attraction vector. Finally, the new particle's neighborhood, that is, the set of particles from which it will receive information in subsequent iterations, is generated at random, respecting the connectivity degree of the swarm's population topology. A pseudo-code version of IPSO is shown in Algorithm 4.

The model particle can be selected in several ways. Here we present the results obtained with the best particle as a model. Experimental results indicate that choosing the model particle at random does not produce significantly different results. We conjecture that this result is due to the tendency that particles have to cluster in the search space. In such a case, the distance between the best and a random particle would not be large enough to produce significantly different results.

4.2 Incremental Particle Swarm Optimizer with Local Search

The second instantiation of the ISL framework in the context of PSO algorithms is an incremental particle swarm optimizer with local search (IPSOLS). This algorithm works similarly to IPSO; However, in IPSOLS, particles not only move using the traditional PSO rules, but also by invoking a local search procedure. In the context of the ISL framework, the local search procedure can be interpreted as a particle's "individual learning" ability because it allows a particle to improve its solution in the absence of any social influence.

In IPSOLS, the local search procedure is called only when it is expected to be beneficial; the local search procedure is called only when a particle's previous best position is not considered to be already in a local optimum. We determine when to call again the local search procedure by letting the local search procedure return a value that indicates either

Algorithm 4 Incremental particle swarm optimizer: IPSO

```
Input: Objective function $f : \Theta \subseteq \mathbb{R}^n \to \mathbb{R}$, the initialization domain $\Theta' \subseteq \Theta$, the agent-addition
  criterion $A$, the maximum population size $N$, and the parameters used by the PSO rules ($\varphi_1$,
  $\varphi_2$, and $\chi$).
Output: The best found solution $\boldsymbol{sol} \in \Theta$

  /* Initialization */
  $t \leftarrow 0$ /* Iteration counter */
  $i \leftarrow 1$ /* Population size */
  Initialize position vector $\boldsymbol{x}_i^t$ to random values within $\Theta'$
  Initialize velocity vector $\boldsymbol{v}_i^t$ to zero
  $\boldsymbol{p}_i^t \leftarrow \boldsymbol{x}_i^t$

  /* Main Loop */
  repeat
    /* PSO Rules */
    for $j = 1$ to $i$ do
      Generate $\boldsymbol{x}_j^{t+1}$ using Eqs. 4.1 and 2.4
      if $f(\boldsymbol{x}_j^{t+1}) < f(\boldsymbol{p}_j^t)$ then
        $\boldsymbol{p}_j^{t+1} \leftarrow \boldsymbol{x}_j^{t+1}$
      end if
    end for

    /* Population Growth and Social Learning */
    if Particle addition criterion $A$ is met and $i < N$ then
      Initialize vector $\boldsymbol{p}_{i+1}^{t+1}$ using Eq. 4.2 for each component
      Initialize velocity vector $\boldsymbol{v}_{i+1}^{t+1}$ to zero
      $\boldsymbol{x}_{i+1}^{t+1} \leftarrow \boldsymbol{p}_{i+1}^{t+1}$
      $i \leftarrow i + 1$
    end if
    $t \leftarrow t + 1$
    $\boldsymbol{sol} \leftarrow \underset{j \in \{1,\dots,i\}}{\operatorname{argmin}} f(\boldsymbol{p}_j^t)$
  until Stopping criterion is satisfied
```

that it finished because a very small difference between two solutions was detected or that the maximum number of iterations allocated to it was reached. In the first case, it is assumed that the local search has converged to a local optimum, and the particle does not invoke the procedure again because no further significant improvements are expected. In the second case, the particle may call the local search procedure again because further significant improvements can still be achieved. The two parameters of the local search procedure that control these exit criteria are the tolerance and the maximum number of iterations respectively. IPSOLS is sketched in Algorithm 5.

In principle, any local search algorithm for continuous optimization can be used with IPSOLS. In the first set of experiments (reported in Section 4.4), we use Powell's conjugate directions set method using Brent's technique (Brent, 1973) as the auxiliary line minimization algorithm. In Section 4.5.2, we explore the impact that the selection of a specific local search algorithm has on the performance of IPSOLS.

Powell's conjugate directions set method tries to minimize an n-dimensional objective function by constructing a set of *conjugate directions* through a series of line searches. Directions $\boldsymbol{v}_i,\ \ i \in \{1 : n\}$, are said to be conjugate with respect to an $n \times n$ positive definite matrix A, if

$$\boldsymbol{v}_i^T A \boldsymbol{v}_j = 0, \qquad \forall i, j \in \{1 : n\},\ i \neq j\,.$$

Additionally, directions $\boldsymbol{v}_i,\ \ i \in \{1 : n\}$, must satisfy linear independence to be considered conjugate. Conjugate search directions are attractive because if A is the Hessian

Algorithm 5 Incremental particle swarm optimizer with local search: IPSOLS

```
Input: Objective function $f : \Theta \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, the initialization domain $\Theta' \subseteq \Theta$, the agent-addition
  criterion $A$, the maximum population size $N$, the local search procedure parameters (tolerance,
  maximum number of iterations, step size) and the parameters used by the PSO rules ($\varphi_1$, $\varphi_2$,
  and $\chi$).
Output: The best found solution $\boldsymbol{sol} \in \Theta$

  /* Initialization */
  $t \leftarrow 0$ /* Iteration counter */
  $i \leftarrow 1$ /* Population size */
  Initialize position vector $\boldsymbol{x}_i^t$ to random values within $\Theta'$
  Initialize velocity vector $\boldsymbol{v}_i^t$ to zero
  $\boldsymbol{p}_i^t \leftarrow \boldsymbol{x}_i^t$
  $e_i \leftarrow$ true /* If $e_i =$ true, a local search should be invoked for particle $i$ */

  /* Main Loop */
  repeat
    /* Local Search */
    for $j = 1$ to $i$ do
      if $e_j =$ true then
        $e_j \leftarrow$ localsearch($f, \boldsymbol{p}_j^t$) /* true if exited without converging, else returns false */
      end if
    end for

    /* PSO Rules */
    for $j = 1$ to $i$ do
      Generate $\boldsymbol{x}_j^{t+1}$ using Eqs. 4.1 and 2.4
      if $f(\boldsymbol{x}_j^{t+1}) < f(\boldsymbol{p}_j^t)$ then
        $\boldsymbol{p}_j^{t+1} \leftarrow \boldsymbol{x}_j^{t+1}$
        $e_j \leftarrow$ true
      end if
    end for

    /* Population Growth and Social Learning */
    if Agent-addition criterion $A$ is met and $i < N$ then
      Initialize vector $\boldsymbol{p}_{i+1}^{t+1}$ using Eq. 4.2 for each component
      Initialize velocity vector $\boldsymbol{v}_{i+1}^{t+1}$ to zero
      $\boldsymbol{x}_{i+1}^{t+1} \leftarrow \boldsymbol{p}_{i+1}^{t+1}$
      $e_{i+1} \leftarrow$ true
      $i \leftarrow i + 1$
    end if

    $t \leftarrow t + 1$
    $\boldsymbol{sol} \leftarrow \underset{j \in \{1,\ldots,i\}}{\text{argmin}} f(\boldsymbol{p}_j^t)$
  until Stopping criterion is satisfied
```

matrix of the objective function, it can be minimized in exactly n line searches (Press et al., 1992).

Powell's conjugate directions set method starts from an initial point $\mathbf{p}_0 \in \mathbb{R}^n$. It then performs n line searches using the unit vectors $\boldsymbol{e}_i$ as initial search directions $\boldsymbol{u}_i$. A parameter of the algorithm is the initial step size s of the search. At each step, the new initial point from which the next line search is carried out is the point where the previous line search found a relative minimum. A point $\mathbf{p}_n$ denotes the minimum discovered after all n line searches. Next, the method eliminates the first search direction by doing $\boldsymbol{u}_i = \boldsymbol{u}_{i+1}, \ \forall i \in \{1 : n-1\}$, and replacing the last direction $\boldsymbol{u}_n$ for $\mathbf{p}_n - \mathbf{p}_0$. Then, a move to the minimum along the direction $\boldsymbol{u}_n$ is performed. The next iteration is executed starting from the minimum found in the last step. We used an implementation of the method similar to the one described by Press et al. (1992). In this implementation, line searches receive a parameter referred to as *step size* that determines the initial points from which the method performs its initial bracketing steps. The line search method is restricted to a line segment equal to 100 times the length of the step size. Thus, the line search procedure is not constrained within a line segment of length equal to the step size parameter, but can potentially explore the full search direction. Our implementation terminates when the maximum number of iterations, MaxITER, is reached or when the tolerance FTol, that is, the relative change between solutions found in two consecutive iterations, falls below a certain threshold.

4.3 Determining the Probability Density Function Induced by the Social Learning Rule

The position of a newly added particle in IPSO and IPSOLS can be modeled as a random variable Z which is a function of two independent continuous random variables X and Y. X is a uniformly distributed random variable in the complete initialization range $[x_{min}, x_{max})$, while Y is a uniformly distributed random variable in the range $[0, 1)$. Y determines the strength of the attraction toward the position of the particle used as a model (the best particle in the swarm in our case). The model's position is, strictly speaking, also a random variable due to the fact that it is the result of a number of iterations of the PSO position-update mechanism. However, when the initialization rule is invoked, the model's position can be taken as a constant. Based on Eq. 4.2, Z is defined as follows:

$$Z = X + Y(c - X)\,, \tag{4.3}$$

where $x_{min} \le c < x_{max}$ is a constant representing the location of the attractor particle.

The distribution function F_Z of Z is given by

$$F_Z(a) = P(Z \le a) = \iint\limits_{(x,y):x+y(c-x)\le a} f(x,y)\,dx\,dy\,, \tag{4.4}$$

where $f(x, y)$ is the joint probability distribution of X and Y.

Since X and Y are independent, we have that

$$f(x,y) = f_X(x) f_Y(y) = \frac{1}{x_{max} - x_{min}}\,, \tag{4.5}$$

where f_X and f_Y are the marginal probability functions of X and Y respectively. This holds for $x_{min} \le x < x_{max}$ and $0 \le y < 1$.

Using 4.5 and considering that $y = \frac{a-x}{c-x}$, we can rewrite 4.4 as follows

$$\begin{aligned} F_Z(a) &= \frac{1}{x_{max} - x_{min}} \int_{x_{min}}^{x_{max}} y\,dx \\ &= \frac{1}{x_{max} - x_{min}} \int_{x_{min}}^{x_{max}} \frac{a-x}{c-x}\,dx\,. \end{aligned} \tag{4.6}$$

Eq. 4.6 must be solved in two parts: when $x_{min} \le x \le a < c$ and when $c < a \le x < x_{max}$. In the special case when $x = c$, $F_Z(a) = c/(x_{max} - x_{min})$ (see Eq. 4.3).

When $x_{min} \le x \le a < c$, we obtain

$$\begin{aligned} F_Z(a) &= \frac{1}{x_{max} - x_{min}} \int_{x_{min}}^{a} \frac{a-x}{c-x}\,dx \\ &= \frac{1}{x_{max} - x_{min}} \left[a + (a-c) \ln \left| \frac{c - x_{min}}{c-a} \right| \right]\,. \end{aligned} \tag{4.7}$$

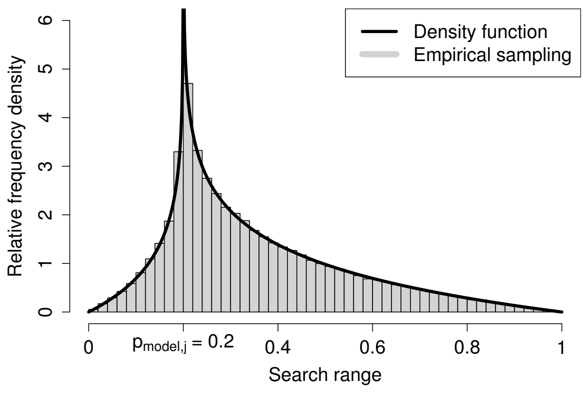


Figure 4.1: Probability density function induced by the initialization rule of new particles. In the figure, the attractor $p_{model,j} = 0.2$. The initialization range in this example is $[0, 1)$. The figure shows both the analytical density function and the density histogram obtained using Monte Carlo simulation (10^5 samples).

When $c < a \leq x < x_{max}$, we obtain

$$\begin{aligned} F_Z(a) &= \frac{1}{x_{max} - x_{min}} \left[1 - \int_a^{x_{max}} \frac{a - x}{c - x}\, dx \right] \\ &= \frac{1}{x_{max} - x_{min}} \left[a + (a - c) \ln \left| \frac{c - x_{max}}{c - a} \right| \right] . \end{aligned} \tag{4.8}$$

Hence the probability density function f_Z of Z is given by

$$f_Z(z) = \frac{d}{dz} F_Z(z) = \frac{1}{x_{max} - x_{min}} \begin{cases} \ln \left| \frac{c - x_{min}}{c - z} \right| & \text{if } z < c \\ 0 & \text{if } z = c \\ \ln \left| \frac{c - x_{max}}{c - z} \right| & \text{if } z > c\,. \end{cases} \tag{4.9}$$

Changing variables, we obtain the probability density function induced by the social learning rule for dimension j:

$$f_{X_j}(x_j) = \frac{1}{x_{max,j} - x_{min,j}} \cdot \begin{cases} \ln \left| \frac{p_{model,j} - x_{min,j}}{p_{model,j} - x_j} \right| & \text{if } x_j < p_{model,j} \\ 0 & \text{if } x_j = p_{model,j} \\ \ln \left| \frac{p_{model,j} - x_{max,j}}{p_{model,j} - x_j} \right| & \text{if } x_j > p_{model,j}\,, \end{cases} \tag{4.10}$$

where $x_{min,j}$ and $x_{max,j}$ are the minimum and maximum limits of the initialization range over the problem's jth dimension and $x_{min,j} \leq x_j < x_{max,j}$. Figure 4.1 shows the exact density function and a density histogram obtained using Monte Carlo simulation when the initialization range is $[0, 1)$ and $p_{model,j} = 0.2$. In a density histogram, the height of each rectangle is equal to $\frac{o_i}{w_i N}$, where o_i is the number of observations of class i in an experiment of N samples. The value w_i is known as class i's width, and it is the length of the range that defines class i. In our case, we set the class width to $w_i = 0.02$.

Most of the samples concentrate around the model's position as desired. Note, however, that there is a nonzero probability of sampling regions far away from the model. This probability distribution offers a certain level of exploration-by-initialization which would be difficult to obtain with a normally distributed initialization around the model particle's position. The problem would be that setting the right value for the standard deviation would depend on the model particle's position. The probability density function induced

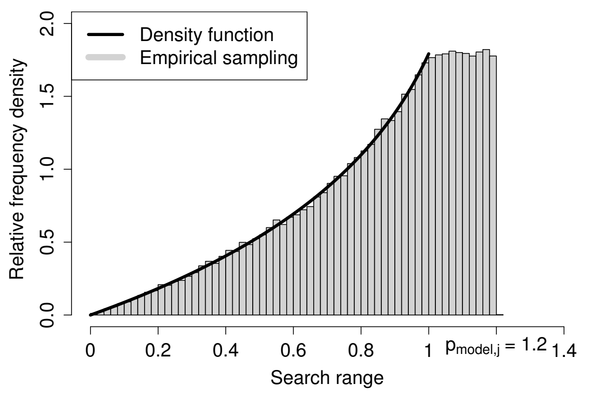


Figure 4.2: Probability density function induced by the initialization rule of new particles when the attractor lies outside the original initialization range. In the figure, the attractor $p_{model,j} = 1.2$. The initialization range is $[0, 1)$. The figure shows that the density function follows the analytical density function up to the limit of the original initialization range. The histogram obtained using Monte Carlo simulation (10^5 samples) shows the actual density function.

by the new particles initialization rule is not symmetric except in the case $p_{model,j} = (x_{max,j} + x_{min,j})/2$. The expected value of a new particle's position is the following:

$$\begin{aligned} E(x'_{new,j}) &= E(x_{new,j}) + E(U)\,(p_{model,j} - E(x_{new,j})) \\ &= E(x_{new,j}) + \frac{1}{2}\,(p_{model,j} - E(x_{new,j})) \\ &= \frac{x_{max,j} + x_{min,j}}{4} + \frac{p_{model,j}}{2}\,. \end{aligned} \tag{4.11}$$

The analysis presented above is valid only if the attractor particle's position is within the range $[x_{min,j}, x_{max,j})$. If the attractor is outside the initialization range, the probability density function remains the same within the initialization range. However, the probability density function is similar to a uniform distribution outside this range (see Figure 4.2).

Under these conditions, a new particle will follow the model from only one of its sides. The initialization rule is not able to position a new particle beyond the location of the attractor particle if this particle is outside the original initialization range. This effect is not necessarily a drawback because one would usually expect the sought global optimum to lie within the chosen initialization region.

4.4 Experimental Evaluation

In this section, we first describe the setup used to carry out our experiments. Next, we present and discuss the results of the empirical performance evaluation of the algorithms presented in Sections 4.1 and 4.2.

4.4.1 Setup

The performance of IPSO and IPSOLS was compared to that of the following algorithms:

1. A constricted PSO algorithm with constant population size. This algorithm was included in the evaluation in order to measure the contribution of the incremental population component used in IPSO. This algorithm is labeled as PSO-X, where X is the population size.

2. A recently proposed PSO algorithm, called EPUS, in which the population size varies over time (Hsieh et al., 2009). This algorithm increases the population size by one if the best-so-far solution is not improved during h consecutive iterations and if the current population size is not larger than a maximum limit. The new particle's position is equal to the result of a crossover operation on the personal best positions of two randomly selected particles. If the best-so-far solution is improved during h consecutive iterations, the worst particle of the swarm is removed from the swarm unless the population size falls below a minimum limit after the operation. Finally, if the population size is equal to the maximum limit but the swarm is unable to improve the best-so-far solution during h consecutive iterations, the worst particle is replaced by a new one. We do not use the mutation and solution sharing mechanisms described in (Hsieh et al., 2009) in order not to confound the effects of the variable population size with those of these operators.

3. A hybrid particle swarm optimization algorithm with local search (labeled PSOLS). This algorithm is a constant population size particle swarm algorithm in which the particles' previous best positions undergo an improvement phase (via Powell's conjugate directions set method) before the velocity update rule is applied. The local search is only applied when a particle's previous best position is not located in a local optimum, just as is done in IPSOLS. PSOLS was included in the evaluation because, by comparing its performance to that of IPSOLS, we can measure the contribution of the incremental population component in combination with a local search procedure.

4. A hybrid algorithm (labeled EPUSLS) that combines EPUS with local search (Powell's conjugate directions set method). This algorithm allows us to measure the relative performance differences that may exist between purely increasing and variable population size approaches in combination with a local search procedure. The same parameter settings used for EPUS were used for EPUSLS.

5. A random restart local search algorithm (labeled RLS). Every time the local search procedure (also Powell's conjugate directions set method) converges, it is restarted from a newly generated random solution. The best solution found so far is considered to be the output of the algorithm. This algorithm was considered a baseline for the evaluation of the effectiveness of the PSO component in EPUSLS, PSOLS, and IPSOLS.

All algorithms were run on a set of twelve commonly used benchmark functions whose mathematical definition is shown in Table 4.1. In all cases, we used the 100-dimensional versions of the benchmark functions ($n = 100$). In our experimental setup, each algorithm was run with the same parameter settings across all benchmark functions. The parameter settings used for each algorithm are the most commonly used in the PSO literature. These settings are listed in Table 4.2.

Our results are based on statistics taken from 100 independent runs, each of which was stopped whenever one of the following criteria was met: (i) 10^6 function evaluations had been performed, or (ii) the objective function value was less than or equal to 10^{-15}. However, it is still possible to find solutions with a lower value than this threshold because the stopping criteria were evaluated outside the local search procedure. To eliminate the effects of any possible search bias toward the origin of the coordinate system, at each run, a benchmark function was randomly shifted within the specified search range. Functions Schwefel and Step were not shifted since their optima are not at the origin of the coordinate system. Bound constraints were enforced by putting variable values of candidate solutions on the corresponding bounds. This mechanism proved to be effective and easily applicable to both PSO and local search components.

PSOLS was run with fewer particles than PSO because larger populations would have prevented us from observing the effects that are due to the interaction of the PSO and local search components, given the stopping criteria used. Given the number of function evaluations required by each invocation of the local search procedure and the maximum

Table 4.1: Benchmark functions used for evaluating IPSO and IPSOLS

Name	Definition	Search Range $[x_{min},x_{max}]^n$
Ackley	$-20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}}-e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}+20+e$	[-32,32]^n
Axis-parallel Hyper-ellipsoid	$\sum_{i=1}^{n}(ix_i)^2$	[-100,100]^n
Expanded Schaffer	$ES(\boldsymbol{x})=\sum_{i=1}^{n-1} S(x_i,x_{i+1})+S(x_n,x_1)$, where $S(x,y)=0.5+\frac{\sin^2(\sqrt{x^2+y^2})-0.5}{(1+0.001(x^2+y^2))^2}$	[-100,100]^n
Griewank	$\frac{1}{4000}\sum_{i=1}^{n} x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	[-600,600]^n
Penalized function	$\frac{\pi}{n}\{10\sin^2(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right]+(y_n-1)^2\} + \sum_{i=1}^{n}\mathrm{u}(x_i,10,100,4)$, where $y_i=1+(x_i+1)/4\,,\ \mathrm{u}(x,a,k,m)=\begin{cases} k(x_i-a)^m & \text{if } x_i>a \\ k(-x_i-a)^m & \text{if } x_i<a \\ 0 & \text{otherwise}\end{cases}$	[-50,50]^n
Rastrigin	$10n+\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i))$	[-5.12,5.12]^n
Rosenbrock	$\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	[-30,30]^n
Salomon	$1-\cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right)+0.1\sqrt{\sum_{i=1}^{n} x_i^2}$	[-100,100]^n
Schwefel	$418.9829n+\sum_{i=1}^{n}-x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	[-500,500]^n
Sphere	$\sum_{i=1}^{n} x_i^2$	[-100,100]^n
Step	$6n+\sum_{i=1}^{n}\lfloor x_i\rfloor$	[-5.12,5.12]^n
Weierstrass	$\sum_{i=1}^{n}\left(\sum_{k=1}^{k_{max}} a^k\cos\left(2\pi b^k(x_i+0.5)\right)\right)-n\sum_{k=1}^{k_{max}}\left[a^k\cos\left(\pi b^k\right)\right]$, where $a=0.5,\ b=3,\ k_{max}=20$	[-0.5,0.5]^n

Table 4.2: Parameter settings used for evaluating IPSO and IPSOLS

Setting(s)	Algorithm(s)
Acceleration coefficients: $\varphi_1=\varphi_2=2.05$	All except RLS
Constriction coefficient: $\chi=0.729$	All except RLS
Maximum velocity: $V_{max}=\pm x_{max}$	All except RLS
Population size: $10, 100, 1000$ particles	PSO
Population size: $5, 10, 20$ particles	PSOLS
Population size control parameter: $h=2$	EPUS, EPUSLS
Minimum population size: 1 particle	EPUS, EPUSLS, IPSO, IPSOLS
Maximum population size: 1000 particles	EPUS, EPUSLS, IPSO, IPSOLS
Model particle for initialization: Best	IPSO, IPSOLS
Powell's method tolerance: 0.01	EPUSLS, IPSOLS, PSOLS, RLS
Powell's method maximum number of iterations: 10	EPUSLS, IPSOLS, PSOLS, RLS
Powell's method step size: 20% of the length of the search range	EPUSLS, IPSOLS, PSOLS, RLS

number of function evaluations allocated for each experiment, a large population would essentially behave as a random restart local search, which was included in the comparison.

All particle swarm-based algorithms (PSO, PSOLS, EPUS, EPUSLS, IPSO and IPSOLS) were run with two population topologies: a fully connected topology, in which each particle is a neighbor to all others including itself, and the so-called ring topology, in which each particle has two neighbors apart from itself. In the incremental algorithms, the new particle is randomly placed within the topological structure.

4.4.2 Performance Evaluation Results

Algorithms for continuous optimization are often evaluated according to two different criteria. One of these criteria measures the quality of the solutions (through the objective function values associated with them) that algorithms are able to find, given a maximum number of function evaluations. The other criterion measures the number of function evaluations needed by algorithms to reach a target objective function value. Since the algorithms used in our study are stochastic, both performance measures are also stochastic. For this reason, we look at the distribution of the objective function values obtained after a certain number of function evaluations (the number of function evaluations used for tackling a problem is also called *run length* (Hoos and Stützle, 2004)), and the distribution of the number of function evaluations needed to reach some target objective function values.[1] We also look at some central tendency measures to have a more aggregated summary of the performance of the compared algorithms. Finally, we present a summary of the statistical data analysis performed on all the data. In the discussion of the results, we pay particular attention to the two main components of the ISL-based algorithms: the variable population size and the use of a local search procedure.

Particle Addition Schedule

The first aspect that we investigate is the effect of the particle addition schedule on the performance of IPSO and IPSOLS. Figure 4.3 shows the average solution quality[2] obtained with three instantiations of IPSO and four of the constant population size PSO as a function of the number of function evaluations used. The three instantiations of IPSO are labeled "IPSO-X", meaning that a new particle is added every X iterations. The labels used for the constant population size PSO are explained in Section 4.4.1.

Clearly, the particle addition schedule affects the exploration-exploitation behavior of IPSO. Faster schedules encourage exploration while slower ones encourage exploitation. The result of this behavior is that better solutions are found in the long run with IPSO with a fast particle addition schedule. In IPSOLS, the exploitative behavior induced by the local search procedure needs to be balanced with an exploration-encouraging, fast particle-addition schedule. Thus, in the experiments that are described next, we use the fastest particle addition schedule, that is, we add a particle every iteration of the algorithm until a maximum population size is reached. In Section 4.5, both the initial population size and the particle addition schedule are free parameters that are later tuned.

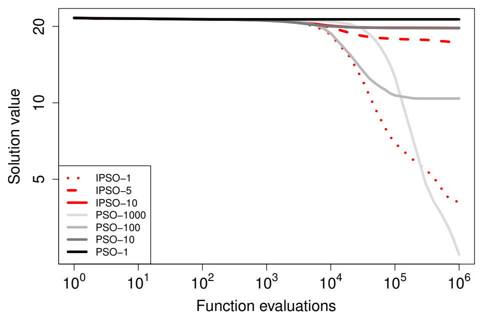
Constant *vs.* Variable Population Size

The distributions of the objective function values after 10^4 and 10^6 function evaluations are shown in Figures 4.4 and 4.5, respectively. On top of each box plot there may be two rows of symbols. The lower row, made of $+$ symbols, indicates in which cases a statistically significant difference exists between the marked algorithm and IPSO (in favor of IPSO). The upper row, made of $\times$ symbols, indicates in which cases a statistically significant difference exists between the marked algorithm and IPSOLS (in favor of IPSOLS). Significance was determined at the 5% level using a Wilcoxon test with Holm's correction method for multiple comparisons.

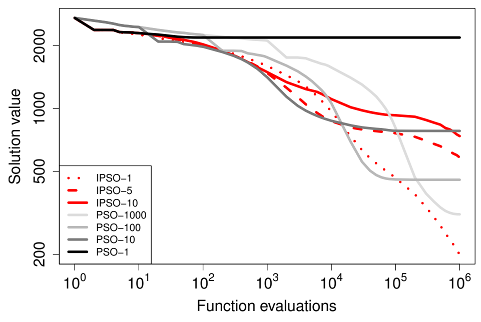
The performance of constant population size PSO algorithms without local search greatly depends on the population size. These results, together with the ones of the previous section, confirm the trade-off between solution quality and speed that we mentioned at the beginning of this chapter. Swarms of 10 particles usually find better solutions than

[1]In this dissertation's supplementary information web page Montes de Oca (2011), the reader can find the complete data that, for the sake of conciseness, are not included. Nevertheless, the main results are discussed in the text.

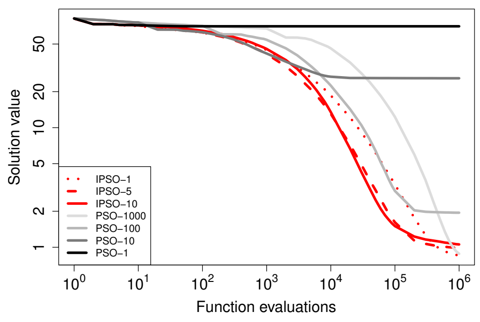
[2]The quality of a solution is its objective function value. For minimization problems, the lower the objective function value, the better. In the rest of the chapter, the terms "solution quality" and "objective function value" are used interchangeably.



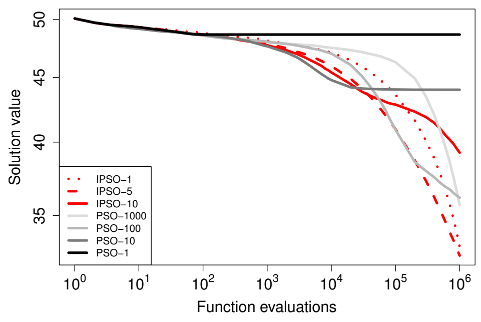
(a) Ackley



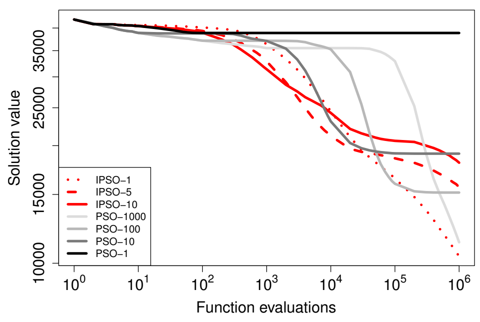
(b) Rastrigin



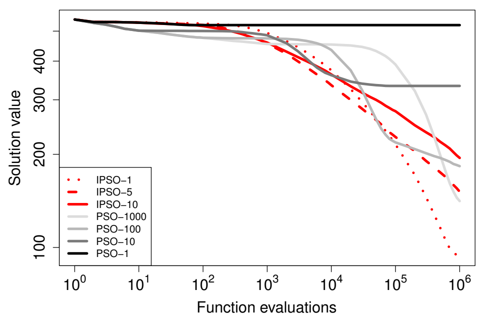
(c) Salomon



(d) Schaffer



(e) Schewfel

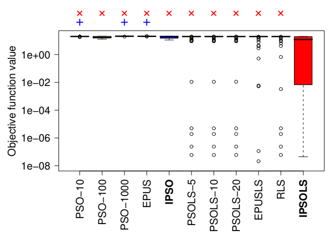


(f) Step

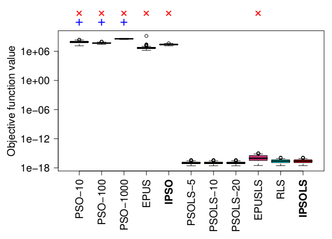
Figure 4.3: Effect of the particle addition schedules on the performance of IPSO. We plot the average objective function value as a function of the number of function evaluations. In the figure, we plot the results obtained with three particle addition schedules as well as the results obtained with four instantiations of a constant population size PSO algorithm. Fully-connected topologies were used to produce the data on which this plots are based.

larger swarms up to around 10^4 function evaluations. The swarms of 100 particles are typically the best performing after 10^5 function evaluations, and after 10^6 function evaluations, the swarms with 1000 particles often return the best solutions. This tendency can also be seen in Table 4.3, which lists the median objective function values obtained by the tested algorithms on all benchmark functions at different run lengths.

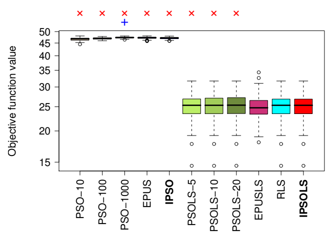
Regarding the algorithms with variable population size, it can be said that IPSO is the best among the studied algorithms for runs of up to 10^2 function evaluations. The data in Table 4.3 show that IPSO finds the best median objective function values for 11 out of the 12 functions used. IPSOLS and RLS find the best solutions for 6 out of the 12 possible



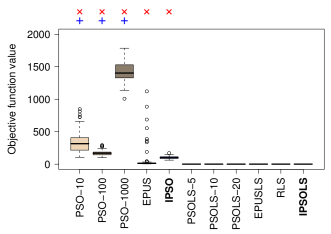
(a) Ackley



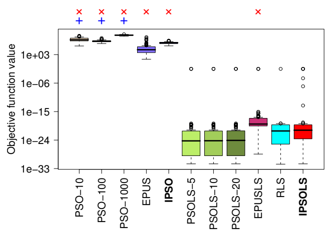
(b) Axis-parallel Hyper-ellipsoid



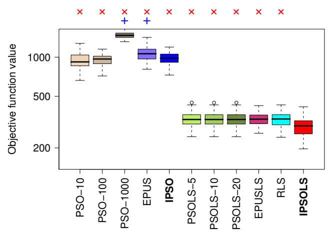
(c) Extended Schaffer



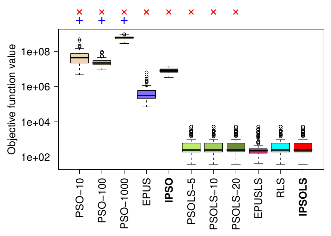
(d) Griewank



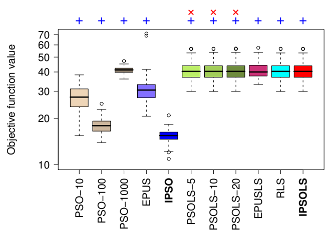
(e) Penalized function



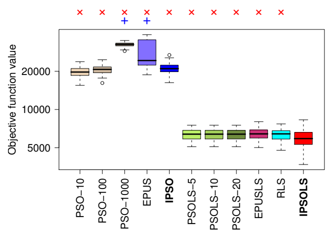
(f) Rastrigin



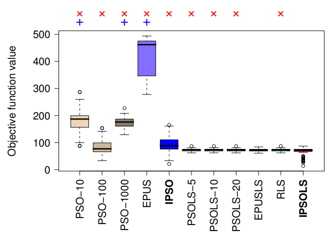
(g) Rosenbrock



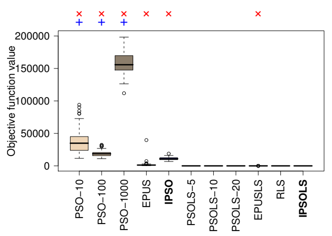
(h) Salomon



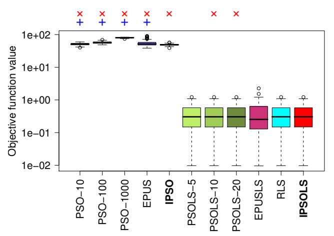
(i) Schwefel



(j) Step

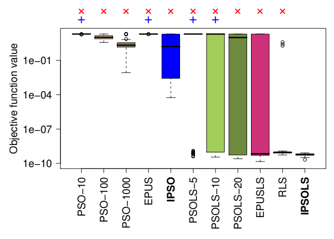


(k) Sphere

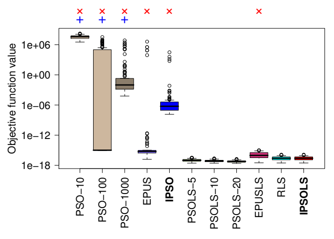


(l) Weierstrass

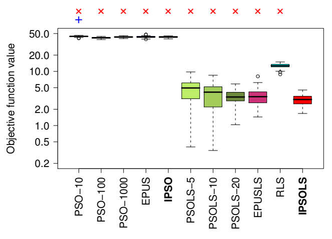
Figure 4.4: The box plots show the distribution of the solution quality obtained with the compared algorithms for runs of up to 10^4 function evaluations. These results correspond to the case in which a fully-connected topology is used with all particle swarm-based algorithms. A symbol on top of a box plot denotes a statistically significant difference at the 5% level between the results obtained with the indicated algorithm and those obtained with IPSO (in favor of IPSO, marked with a $+$ symbol) or with IPSOLS (in favor of IPSOLS, marked with a $\times$ symbol).



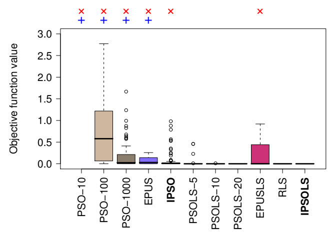
(a) Ackley



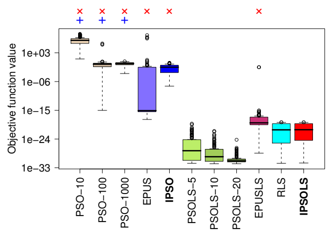
(b) Axis-parallel Hyper-ellipsoid



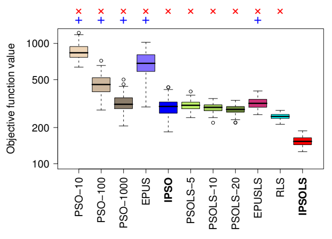
(c) Extended Schaffer



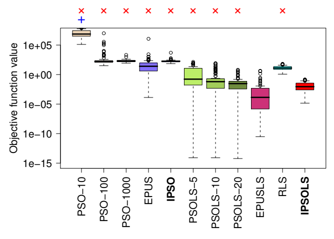
(d) Griewank



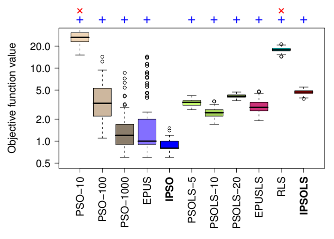
(e) Penalized function



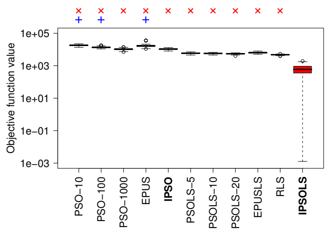
(f) Rastrigin



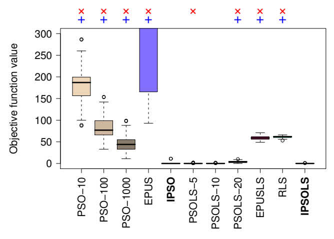
(g) Rosenbrock



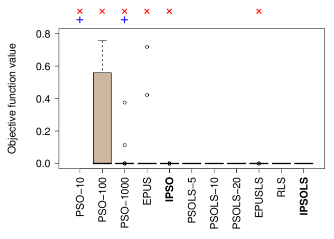
(h) Salomon



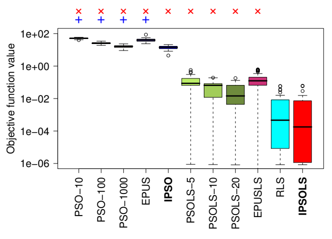
(i) Schwefel



(j) Step



(k) Sphere



(l) Weierstrass

Figure 4.5: The box plots show the distribution of the solution quality obtained with the compared algorithms for runs of up to 10^6 function evaluations. These results correspond to the case in which a fully-connected topology is used with all particle swarm-based algorithms. In the Griewank and Sphere functions, the solution values obtained with the traditional PSO algorithm with 10 particles are so much higher than those obtained with the other algorithms that its box plot does not appear. A symbol on top of a box plot denotes a statistically significant difference at the 5% level between the results obtained with the indicated algorithm and those obtained with IPSO (in favor of IPSO, marked with a $+$ symbol) or with IPSOLS (in favor of IPSOLS, marked with a $\times$ symbol).

Table 4.3: Median objective function values at different run lengths[1]

Function Evaluations	Algorithm	Benchmark function											
		Ackley	Axis-parallel Hyper-ellipsoid	Extended Schaffer	Griewank	Penalized	Rastrigin	Rosenbrock	Salomon	Schwefel	Sphere	Step	Weierstrass
10^2	PSO-10^1	**2.13e+01**	1.02e+09	**4.86e+01**	3.19e+03	9.20e+09	1.96e+03	2.73e+09	6.17e+01	3.65e+04	3.55e+05	**3.72e+02**	8.72e+01
	PSO-10^2	2.14e+01	1.48e+09	**4.86e+01**	4.42e+03	1.88e+10	2.24e+03	4.99e+09	7.08e+01	3.73e+04	4.91e+05	4.78e+02	9.01e+01
	PSO-10^3	2.14e+01	1.48e+09	**4.86e+01**	4.42e+03	1.88e+10	2.24e+03	4.99e+09	7.08e+01	3.73e+04	4.91e+05	4.78e+02	9.01e+01
	EPUS	2.14e+01	1.16e+09	4.87e+01	3.57e+03	1.02e+10	2.05e+03	3.17e+09	6.48e+01	3.86e+04	3.96e+05	5.02e+02	9.14e+01
	IPSO	**2.13e+01**	**9.54e+08**	**4.86e+01**	**3.00e+03**	**5.93e+09**	**1.84e+03**	**2.00e+09**	**5.83e+01**	**3.64e+04**	**3.33e+05**	4.04e+02	**8.63e+01**
	PSOLS-5	2.15e+01	1.89e+09	4.94e+01	5.03e+03	2.58e+10	2.45e+03	6.60e+09	7.72e+01	3.88e+04	5.59e+05	5.04e+02	9.15e+01
	PSOLS-10	2.15e+01	1.75e+09	4.91e+01	4.85e+03	2.41e+10	2.40e+03	6.12e+09	7.53e+01	3.85e+04	5.39e+05	4.98e+02	9.13e+01
	PSOLS-20	2.14e+01	1.66e+09	4.90e+01	4.74e+03	2.21e+10	2.36e+03	5.71e+09	7.39e+01	3.81e+04	5.27e+05	4.90e+02	9.10e+01
	EPUSLS	2.15e+01	2.15e+09	4.96e+01	5.23e+03	3.09e+10	2.51e+03	7.56e+09	8.07e+01	3.91e+04	5.81e+05	5.16e+02	9.21e+01
	RLS	2.15e+01	2.21e+09	4.97e+01	5.15e+03	3.01e+10	2.52e+03	7.39e+09	8.02e+01	3.91e+04	5.72e+05	5.14e+02	9.20e+01
	IPSOLS	2.15e+01	2.21e+09	4.97e+01	5.15e+03	3.01e+10	2.52e+03	7.39e+09	8.02e+01	3.91e+04	5.72e+05	5.14e+02	9.20e+01
10^3	PSO-10^1	2.11e+01	2.96e+08	4.78e+01	1.12e+03	**1.06e+09**	1.38e+03	**4.59e+08**	**4.05e+01**	2.63e+04	1.24e+05	1.87e+02	6.77e+01
	PSO-10^2	2.11e+01	5.76e+08	4.80e+01	2.12e+03	3.55e+09	1.66e+03	1.27e+09	5.02e+01	3.44e+04	2.35e+05	2.52e+02	8.37e+01
	PSO-10^3	2.13e+01	1.28e+09	4.80e+01	3.99e+03	1.51e+10	2.12e+03	4.08e+09	6.76e+01	3.56e+04	4.43e+05	4.55e+02	8.71e+01
	EPUS	2.12e+01	4.27e+08	4.80e+01	1.58e+03	2.01e+09	1.61e+03	7.62e+08	4.91e+01	3.63e+04	1.70e+05	4.82e+02	7.99e+01
	IPSO	2.10e+01	3.01e+08	4.79e+01	1.22e+03	1.13e+09	1.53e+03	5.07e+08	4.09e+01	3.04e+04	1.35e+05	**1.86e+02**	7.43e+01
	PSOLS-5	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	2.99e+09	7.75e+02	1.37e+09	7.60e+01	**1.38e+04**	**0.00e+00**	2.44e+02	2.22e+01
	PSOLS-10	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	3.15e+09	7.86e+02	1.41e+09	7.46e+01	1.39e+04	**0.00e+00**	2.45e+02	2.25e+01
	PSOLS-20	2.05e+01	1.41e-16	4.63e+01	9.69e-01	3.52e+09	8.15e+02	1.50e+09	7.36e+01	1.40e+04	**0.00e+00**	2.49e+02	2.32e+01
	EPUSLS	**2.04e+01**	5.06e-16	4.63e+01	**9.56e-01**	2.27e+09	**7.56e+02**	1.12e+09	7.74e+01	1.39e+04	2.94e-22	2.44e+02	**2.18e+01**
	RLS	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	2.93e+09	7.65e+02	1.36e+09	7.75e+01	**1.38e+04**	**0.00e+00**	2.44e+02	**2.18e+01**
	IPSOLS	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	2.93e+09	7.65e+02	1.36e+09	7.75e+01	**1.38e+04**	**0.00e+00**	2.44e+02	**2.18e+01**
10^4	PSO-10^1	2.01e+01	8.37e+07	4.67e+01	3.15e+02	4.69e+07	9.19e+02	4.37e+07	2.74e+01	1.97e+04	3.49e+04	1.87e+02	5.10e+01
	PSO-10^2	1.72e+01	4.95e+07	4.70e+01	1.68e+02	1.62e+07	9.67e+02	2.20e+07	1.79e+01	2.07e+04	1.86e+04	7.70e+01	5.60e+01
	PSO-10^3	2.09e+01	3.70e+08	4.74e+01	1.40e+03	1.39e+09	1.47e+03	6.16e+08	4.13e+01	3.25e+04	1.56e+05	1.76e+02	8.04e+01
	EPUS	2.06e+01	4.89e+06	4.72e+01	1.23e+01	3.48e+04	1.06e+03	3.14e+05	3.05e+01	2.43e+04	1.23e+03	4.62e+02	5.16e+01
	IPSO	2.00e+01	2.51e+07	4.71e+01	9.89e+01	5.34e+06	9.81e+02	7.78e+06	**1.54e+01**	2.10e+04	1.09e+04	8.80e+01	4.92e+01
	PSOLS-5	1.99e+01	**1.08e-17**	2.53e+01	**2.27e-01**	**5.59e-25**	3.31e+02	2.49e+02	4.03e+01	6.40e+03	**0.00e+00**	7.30e+01	3.04e-01
	PSOLS-10	1.99e+01	**1.08e-17**	2.53e+01	**2.27e-01**	6.17e-25	3.31e+02	2.49e+02	4.03e+01	6.40e+03	**0.00e+00**	7.30e+01	3.04e-01
	PSOLS-20	1.99e+01	**1.08e-17**	2.54e+01	**2.27e-01**	7.93e-25	3.31e+02	2.49e+02	4.03e+01	6.40e+03	**0.00e+00**	7.30e+01	3.04e-01
	EPUSLS	1.99e+01	9.68e-17	**2.48e+01**	2.33e-01	1.17e-19	3.33e+02	**2.26e+02**	4.00e+01	6.43e+03	1.59e-22	**7.20e+01**	**2.53e-01**
	RLS	1.99e+01	2.49e-17	2.53e+01	2.33e-01	7.56e-22	3.34e+02	2.49e+02	4.03e+01	6.43e+03	**0.00e+00**	7.30e+01	3.04e-01
	IPSOLS	**1.22e+01**	2.49e-17	2.53e+01	2.31e-01	1.33e-21	**2.95e+02**	2.49e+02	4.03e+01	**5.90e+03**	**0.00e+00**	**7.20e+01**	3.04e-01
10^5	PSO-10^1	1.99e+01	3.84e+07	4.50e+01	1.04e+02	7.95e+06	8.36e+02	7.43e+06	2.63e+01	1.80e+04	1.18e+04	1.87e+02	5.10e+01
	PSO-10^2	1.03e+01	2.12e+05	4.50e+01	1.07e+00	4.90e+00	4.57e+02	1.10e+03	3.75e+00	1.35e+04	1.31e+01	7.70e+01	2.55e+01
	PSO-10^3	1.00e+01	9.08e+06	4.62e+01	3.04e+01	1.95e+05	6.86e+02	1.50e+06	9.61e+00	1.60e+04	3.27e+03	4.40e+01	4.27e+01
	EPUS	1.98e+01	1.46e-02	4.57e+01	3.32e-02	8.79e-01	7.96e+02	2.43e+02	9.50e+00	1.82e+04	8.57e-07	4.52e+02	4.11e+01
	IPSO	4.19e+00	8.31e+04	4.56e+01	1.13e+00	1.23e+01	5.73e+02	2.89e+03	**3.13e+00**	1.55e+04	1.37e+01	1.10e+01	2.45e+01
	PSOLS-5	1.99e+01	9.52e-18	1.06e+01	**0.00e+00**	2.20e-28	3.06e+02	1.56e+01	8.90e+00	5.90e+03	**0.00e+00**	2.10e+01	8.64e-02
	PSOLS-10	1.98e+01	7.11e-18	1.54e+01	**0.00e+00**	2.90e-30	2.94e+02	6.64e+01	2.13e+01	5.70e+03	**0.00e+00**	6.70e+01	6.98e-02
	PSOLS-20	9.81e+00	**5.91e-18**	1.54e+01	**0.00e+00**	**1.40e-31**	2.84e+02	6.64e+01	2.13e+01	5.47e+03	**0.00e+00**	6.70e+01	6.98e-02
	EPUSLS	1.98e+01	9.68e-17	**6.24e+00**	2.32e-01	1.17e-19	3.33e+02	**1.27e+00**	6.60e+00	6.43e+03	1.59e-22	7.00e+01	1.32e-01
	RLS	2.76e+00	2.49e-17	1.52e+01	**0.00e+00**	7.56e-22	2.81e+02	6.50e+01	2.14e+01	5.45e+03	**0.00e+00**	6.65e+01	**6.69e-02**
	IPSOLS	**8.54e-10**	2.49e-17	6.68e+00	**0.00e+00**	8.27e-22	**1.99e+02**	1.78e+00	6.00e+00	**2.81e+03**	**0.00e+00**	**9.00e+00**	7.27e-02
10^6	PSO-10^1	1.99e+01	3.84e+07	4.43e+01	1.04e+02	7.95e+06	8.36e+02	7.35e+06	2.63e+01	1.80e+04	1.18e+04	1.87e+02	5.10e+01
	PSO-10^2	1.02e+01	9.99e-16	4.17e+01	5.79e-01	2.77e-01	4.56e+02	1.64e+02	3.30e+00	1.35e+04	1.00e-15	7.70e+01	2.55e+01
	PSO-10^3	2.13e+00	9.85e-03	4.31e+01	2.46e-02	4.50e-01	3.12e+02	1.98e+02	1.20e+00	1.06e+04	3.91e-07	4.40e+01	1.65e+01
	EPUS	1.98e+01	5.92e-16	4.34e+01	2.95e-02	7.97e-16	6.83e+02	2.55e+01	1.00e+00	1.65e+04	4.98e-16	4.30e+02	3.99e+01
	IPSO	1.60e+00	5.64e-07	4.32e+01	7.40e-03	3.20e-02	2.99e+02	1.86e+02	**8.00e-01**	1.06e+04	2.82e-11	**0.00e+00**	1.43e+01
	PSOLS-5	1.99e+01	9.52e-18	4.94e+00	**0.00e+00**	2.20e-28	3.06e+02	1.68e-01	3.40e+00	5.90e+03	**0.00e+00**	**0.00e+00**	8.64e-02
	PSOLS-10	1.98e+01	7.11e-18	4.14e+00	**0.00e+00**	2.90e-30	2.94e+02	6.31e-02	2.45e+00	5.70e+03	**0.00e+00**	**0.00e+00**	6.50e-02
	PSOLS-20	9.81e+00	**5.91e-18**	3.37e+00	**0.00e+00**	**1.40e-31**	2.83e+02	2.97e-02	4.10e+00	5.47e+03	**0.00e+00**	4.00e+00	1.45e-02
	EPUSLS	6.89e-10	9.35e-17	3.41e+00	6.41e-16	1.17e-19	3.18e+02	**1.37e-04**	2.90e+00	6.36e+03	1.59e-22	5.90e+01	1.24e-01
	RLS	8.92e-10	2.49e-17	1.27e+01	**0.00e+00**	7.56e-22	2.47e+02	1.28e+01	1.78e+01	4.77e+03	**0.00e+00**	6.20e+01	4.64e-04
	IPSOLS	**5.91e-10**	2.49e-17	**3.02e+00**	**0.00e+00**	8.27e-22	**1.53e+02**	8.89e-03	4.70e+00	**5.92e+02**	**0.00e+00**	**0.00e+00**	**1.78e-04**

[1] Results of PSO-based algorithms obtained with a fully-connected topology. The lowest median objective function values are highlighted in boldface.

Table 4.4: Number of Times IPSO Performs Either Better or no Worse[1] than Other PSO-based Algorithms at Different Run Lengths[2]

Evaluations	PSO-10^1	PSO-10^2	PSO-10^3	EPUS
10^2	17 (5)	22 (2)	22 (2)	22 (2)
10^3	1 (5)	23 (1)	24 (0)	19 (2)
10^4	10 (2)	17 (7)	24 (0)	10 (2)
10^5	14 (0)	3 (7)	22 (2)	8 (2)
10^6	23 (0)	10 (5)	19 (5)	9 (3)

[1] No worse cases shown in parenthesis.
[2] 12 problems × 2 topologies = 24 cases.

cases for runs of up to 10^3 function evaluations; however, the best results are distributed among all the tested algorithms. For 10^4 or more function evaluations, algorithms that use local search find the best solutions (except for the Salomon function). IPSOLS finds at least the same number of best solutions as the other local search-based algorithms. For runs of up to one million function evaluations, IPSOLS finds 8 out of the 12 possible best median solutions.

Data from Figures 4.4 and 4.5, and Table 4.3 suggest that in contrast with constant population size PSO algorithms, the performance of EPUS and IPSO does not greatly depend on the duration of a run. A strong point in favor of PSO algorithms that vary the population size over time is that both EPUS and IPSO compete with the best constant population size PSO algorithm at different run durations. However, the mechanism used for varying the size of the population does have an impact on performance as demonstrated in Table 4.4, which shows the number of times IPSO performs at least as well (in a statistical sense) as other PSO-based algorithms at different run durations. In total, 24 cases are considered, which are the result of summarizing the results obtained on the 12 benchmark functions using both the fully connected and ring topologies. Also in Table 4.4, one can see that IPSO dominates at least two of the constant population size PSO algorithms at different run durations. For runs of up to 10^5 function evaluations, the constant population size PSO algorithms with 100 and 1000 particles are dominated by IPSO. For longer runs, IPSO dominates the algorithms with 10 and 1000 particles. The data shown in Table 4.4 demonstrate that the performance of IPSO follows closely the performance of the best constant population size PSO algorithm for a certain run-length. Regarding the difference in performance due to differences in the mechanism for varying the population size, IPSO dominates EPUS for short runs. For long runs, IPSO performs better or not worse than EPUS in half of the cases.

Use vs. No Use of Local Search

The local search component plays a major role in the performance of the algorithms that include it. Table 4.3 and Figures 4.4 and 4.5 show that for runs of at least 10^3 function evaluations, the quality of the solutions obtained with the algorithms that include a local search procedure is typically higher than the quality of the solutions obtained with the algorithms that do not. The only case in which an algorithm without a local search component (IPSO) dominates is when solving the Salomon function. Speed is also affected by the use of a local search component. Table 4.5 lists the first, second, and third quartiles of the algorithms' run-length distributions (Hoos and Stützle, 2004). A hyphen in an entry indicates that the target objective function value was not reached within the 10^6 function evaluations allocated for the experiments. Therefore, if there is a hyphen in a third quartile entry, least 25% of the runs did not find the target objective function value. A similar reasoning applies if there is a hyphen in a first or second quartile entry. The data in Table 4.5 show that the algorithms that combine a variable population size with a local search component (EPUSLS and IPSOLS) are the fastest and most reliable of the studied algorithms. EPUSLS and IPSOLS together are the fastest algorithms for 9 out of the 12 considered functions.

Table 4.5: First, second, and third quartiles of the number of function evaluations needed to find a target solution value[1]

Algorithm	Quartile	Benchmark function (Target value)											
		Ackley (10)	Axis-parallel Hyper-ellipsoid (0.01)	Extended Schaffer (10)	Griewank (0.01)	Penalized (0.001)	Rastrigin (1000)	Rosenbrock (0.01)	Salomon (10)	Schwefel (10000)	Sphere (0.01)	Step (10)	Weierstrass (0.01)
	3rd	-	-	-	-	-	1.64e+04	-	-	-	-	-	-
PSO-10^1	2nd	-	-	-	-	-	4.41e+03	-	-	-	-	-	-
	1st	-	-	-	-	-	2.81e+03	-	-	-	-	-	-
	3rd	-	-	-	-	-	1.05e+04	-	3.09e+04	-	-	-	-
PSO-10^2	2nd	-	3.65e+05	-	-	-	9.26e+03	-	2.61e+04	-	2.04e+05	-	-
	1st	3.02e+04	2.84e+05	-	-	-	8.01e+03	-	2.11e+04	-	1.65e+05	-	-
	3rd	1.28e+05	-	-	-	-	6.00e+04	-	1.04e+05	-	7.04e+05	-	-
PSO-10^3	2nd	9.94e+04	-	-	-	-	5.44e+04	-	9.46e+04	-	6.21e+05	-	-
	1st	8.56e+04	9.16e+05	-	5.93e+05	-	4.74e+04	-	8.83e+04	4.23e+05	5.71e+05	-	-
	3rd	-	1.10e+05	-	-	-	2.18e+04	-	1.41e+05	-	5.49e+04	-	-
EPUS	2nd	-	1.02e+05	-	-	1.97e+05	1.33e+04	-	9.48e+04	-	5.15e+04	-	-
	1st	-	9.57e+04	-	5.38e+04	1.23e+05	8.45e+03	-	6.10e+04	-	4.75e+04	-	-
	3rd	-	7.10e+05	-	-	-	1.25e+04	-	**2.21e+04**	-	3.09e+05	2.26e+05	-
IPSO	2nd	3.41e+04	6.24e+05	-	3.44e+05	-	9.53e+03	-	**2.00e+04**	-	2.82e+05	1.53e+05	-
	1st	2.16e+04	5.68e+05	-	2.70e+05	9.64e+05	7.25e+03	-	**1.87e+04**	6.47e+05	2.65e+05	1.03e+05	-
	3rd	-	1.39e+03	1.34e+05	**2.61e+04**	**1.23e+03**	9.37e+02	-	9.81e+04	1.70e+03	8.62e+02	1.98e+05	-
PSOLS-5	2nd	-	9.38e+02	1.12e+05	**1.25e+04**	**1.12e+03**	8.93e+02	-	9.49e+04	1.60e+03	8.52e+02	1.59e+05	-
	1st	-	8.51e+02	9.54e+04	**6.89e+03**	**1.09e+03**	8.47e+02	-	9.24e+04	1.46e+03	8.48e+02	1.42e+05	-
	3rd	-	1.39e+03	2.15e+05	**2.66e+04**	1.23e+03	9.42e+02	-	1.78e+05	1.71e+03	8.67e+02	3.68e+05	-
PSOLS-10	2nd	-	9.44e+02	1.92e+05	**1.25e+04**	1.13e+03	8.98e+02	-	1.75e+05	1.60e+03	8.57e+02	2.92e+05	-
	1st	2.65e+04	8.56e+02	1.76e+05	**6.90e+03**	1.10e+03	8.52e+02	4.33e+05	1.72e+05	1.47e+03	8.53e+02	2.51e+05	-
	3rd	-	1.40e+03	3.76e+05	**2.66e+04**	1.24e+03	9.52e+02	-	3.39e+05	1.72e+03	8.77e+02	6.68e+05	-
PSOLS-20	2nd	-	9.54e+02	3.53e+05	**1.25e+04**	1.14e+03	9.08e+02	-	3.34e+05	1.61e+03	8.67e+02	5.80e+05	-
	1st	2.65e+04	8.66e+02	3.38e+05	**6.91e+03**	1.11e+03	8.62e+02	7.16e+05	3.31e+05	1.48e+03	8.63e+02	5.10e+05	1.85e+05
	3rd	-	**1.39e+03**	**4.75e+04**	-	**1.22e+03**	**9.10e+02**	**8.70e+05**	4.31e+04	1.69e+03	8.59e+02	-	-
EPUSLS	2nd	3.04e+05	**8.60e+02**	**3.44e+04**	1.41e+05	**1.12e+03**	**8.54e+02**	**4.87e+05**	3.21e+04	1.61e+03	8.49e+02	-	-
	1st	3.92e+04	**8.48e+02**	**2.99e+04**	6.66e+03	**1.10e+03**	**8.22e+02**	**3.20e+05**	2.86e+04	1.48e+03	8.45e+02	-	-
	3rd	1.76e+05	1.38e+03	-	3.08e+04	**1.22e+03**	9.33e+02	-	-	**1.70e+03**	**8.58e+02**	-	**8.58e+05**
RLS	2nd	7.01e+04	9.34e+02	-	1.40e+04	**1.12e+03**	8.89e+02	-	-	**1.59e+03**	**8.48e+02**	-	**3.80e+05**
	1st	3.36e+04	8.47e+02	-	6.91e+03	**1.09e+03**	8.43e+02	-	-	**1.46e+03**	**8.44e+02**	-	**1.37e+05**
	3rd	**1.42e+04**	1.38e+03	6.25e+04	2.46e+04	**1.22e+03**	9.33e+02	-	3.41e+04	**1.70e+03**	**8.58e+02**	**1.02e+05**	6.88e+05
IPSOLS	2nd	**1.03e+04**	9.34e+02	3.81e+04	1.36e+04	**1.12e+03**	8.89e+02	8.96e+05	3.10e+04	**1.59e+03**	**8.48e+02**	**8.87e+04**	4.11e+05
	1st	**7.50e+03**	8.47e+02	3.11e+04	6.82e+03	**1.09e+03**	8.43e+02	3.82e+05	2.86e+04	**1.46e+03**	**8.44e+02**	**4.30e+04**	1.83e+05

[1] Results of PSO-based algorithms obtained with a fully-connected topology. The target value for each function is indicated under its name in parentheses. The results with the lowest median number of function evaluations are highlighted in boldface.

Table 4.6: Number of Times IPSOLS Performs Either Better or no Worse[1] than Other PSO-Local Search Hybrids at Different Run Lengths[2]

Evaluations	PSOLS-5	PSOLS-10	PSOLS-20	EPUSLS	RLS
10^2	0 (3)	0 (2)	0 (2)	0 (24)	0 (23)
10^3	15 (6)	15 (6)	15 (6)	0 (23)	0 (23)
10^4	14 (5)	15 (4)	15 (4)	12 (11)	8 (15)
10^5	15 (4)	14 (5)	14 (5)	19 (4)	14 (9)
10^6	12 (5)	10 (7)	12 (5)	18 (2)	14 (9)

[1] No worse cases shown in parenthesis.
[2] 12 problems × 2 topologies = 24 cases.

In terms of the quality of the solutions found by the local search-based algorithms, IPSOLS outperforms EPUSLS, as seen in Tables 4.3 and 4.6. Table 4.6 shows the number of times IPSOLS performs either better or no worse (in a statistical sense) than other PSO-local search hybrids at different run durations.

The difference in performance between the constant population size algorithms that we observed when they do not use local search, almost disappears when local search is used. For runs of some hundreds of function evaluations, IPSOLS performs no better than any other hybrid PSO–local search algorithms (see first row in Table 4.6). These results occur for two reasons: (i) the Powell's conjugate directions set method has to perform at least n line searches (n is the number of dimensions of the problem) before significantly improving the quality of the solutions found, and (ii) PSOLS first explores and then invokes the local search component. However, for longer runs, IPSOLS clearly dominates all other hybrid algorithms, including EPUSLS.

IPSOLS is an algorithm that repeatedly calls a local search procedure from different initial solutions. In this respect, IPSOLS is similar to a random restart local search algorithm (RLS). However, the difference between RLS and IPSOLS is that in RLS, the new initial solutions are chosen at random, while in IPSOLS the new initial solutions are the result of the application of the PSO rules. Thus, a comparison of the results obtained with these two algorithms can give us an indication of the impact of the PSO component in IPSOLS. The results presented in Figure 4.5 indicate that IPSOLS outperforms RLS in all problems except on Axis-parallel Hyper-ellipsoid, Griewank, Penalized, Sphere and Weierstrass. In the case of the Sphere function, the local search procedure alone is able to find the optimum (with an objective function value that is lower than or equal to 10^{-15}, one of our stopping criteria). In the case of the Griewank function, IPSOLS solves the problem with a population of around 3 particles — data shown in Montes de Oca (2011). Thus, IPSOLS's behavior is similar to that of RLS when its population does not grow significantly (see Figure 4.6).

In Figure 4.6, we show two examples of the compared algorithms' behavior over time. These examples correspond to the solution development over the number of function evaluations obtained by a selection of the compared algorithms on the Rastrigin and Sphere functions. These functions are chosen so that the behavior of the algorithms on unimodal (Sphere) and multimodal (Rastrigin) functions can be compared. In these figures, we also show the average population size growth over time in IPSO, EPUS, EPUSLS and IPSOLS.

In some cases, as noted before, IPSOLS is outperformed by other algorithms for short runs (in our case, runs between 10^2 and 10^3 function evaluations). However, IPSOLS improves dramatically once the population size starts growing, as exemplified in Figure 4.6(d) in which IPSOLS starts differentiating from RLS, EPUSLS and PSOLS after approximately 5,000 function evaluations. IPSOLS improves rapidly once the local search procedure begins to make progress, as seen in Figure 4.6(c). When IPSOLS and EPUSLS are applied to the Sphere function, the population size does not grow (see Figure 4.6(e)). As a result, IPSOLS, EPUSLS, and RLS are equivalent on problems solvable by local search alone. In most cases, the population growth in IPSOLS is independent of the population topology used — data shown in Montes de Oca (2011).

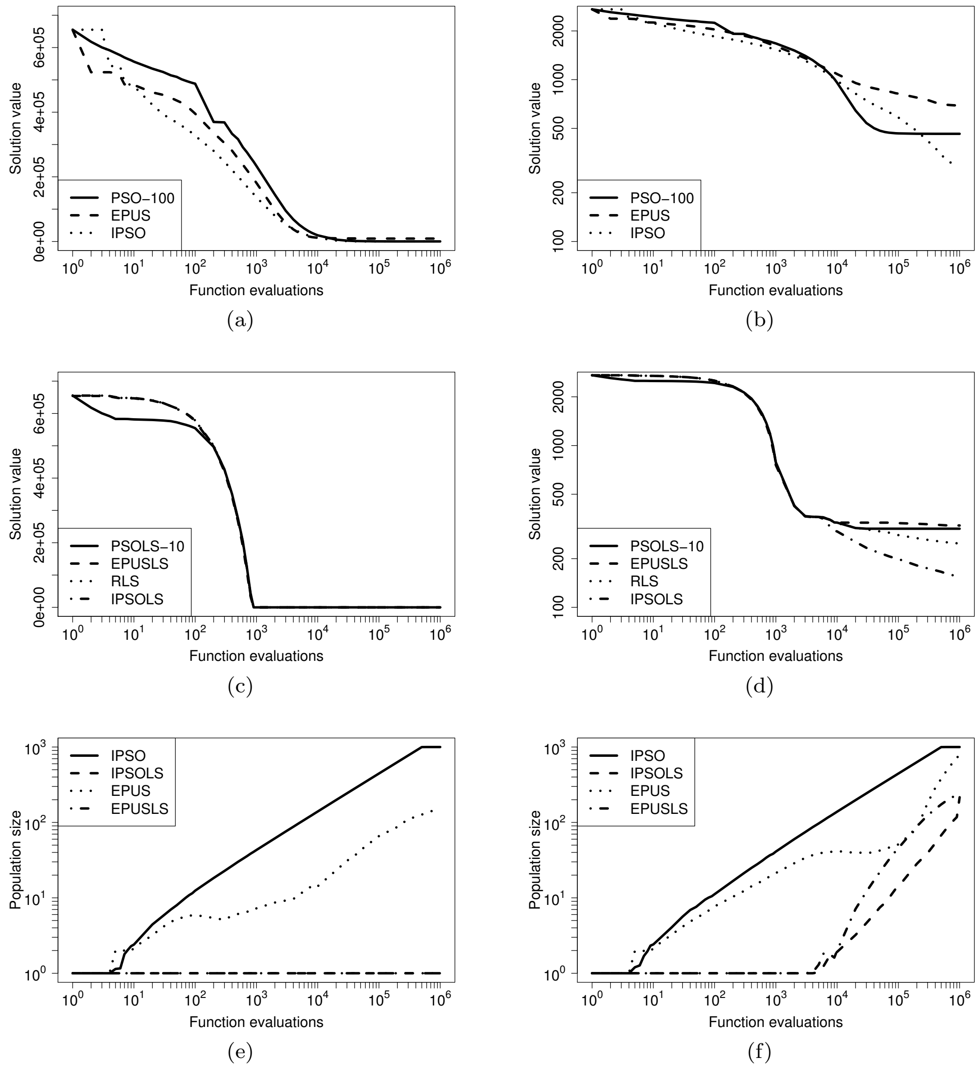


Figure 4.6: Solution development over time obtained by a selection of the compared algorithms (PSO-based algorithms using a fully-connected topology) on the Sphere and Rastrigin functions. Figures (a) and (b) show the results without local search. Figures (c) and (d) show the results with local search. Figures (e) and (f) show the average population size growth in IPSO, EPUS, EPUSLS and IPSOLS.

An exception in the conclusions of the analysis of the results has been the Salomon function case. This function can be thought of as a multidimensional wave that is symmetric in all directions with respect to the optimum. We believe that the poor performance of all the tested local search-based algorithms is due to the undulatory nature of this function. When the local search is invoked in the proximity of the global optimum, valleys that are far away from it can actually attract the local search method. As a result, the global optimization algorithm that calls the local search method is "deceived." This phenomenon seems to be exacerbated when Powell's method is applied in high dimensions.

From a practitioner's point of view, there are at least two advantages to using IPSOLS instead of a hybrid PSO algorithm: (i) IPSOLS does not require the practitioner to fix the population size in advance, hoping to have chosen the right size for his/her problem, and (ii) IPSOLS is more robust to the choice of the population's topology. The difference

Table 4.7: Amplitudes used in the Rastrigin function to obtain specific fitness distance correlations (FDCs)

Amplitude	FDC	Amplitude	FDC
155.9111	≈ 0.001	13.56625	≈ 0.6
64.56054	≈ 0.1	10.60171	≈ 0.7
40.09456	≈ 0.2	7.938842	≈ 0.8
28.56419	≈ 0.3	5.21887	≈ 0.9
21.67512	≈ 0.4	0.0	≈ 0.999
16.95023	≈ 0.5	-	-

between the results obtained through IPSOLS with a fully-connected topology and with a ring topology are smaller than the differences observed in the results obtained through the hybrid algorithms — data shown in Montes de Oca (2011).

4.4.3 Is Social Learning Necessary?

The ISL framework calls for social learning when a new agent is added to the population. However, is social learning really necessary in the context of IPSO and IPSOLS? Would the performance of these algorithms be the same if new particles were simply initialized at random? In this section, we present the results of an experiment aimed at measuring the effects of the social learning rule (Eq. 4.2) on the performance of IPSO and IPSOLS.

We measure the extent to which the initialization rule applied to new particles affects the quality of the solution obtained after a certain number of function evaluations with respect to a random initialization. For this purpose, IPSO and IPSOLS are run with initialization mechanisms that induce a bias of different strength toward the best particle of the swarm. These mechanisms are (in increasing order of bias strength): (i) random initialization, (ii) the initialization rule as defined in Eq. 4.2 (labeled as "weak bias"), and (iii) the same rule as defined in Eq. 4.2, but with the random number U drawn from a uniform distribution in the range $[0.95, 1)$ (labeled as "strong bias").

The experiments are carried out on problems derived from the Rastrigin function. Each derived problem has different fitness distance correlation (FDC) (Jones and Forrest, 1995). To compute a problem's FDC, a set of sample solutions are generated. For each sample, its objective function value (called "fitness" in the evolutionary computation field where the measure originated) and its distance to a reference point are computed. The reference point can be the known optimum or the best known solution to the problem. A problem's FDC is simply the correlation between the objective function values of the samples and their distance to the reference point. If a problem's FDC is large and positive, an algorithm that searches in the vicinity of the best-so-far-solution is expected to perform well. If the problem's FDC is low or negative, the problem becomes much harder because the best-so-far solution does not give much information about which regions of the search space are promising.

Since the social learning rule used in IPSO and IPSOLS implicitly assumes that good solutions are close to each other, the hypothesis is that the performance of the algorithms degrades as the problem's FDC approaches zero. Additionally, one hypothesizes that the rate of performance degradation is faster with stronger initialization bias.

The Rastrigin function, whose n-dimensional formulation is $nA+\sum_{i=1}^{n}\left(x_i^2 - A\cos(\omega x_i)\right)$, can be thought of as a parabola with a superimposed (co)sine wave with an amplitude and frequency controlled by parameters A and ω respectively. By changing the values of A and ω one can obtain a whole family of problems. In our experiments, we set $\omega = 2\pi$, as is usually done, and tuned the amplitude A to obtain functions with specific FDCs. The search range and the dimensionality of the problem are set to $[-5.12, 5.12]^n$ and $n = 100$, respectively. The amplitude and the resulting FDCs (estimated using 10^4 uniformly distributed random samples over the search range) are shown in Table 4.7.

IPSO and IPSOLS with the three initialization rules described above were run 100 times on each problem for up to 10^6 function evaluations. To measure the magnitude of the effect of using one or another initialization scheme, we use Cohen's d statistic (Cohen, 1988), which for the case of two samples is defined as follows:

$$d = \frac{\hat{\mu_1} - \hat{\mu_2}}{\sigma_{pooled}}\,, \tag{4.12}$$

with

$$\sigma_{pooled} = \ sqrt\frac{(n_1 - 1)\hat{\sigma_1}^2 + (n_2 - 1)\hat{\sigma_2}^2}{n_1 + n_2 - 2}\,, \tag{4.13}$$

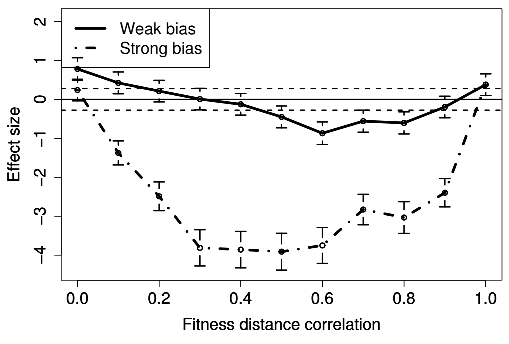
where $\hat{\mu_i}$ and $\hat{\sigma_i}$ are the mean and standard deviation of sample i, respectively (Nakagawa and Cuthill, 2007).

As an effect size index, Cohen's d statistic measures the difference between the mean responses of a treatment and a control group expressed in standard deviation units (Sheskin, 2000). The treatment group is, in our case, the set of solutions obtained with IPSO and IPSOLS using the initialization rule that biases the position of a new particle toward the best particle of the swarm. The control group is the set of solutions obtained with IPSO and IPSOLS when new particles are initialized completely at random. (Since in our case the lower the solution value the better, the order of the operands in the subtraction is reversed.) An effect size value of 0.8, for example, means that the average solution found using the particles' initialization rule is better than 79% of the solutions found without using it. The practical significance that the value associated with an effect has depends, of course, on the situation under consideration; however, Cohen (1988) states that a value of 0.8 can already be considered a large effect.

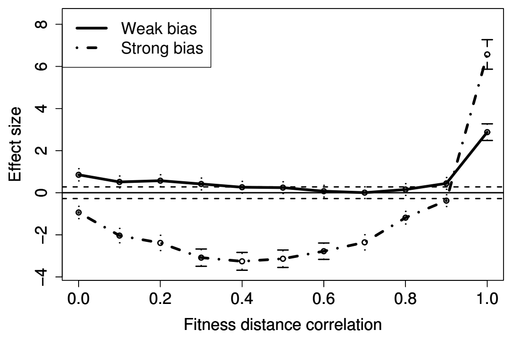
The observed effect sizes with 95% confidence intervals on the solution quality obtained with IPSO and IPSOLS after 10^6 function evaluations are shown in Figure 4.7.

In IPSO, the effects of using the new particles initialization rule are very different from the ones in IPSOLS. In IPSO, the weak bias initialization rule produces better results than random initialization only in two cases: (i) when the problem's FDC is almost equal to one and the algorithm is run with a ring topology, and (ii) when the problem's FDC is close to zero, irrespective of the population topology used. In all other cases, the weak bias initialization rule produces results similar to those obtained with a random initialization. The strong bias initialization rule reports benefits only in the case of a high FDC and a ring topology. In all other cases, the strong bias initialization rule produces results that are significantly worse than the results obtained with a random initialization. The worst performance of IPSO with the strong bias initialization rule is obtained when the problem's FDC is in the range (0.3,0.6). This behavior is a consequence of the new particle's velocity being equal to zero, which effectively reduces the particle's initial exploratory behavior. Setting the new particle's initial velocity to a value different from zero reduces the effect of the initialization bias because it would immediately make the particle move to a quasi-random position right after the first iteration of the algorithm's PSO component. The performance observed when the problem's FDC is close to zero is the result of the fact that with a fixed search range and a high amplitude, the parabolic component of the Rastrigin function has a much lower influence and many of the locally optimal solutions are of the same quality. Thus, moving close to or away from already good solutions has no major impact on the solution quality.

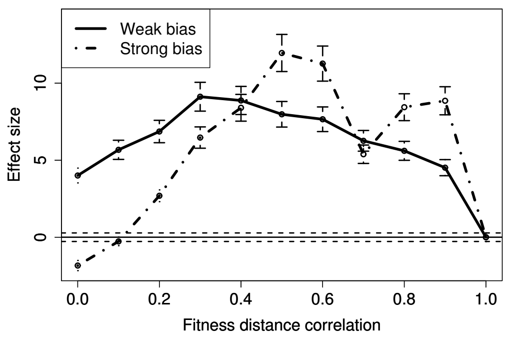
While the effect in IPSO is positive only in a few cases, in IPSOLS, the effect size is not only positive in almost all cases but it is also large. IPSOLS with the weak bias initialization rule produces significantly better solutions than with a random initialization in all but one case. This case corresponds to the situation where the problem's FDC is close to one. When the strong bias initialization rule is used, IPSOLS produces better solutions than with random initialization when the problem's FDC is in the range (0.1, 1.0). In the range (0.4,1.0), the solutions obtained with a strong bias initialization rule are better than or equal to those obtained with a weak bias initialization rule.



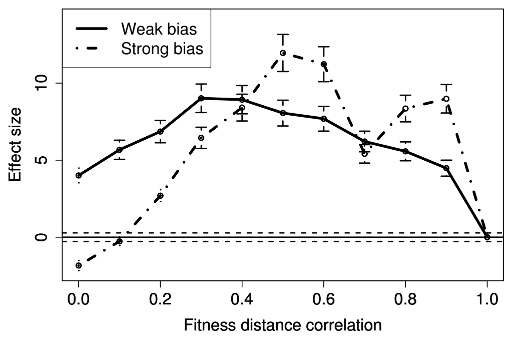
(a) IPSO, Fully-connected topology



(b) IPSO, Ring topology



(c) IPSOLS, Fully-connected topology



(d) IPSOLS, Ring topology

Figure 4.7: Effect size of the new particles initialization rule, as measured using Cohen's d statistic with 95% confidence intervals (indicated by either error bars or dashed lines), on the solution quality obtained with IPSO and IPSOLS after 10^6 function evaluations. Two bias strengths are tested: (i) weak bias and (ii) strong bias. The reference results (line at zero) are obtained with a random initialization.

In IPSOLS, when the problem's FDC is almost one, initializing a new particle completely at random, or with a bias toward the location of the best particle of the swarm, is effectively the same. This phenomenon can be explained if we recall that IPSOLS is a local search algorithm that starts from a single solution. Since a local search algorithm alone can solve a problem with an FDC close to one, no population growth occurs, and the initialization rule is never used. Thus, it does not matter how new particles are initialized. The degradation of the effect size as the problem's FDC decreases can be observed in the range (0.0,0.5) for the strong bias initialization rule and in the range (0.0, 0.3) for the weak bias initialization rule. As hypothesized, the rate of degradation is faster when using a strong bias.

In summary, the use of the weak bias initialization rule in IPSOLS, which is the originally proposed social learning rule, provides significant benefits over random initialization on the family of problems we examined with a fitness distance correlation in the range (0,1).

4.5 IPSOLS+: A Redesigned IPSOLS

In this section, we present a variant of IPSOLS, called IPSOLS+, that is the result of a redesign process based on an automatic parameter tuning system. The motivation for this approach arose from the fact that the once art of algorithm design, has turned into an engineering task (Stützle et al., 2007, 2009). In part, this transition has happened thanks to the recent availability of automatic parameter tuning systems, which make the semi-automated design of high-performing optimization algorithms possible. Nowadays, optimization algorithms can be seen as entities made of a number of components that a designer integrates with one another in order to tackle an optimization problem. However, despite the progress hitherto made, there are still many decisions in the design process that are made by the designer based on intuition or limited information about the interaction between the chosen components and their behavior on the target problem. For example, a designer may choose a mutation operator for a genetic algorithm (Goldberg, 1989) based on information gathered through some limited tests. However, what if the designer's choice of parameter settings during testing is unfortunate? Could one operator make the algorithm have a better average performance than another one once properly tuned?

Clearly, if the designer left all the possibilities open until a final parameter tuning phase, the design problem could be seen as a stochastic optimization problem. First steps in this direction have already been taken with some very promising results (see e.g., (KhudaBukhsh et al., 2009; López-Ibáñez and Stützle, 2010)). In these works, however, the high-level algorithm architecture that defines the component interactions remains fixed and is based on the extensive experience of the authors with the target problems. In this section, we take a step back and explore the integration of automatic tuning as a decision-aid tool *during* the design loop of optimization algorithms. We see two advantages in such a tuning-in-the loop approach: (i) the parameter search space is kept to a reasonable size, and (ii) the result of tuning can give us insight into the interactions between algorithmic components and their behavior on the optimization problem at hand. We describe the whole redesign process of IPSOLS and study the final algorithm's performance scalability as the dimensionality of the problem increases. This scalability study is carried out following the protocol defined by Lozano et al. (2011), which consists in running an algorithm 25 times on each of the 19 benchmark functions listed in Table 4.8 for up to $5000n$ function evaluations, where n is the dimensionality of the function. Algorithms are stopped earlier if the objective function value of the best-so-far solution is lower than 1e$-$14 (we refer to this number as *0-threshold*). When an objective function value lower than the 0-threshold is found, we report 0e+00 instead. This evaluation protocol is used throughout the rest of this chapter.

Some of the benchmark functions shown in Table 4.8 are hybrid, that is, they combine two basic functions (from F_1 to F_{11}). Herrera et al. (2010) describe the combination procedure denoted by the symbol $\oplus$. The parameter m_{ns} is used to control the number of components that are taken from a nonseparable function (functions F_3, F_5, F_9, and F_{10}). A higher m_{ns}, results in a larger number of components that come from a nonseparable function.

For functions F_1 to F_{11} (and some of the hybrid functions, F_{12} to F_{19}), candidate solutions, $\boldsymbol{x}$, are transformed as $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o}$ before evaluation. This transformation shifts the optimal solution from the origin of the coordinate system to $\boldsymbol{o}$, with $\boldsymbol{o} \in [X_{\min}, X_{\max}]^n$. For function F_3, the transformation is $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{o} + \mathbf{1}$.

4.5.1 Tuning Algorithm: Iterated F-Race

During the redesign cycle of IPSOLS, we used iterated F-Race (Balaprakash et al., 2007; Birattari et al., 2010) to perform an ad-hoc tuning of the algorithm's parameters on the complete benchmark function set shown in Table 4.8. F-Race (Birattari et al., 2002; Birattari, 2009) is at the core of iterated F-Race. F-Race is a method used for selecting the best algorithm configuration (a particular setting of numerical and categorical parameters of an algorithm) from of a set of candidate configurations under stochastic evaluations. In F-Race, candidate configurations are evaluated iteratively on a sequence of problem instances. As soon as sufficient statistical evidence is gathered against a candidate configuration, it is discarded from the race. The process continues until only one candidate configuration remains, or until the maximum number of evaluations or instances is reached.

The generation of candidate configurations is independent of F-Race. Iterated F-Race is a method that combines F-Race with a process capable of generating promising can-

Table 4.8: Scalable Benchmark Functions for Large Scale Optimization

Name	Definition	Search Range $[X_{\min}, X_{\max}]^n$
F_1	$\sum_{i=1}^{n} z_i^2$	[-100,100]^n
F_2	$\max_i\{\lvert z_i\rvert, 1 \leq i \leq n\}$	[-100,100]^n
F_3	$\sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2]$	[-100,100]^n
F_4	$10n + \sum_{i=1}^{n} (z_i^2 - 10\cos(2\pi z_i))$	[-5,5]^n
F_5	$\frac{1}{4000} \sum_{i=1}^{n} z_i^2 - \prod_{i=1}^{n} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1$	[-600,600]^n
F_6	$-20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi z_i)} + 20 + e$	[-32,32]^n
F_7	$\sum_{i=1}^{n} \lvert z_i\rvert + \prod_{i=1}^{n} \lvert z_i\rvert$	[-10,10]^n
F_8	$\sum_{i=1}^{n} \left(\sum_{j=1}^{i} z_j\right)^2$	[-65.536,65.536]^n
F_9	$\sum_{i=1}^{n-1} f_{10}(z_i, z_{i+1}) + f_{10}(z_n, z_1)$, where $f_{10}(x, y) = (x^2 + y^2)^{0.25}(\sin^2(50(x^2 + y^2)^{0.1}) + 1)$	[-100,100]^n
F_{10}	$\sum_{i=1}^{n-1} (z_i^2 + 2z_{i+1}^2 - 0.3\cos(3\pi z_i) - 0.4\cos(4\pi z_{i+1}) + 0.7)$	[-15,15]^n
F_{11}	$\sum_{i=1}^{n-1} (z_i^2 + z_{x+1}^2)^{0.25}(\sin^2(50(z_1^2 + z_{i+1}^2)^{0.1}) + 1)$	[-100,100]^n
F_{12}	$F_9 \oplus F_1$, $m_{ns} = 0.25$	[-100,100]^n
F_{13}	$F_9 \oplus F_3$, $m_{ns} = 0.25$	[-100,100]^n
F_{14}	$F_9 \oplus F_4$, $m_{ns} = 0.25$	[-5,5]^n
F_{15}	$F_{10} \oplus F_7$, $m_{ns} = 0.25$	[-10,10]^n
F_{16}	$F_9 \oplus F_1$, $m_{ns} = 0.5$	[-100,100]^n
F_{17}	$F_9 \oplus F_3$, $m_{ns} = 0.75$	[-100,100]^n
F_{18}	$F_9 \oplus F_4$, $m_{ns} = 0.75$	[-5,5]^n
F_{19}	$F_{10} \oplus F_7$, $m_{ns} = 0.75$	[-10,10]^n

didate configurations. Iterated F-Race consists of the steps of configuration generation, selection and refinement iteratively. For numerical parameters, the configuration generation step involves sampling from Gaussian distributions centered at promising solutions. The standard deviations of these Gaussian distributions vary over time in order to focus the search around the best-so-far solutions. For categorical parameters, the configuration generation procedure samples from a discrete distribution that gives the highest probability to the values that are found in the best configurations. The process is described in detail in (Birattari et al., 2010; López-Ibáñez et al., 2011).

Tuning Setup

While tuning, problem instances are usually fed into iterated F-Race as a stream. At each step of F-Race, all surviving candidate configurations are evaluated on one additional problem instance. Once each surviving candidate configuration has been evaluated, a statistical test is applied in order to determine whether there are configurations that have, up to that point, performed significantly worse than others. First, the Friedman test is applied and, if its null hypothesis of equal performance of all surviving candidate configurations is rejected, Friedman post-tests are used to eliminate configurations that perform worse than the best one (Birattari et al., 2002; Birattari, 2009). In our case, we have a limited set of 19 benchmark functions. Additionally, the set of benchmark functions may result in very different performances because they include very easy functions, such as Sphere (F_1) but also much more difficult ones, such as Rastrigin (F_4) and hybrid functions

Table 4.9: Free parameters in IPSOLS (all versions)

Parameter	Range/Domain
w	$[0, 1]$
φ_1	$[0, 4]$
φ_2	$[0, 4]$
Init. pop. size	$[1, 100]$
Particle addition schedule, k	$[1, 10]$
Topology[1]	$\{FC, R\}$
FTol	$[-13, -1]$
ρ_{start}[2]	$[1, 50]$
s[3]	$[\epsilon, 100]$
MaxFES[2]	$[2^2, 2^{10}]$
MaxITER[3]	$[1, 10]$
MaxFailures	$[1, 20]$
MaxStagIter	$[2, 30]$

[1] FC stands for fully connected, R for ring.
[2] Used with BOBYQA (see Section 4.5.2).
[3] Used with Powell's conjugate directions method.

(F_{12}–F_{19}). Therefore, we used a modified version of F-race. We define a *block evaluation* as running all (surviving) candidate configurations once on each benchmark function in a *block* (Chiarandini et al., 2006); a block consists of all 19 benchmark functions. After each block evaluation, the usual statistical tests in F-race are applied, and inferior candidate configurations are eliminated. Throughout a run of iterated F-race, a cumulative average solution value is computed for each surviving solution. This value is used as a rough indicator of their quality, which is in turn used to select the prototypes that serve as guides for the next iteration of iterated F-race.

Tuning was performed on low dimensionality ($n = 10$) versions of the benchmark functions, using the hypothesis that some structure of the problems is maintained across versions of different dimensionality. Testing was carried out on versions of medium size ($n = 100$). Thus, we distinguish between a training set used for tuning and a test set for evaluation. The scalability study of the final design was carried out on the versions of higher dimensionality ($n = \{50, 100, 200, 500, 1000\}$). To have a higher chance of selecting a good candidate configuration, we launched 10 independent runs of the iterated F-Race algorithm and finally selected the one with the lowest average objective function value as the tuned configuration.

To use iterated F-Race, one needs to select the parameters to be tuned, such as their range or domain, and the set of instances on which tuning is performed. The list of free parameters and their corresponding range or domain as used with iterated F-Race is given in Table 4.9. A description of the meaning and effect of these parameters is given in the sections where a variant of IPSOLS is described.

Other parameter settings for IPSOLS, for both tuned and non-tuned versions, remained fixed. A list of them with their settings is shown in Table 4.10.

Table 4.10: Fixed parameters in IPSOLS

Parameter	Value
Max. pop. size	1000
$V_{\max,j}$, $1 \leq j \leq n$	$X_{\max,j}$
Bound constraint handling (PSO part)	Fix on the bound, set velocity to zero
ϵ	*0-threshold* (see Section 4.5.2).

Finally, iterated F-Race itself has a number of parameters that need to be set before it can be used. These parameters and the values used are listed in Table 4.11.

In iterated F-Race, the number of iterations L is equal to $2 + \text{round}(\log_2(d))$, where d is the number of parameters to tune. In iterated F-Race, each iteration has a different

Table 4.11: Iterated F-Race parameter settings

Parameter	Value
Max. Evaluations (B)	5e+04
μ_l	76+l

maximum number of evaluations. This number, denoted by B_l, is equal to $(B - B_{\text{used}})/(L - l + 1)$, where l is the iteration counter, B is the overall maximum number of evaluations, and B_{used} is the number of evaluations used until iteration $l-1$. The number of candidate configurations tested during iteration l is equal to $\lfloor B_l/\mu_l \rfloor$. For more information on the parameters of iterated F-Race and their effect, please consult (Balaprakash et al., 2007; Birattari et al., 2010).

4.5.2 Stage I: Choosing a Local Search Method

In this section, we explore the impact of the local search method on the performance of IPSOLS. As an alternative to Powell's conjugate directions set method, we consider Powell's BOBYQA method (Powell, 2009), which is described below. To ease the naming of intermediate variants of IPSOLS we use the following convention: the variant introduced at stage X, is labeled as IPSOLS-Stage-X.

The acronym BOBYQA stands for *bound constrained optimization by quadratic approximation.* The algorithm that it represents was proposed by Powell (2009). BOBYQA is a derivative-free optimization algorithm based on the trust-region paradigm (Conn et al., 2000). BOBYQA is an extension of the NEWUOA (Powell, 2006) algorithm that is able to deal with bound constraints. At each iteration, BOBYQA computes and minimizes a quadratic model that interpolates m points in the current trust region. These points are automatically generated by the algorithm starting from an initial guess provided by the user. Then, either the best-so-far solution or the trust-region radius is updated. The recommended number of points to compute the quadratic model is $m = 2n+1$ (Powell, 2009), where n is the dimensionality of the search space. We decided to study the performance of IPSOLS with BOBYQA as its local search method because the number of points BOBYQA needs to compute the model is linear with respect to the dimensionality of the search space. In fact, BOBYQA's author proposed it as a method for tackling large-scale optimization problems (Powell, 2009). NEWUOA, and by extension BOBYQA, are considered to be state-of-the-art continuous optimization techniques (More and Wild, 2009; Auger et al., 2009).

We used the implementation that comes with NLopt, a library for nonlinear optimization (Johnson, 2008). The main parameter that controls this method in NLopt is the initial trust-region radius, ρ_{start}. The stopping condition depends on the values assigned to FTol, defined in the same way as in Powell's conjugate directions method, and to MaxFES, which is the maximum number of function evaluations allocated for the method.

Conjugate Directions *vs.* BOBYQA

To measure the performance differences due to the local search methods, we compared the default and tuned versions of IPSOLS-Stage-I with Powell's conjugate directions method and BOBYQA. The default and the best parameter settings found through iterated F-Race are shown in Table 4.12.[3]

For the default configuration of IPSOLS-Stage-I as well as of the subsequent configurations, we use the inertia weight, w, instead of the constriction factor, χ. The two acceleration coefficients, φ_1 and φ_2, were set according to Clerc and Kennedy's analytical analysis of the PSO algorithm (Clerc and Kennedy, 2002). For other parameters, values have been set by the designers of the algorithms based on experience and on commonly

[3]For conciseness, we present here only the most relevant results. The complete set of results can be found in this dissertation's companion website Montes de Oca (2011).

Table 4.12: Default and best configuration obtained through iterated F-Race for IPSOLS-Stage-I in 10-dimensional instances[†]

Method	Configuration	w	φ_1	φ_2	Init. pop. size	k	Topology	FTol	s/ρ_{start}	MaxITER/FES
BOBYQA	Default	0.72	1.49	1.49	1	1	FC	-1	20%	$10(2n+1)$[‡]
	Tuned	0.25	0.0	2.9	50	6	R	-3.4	10%	210
Conjugate Directions	Default	0.72	1.49	1.49	1	1	FC	-1	20%	10
	Tuned	0.0	0.0	1.82	3	6	R	-2.92	15.75%	9

[†] FC stands for fully connected, R for ring. FTol and MaxITER/FES are parameters that determine the stopping condition of the local search method. The conjugate directions method's step size and BOBYQA's trust-region radius is specified as a percentage of the length of the search space in one dimension.

[‡] n is the dimensionality of the problem.

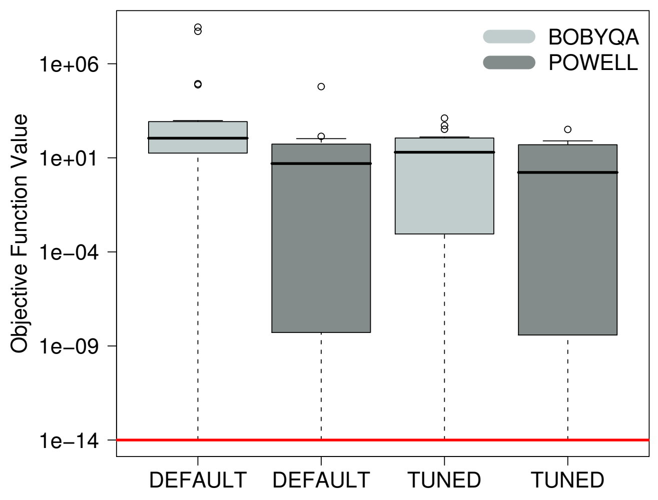


Figure 4.8: Box-plot of the median objective function values obtained by the four configurations listed in Table 4.12. In all cases, we used the 100-dimensional versions of the benchmark functions proposed for this special issue. The line at the bottom represents the 0-threshold.

used parameter settings found in the PSO literature. The range for the parameters ρ_{start} and s is expressed as a percentage of the size of the search range $[x_{\min}, x_{\max}]$ in each dimension. For example, if $x_{\min} = -100$, $x_{\max} = 100$, and $\rho_{\text{start}} = 20$, then the real value that BOBYQA will use as the initial trust-region radius will be $0.2 \cdot 200 = 40$. Before calling either local search method, the value of the parameter FTol is also transformed. The real value sent to the routine is 10^{-Ftol}.

The four configurations shown in Table 4.12 were run on the 100-dimensional version of the 19 benchmark functions suite. In Figure 4.8, we show the distribution of the median objective function value obtained by the resulting four configurations.

The default as well as the tuned configurations of IPSOLS-Stage-I using Powell's conjugate directions method perform better than their BOBYQA-based counterparts. The differences between the two default and two tuned versions were found to be statistically significant at the 5% level using a Wilcoxon test ($p = 0.0008$, and $p = 0.01$, respectively). This result and the best parameter setting found for IPSOLS using Powell's conjugate directions method suggest that line searches and a strong bias towards the best solutions (due to the setting $w = \varphi_1 = 0$) are well suited for optimizing the considered benchmark functions. In fact, these insights are exploited in the next section, where a new strategy for calling and controlling Powell's conjugate directions method is proposed.

4.5.3 Stage II: Changing the Strategy for Calling and Controlling the Local Search Method

The results of the tuning process presented in the previous section are very interesting because they are counterintuitive at first sight. The parameter w in Eq. 2.3 controls the influence of the velocity of particles in the computation of new candidate solutions. Thus, setting $w = 0$ removes all influence of the particles' previous moves on the generation of new solutions. The parameter φ_1 in Eq. 2.3 controls the strength of the attraction of a particle toward the best solution it has ever found. Thus, setting $\varphi_1 = 0$ removes all influence of the particles' memory on the generation of new solutions. Setting $\varphi_2 > 0$, as was the case in the experiments of the previous section, accelerates particles toward

Table 4.13: Default and best configuration obtained through iterated F-Race for IPSOLS-Stage-II in 10-dimensional instances[†]

Configuration	w	φ_1	φ_2	Init. pop. size	k	Topology	FTol	MaxITER
Default	0.72	1.49	1.49	1	1	FC	-1	10
Tuned	0.02	2.51	1.38	85	9	FC	-11.85	10

[†] FC stands for fully connected, R for ring. FTol and MaxITER are parameters that determine the stopping condition of the local search method.

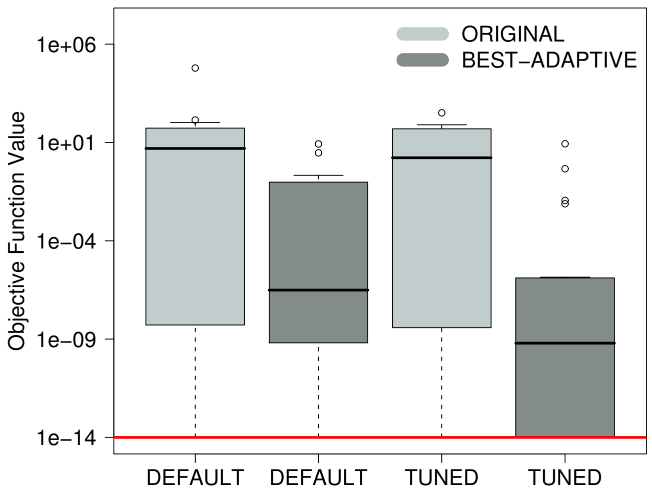


Figure 4.9: Box-plot of the median objective function values obtained by the two configurations listed in Table 4.12 of IPSOLS-Stage-I using Powell's conjugate directions method, and the two configurations listed in Table 4.13 of IPSOLS-Stage-II. In all cases, we used the 100-dimensional versions of the benchmark functions proposed for this special issue. The line at the bottom represents the 0-threshold.

their best neighbor's position. It seems that a ring topology provided the necessary search diversification to avoid the premature convergence that a fully-connected topology would have induced in this case.

In this second stage of the redesign process, we integrate some of these insights into the algorithm itself by changing the strategy for calling and controlling the local search method. This new strategy seeks to enhance the search around the best-so-far-solution. Two changes with respect to the original version are introduced. First, the local search procedure is no longer called from each particle's previous best position. Instead, the local search procedure is only called from the best-so-far solution. Second, the step size that controls the granularity of the local search procedure is no longer fixed; it changes according to the state of the search. This "adaptive" step size control is implemented as follows: a particle, different from the best particle, is picked at random. The maximum norm ($||\cdot||_\infty$) of the vector that separates this random particle from the best particle is used as the local search step size. At the beginning, if the swarm size is equal to one, the step size is a random number in the range $[0, |X|)$, where $|X| = x_{max} - x_{min}$, x_{min} and x_{max} are the minimum and maximum limits of the search range. With these changes, we focus the search around the best-so-far solution with a further local search enhancement through step sizes that tend to decrease over time due to the swarm's convergence.

In Table 4.13, we show the default and best configurations for IPSOLS-Stage-II, as described above.

Table 4.14: Default and best configuration obtained through iterated F-Race for IPSOLS-Stage-III in 10-dimensional instances[†]

Configuration	w	φ_1	φ_2	Init. pop. size	k	Topology	FTol	MaxITER
Default	0.72	1.49	1.49	1	1	FC	-1	10
Tuned	0.0	0.0	2.34	45	10	FC	-1	5

[†] FC stands for fully connected, R for ring. FTol and MaxITER are parameters that determine the stopping condition of the local search method.

A comparison of the results obtained with IPSOLS-Stage-I and IPSOLS-Stage-II is shown in Figure 4.9. Statistically significant differences between the results obtained with IPSOLS-Stage-I and IPSOLS-Stage-II are detected through the application of Wilcoxon's test. The default and tuned configurations of IPSOLS-Stage-II are better than those of IPSOLS-Stage-I ($p = 0.0006$ and $p = 0.01$, respectively).

The parameter values that correspond to the tuned configuration of IPSOLS-Stage-II are now different from those found for IPSOLS-Stage-I. The inertia weight is still very small, but the acceleration coefficient φ_1 is not. Moreover, the initial population size increased from three to 85. This is interesting because, given that local search is only applied to the best particle, increasing the initial population size increases the chances of selecting a good initial solution from which to call the local search method. The increment of the particle addition schedule, k, seems to indicate that IPSOLS-Stage-II needs a greater number of iterations with a constant population size than the previous version. This phenomenon may be due to the fact that the step size now depends on the spatial spread of the swarm. By having a slower particle addition schedule, particles have more time to converge, which allows the local search step size to decrease to levels that allow further progress. The fact that the parameter FTol decreased to a value of -11.85 is also interesting; however, it is not clear whether the local search method actually reaches such small tolerance values, given the fact that the setting for the parameter MaxITER is the same as the default.

4.5.4 Stage III: Vectorial PSO Rules

IPSOLS-Stage-III is the same as IPSOLS-Stage-II except for a modification of Eq. 2.3. In the original PSO algorithm and in most, if not all, variants proposed so far, particles move independently in each dimension of the search space. In contrast, IPSOLS-Stage-III uses a modification of Eq. 2.3 so that particles accelerate *along* the attraction vectors toward their own personal best position and their neighbor's personal best position. This modification is straightforward and consists of using the same pseudorandom numbers on all dimensions instead of generating different numbers for each dimension. This modification is inspired by Powell's conjugate directions method. Once a good direction of improvement is detected, the algorithm searches along that direction. Since the vectorial PSO rules helps the algorithm search along fruitful directions, and Powell's conjugate directions method always starts searching along the original coordinate system, we conjectured that their combination would work well on both separable and nonseparable problems.

Table 4.14 lists the default and the tuned configurations of IPSOLS-Stage-III. The distributions of the median objective function values obtained with IPSOLS-Stage-II and IPSOLS-Stage-III are shown in Figure 4.10.

Through the use of Wilcoxon's test, it is possible to reject the null hypothesis of equal performance when comparing the default configurations ($p = 0.004$). However, it is not possible to reject the null hypothesis in the case of the tuned configurations ($p = 0.06$). The addition of the vectorial update rules does not return a significant improvement over IPSOLS-Stage-II when both versions are tuned; however, with default settings it does. Therefore, we keep the vectorial update rules because they make the IPSOLS-Stage-III less sensitive to different parameter values.

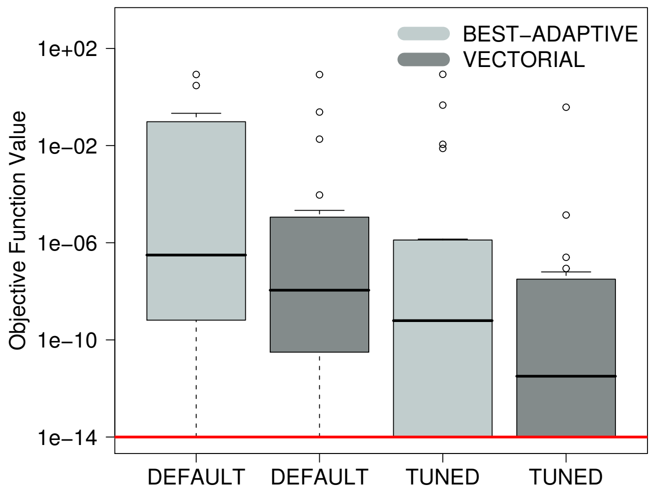


Figure 4.10: Box-plot of the median objective function values obtained by the two configurations listed in Table 4.13 of IPSOLS-Stage-II, and the two configurations listed in Table 4.14 of IPSOLS-Stage-III. In all cases, we used the 100-dimensional versions of the benchmark functions proposed for this special issue. The line at the bottom represents the 0-threshold.

4.5.5 Stage IV: Penalizing Bound Constraints Violation

One advantage of BOBYQA over Powell's conjugate directions method is that is has a built-in mechanism for dealing with bound constraints. In tests with IPSOLS-Stage-III, we observed that Powell's conjugate directions method would make some excursions outside the bounds.

Hence, we opted for including a mechanism into IPSOLS-Stage-III that would help to enforce bound constraints. These constraints are enforced because it is known that some benchmark functions give the impression that very good solutions are outside the defined bounds. A well-known example of this phenomenon is Schwefel's sine root function. In initial experiments, we noted that simply setting solutions to the bound deteriorated the performance of the algorithm significantly. IPSOLS-Stage-IV tries to include a mechanism to enforce boundary constraints using the penalty function

$$P(\boldsymbol{x}) = \mathit{fes} \cdot \sum_{i=1}^{n} B(x_i)\,, \tag{4.14}$$

where $B(x_i)$ is defined as

$$B(x_i) = \begin{cases} 0, & \text{if } x_{\min} \leq x_i \leq x_{\max} \\ (x_{\min} - x_i)^2, & \text{if } x_i < x_{\min} \\ (x_{\max} - x_i)^2, & \text{if } x_i > x_{\max}\,, \end{cases} \tag{4.15}$$

where n is the dimensionality of the problem, x_{min} and x_{max} are the minimum and maximum limits of the search range, respectively, and *fes* is the number of function evaluations that have been used so far. The goal with this penalty function is to discourage long and far excursions outside the bounds. Strictly speaking, Eq. 4.14 does not change the algorithm but the objective function. Nevertheless, we describe it as if it was part of the algorithm because bound constraints handling mechanisms are important components of any algorithm.

Table 4.15: Default and best configuration obtained through iterated F-Race for IPSOLS-Stage-IV in 10-dimensional instances[†]

Configuration	w	φ_1	φ_2	Init. pop. size	k	Topology	FTol	MaxITER
Default	0.72	1.49	1.49	1	1	FC	-1	10
Tuned	0.0	0.0	2.32	64	5	FC	-1.32	4

[†] FC stands for fully connected, R for ring. FTol and MaxITER are parameters that determine the stopping condition of the local search method.

Table 4.15 shows the default and tuned configurations of IPSOLS-Stage-IV. As may be expected, not many changes in the tuned configuration can be found with respect to the tuned configuration of IPSOLS-Stage-III.

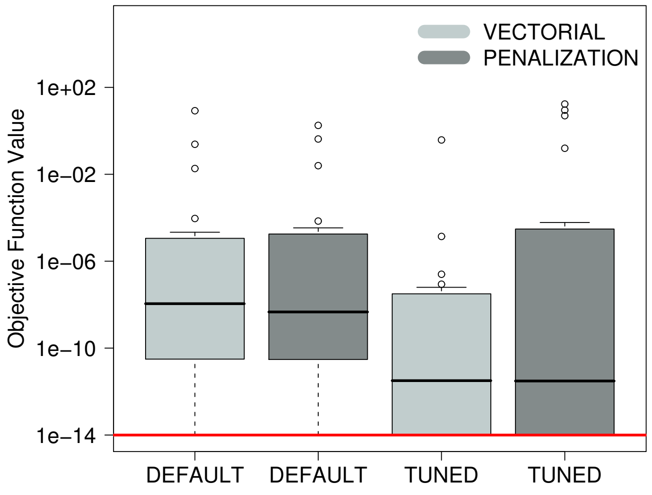


Figure 4.11: Box-plot of the median objective function values obtained by the two configurations listed in Table 4.14 of IPSOLS-Stage-III, and the two configurations listed in Table 4.15 of IPSOLS-Stage-IV. In all cases, we used the 100-dimensional versions of the benchmark functions proposed for this special issue. The line at the bottom represents the 0-threshold.

Figure 4.11 shows the box-plots of the median objective function values obtained with IPSOLS-Stage-III and IPSOLS-Stage-IV. IPSOLS-Stage-III and IPSOLS-Stage-IV differ slightly, and consequently, we expect that their performance is slightly different. Our expectation is confirmed through the application of Wilcoxon's test on the samples of results. The default and tuned versions can be considered equivalent ($p = 0.87$ and $p = 0.81$, respectively). Therefore, mainly motivated by the potential importance of being able to enforce bound constraints, we kept the design of IPSOLS-Stage-IV.

4.5.6 Stage V: Fighting Stagnation by Modifying the Local Search Call Strategy

We noticed in preliminary experiments that IPSOLS-Stage-IV stagnated in functions in which IPSOLS-Stage-I had very good results (e.g., F_5). A common approach to deal with stagnation is to add diversification strategies (Hoos and Stützle, 2004). The first diversification strategy that we added to IPSOLS-Stage-IV is based on the way IPSOLS-Stage-I operates. (The second strategy is described in the next section.) Specifically, in

Table 4.16: Default and best configuration obtained through iterated F-Race for IPSOLS-Stage-V in 10-dimensional instances[†]

Configuration	w	φ_1	φ_2	Init. pop. size	k	Topology	FTol	MaxITER	MaxFailures
Default	0.72	1.49	1.49	1	1	FC	-1	10	2
Tuned	0.0	0.0	2.6	3	1	FC	-1	7	20

[†] FC stands for fully connected, R for ring. FTol and MaxITER are parameters that determine the stopping condition of the local search method.

IPSOLS-Stage-I, all particles call the local search method, but in IPSOLS-Stage-IV, only the best particle does. In IPSOLS-Stage-V, we sporadically let other particles call the local search method. We introduce a parameter, MaxFailures, which determines the maximum number of repeated calls to the local search method from the same initial solution that does not result in a solution improvement. Each particle maintains a failures counter and when that counter reaches the value MaxFailures, the local search procedure cannot be called again from that particle's personal best position. In that situation, local search is applied from a random particle's personal best position as long as this random particle's failures counter is less than MaxFailures. If the random particle's failures counter is equal to MaxFailures, the algorithm continues sampling particles at random until it finds one that satisfies the requirements, or until all particles are tried. This last situation is not likely to happen because new particles are periodically added to the population. When a particle's personal best position improves thanks to a PSO move, the failures counter is reset so that the local search procedure can be called again from that newly discovered solution.

In Table 4.16, the default and tuned configurations of IPSOLS-Stage-V are listed. The distributions of the median objective function values of the default and tuned configurations of IPSOLS-Stage-IV and IPSOLS-Stage-V are shown in Figure 4.12.

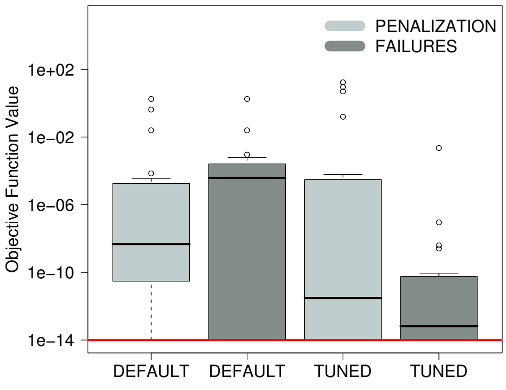


Figure 4.12: Box-plot of the median objective function values obtained by the two configurations listed in Table 4.15 of IPSOLS-Stage-IV, and the two configurations listed in Table 4.16 of IPSOLS-Stage-V. In all cases, we used the 100-dimensional versions of the benchmark functions proposed for this special issue. The line at the bottom represents the 0-threshold.

So far, we have been able to accept all modifications based on the analysis of the distributions of median objective function values. In this case, the introduced modification with default parameter settings is worse than the unmodified version with default settings ($p = 0.03$). After tuning, the null hypothesis that assumes that both samples are drawn from the same population cannot be rejected ($p = 0.36$). It would seem clear that the proposed modification could be safely rejected. However, if we look at the distribution

Table 4.17: Default and best configuration obtained through iterated F-Race for IPSOLS-Stage-VI in 10-dimensional instances[†]

Configuration	w	φ_1	φ_2	Init. pop. size	k	Topology	FTol	MaxITER	MaxFailures	MaxStagIter
Default	0.72	1.49	1.49	1	1	FC	-1	10	2	10
Tuned	0.0	0.84	1.85	1	1	FC	-1	87	13	27

[†] FC stands for fully connected, R for ring. FTol and MaxITER are parameters that determine the stopping condition of the local search method.

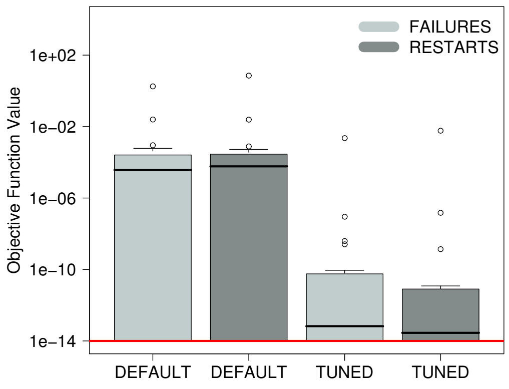


Figure 4.13: Box-plot of the median objective function values obtained by the two configurations listed in Table 4.16 of IPSOLS-Stage-V, and the two configurations listed in Table 4.17 of IPSOLS-Stage-VI. In all cases, we used the 100-dimensional versions of the benchmark functions proposed for this special issue. The line at the bottom represents the 0-threshold.

of the mean objective function values instead of the median[4], the situation is completely different. In this case, the default and the tuned configurations of IPSOLS-Stage-V are better than the one of IPSOLS-Stage-IV ($p = 0.002$ and $p = 0.01$, respectively). This result indicates that the introduced modification indeed reduces the likelihood of IPSOLS-Stage-IV stagnating, but this is seen only in the upper quartiles of the solution quality distributions generated by the algorithm, that is, the worst results are improved but not necessarily the best results. In fact, the lower quartiles of these distributions (i.e., the best solutions) either deteriorated, as with the default configuration, or are not affected, as seems to be the case with the tuned configuration.

4.5.7 Stage VI: Fighting Stagnation with Restarts

In IPSOLS-Stage-VI, we included an algorithm-level diversification strategy. This strategy consists in restarting the algorithm but keeping the best-so-far solution in the population of particles across restarts. In particular, the best-so-far solution becomes the new first particle's current and previous best position. The restart criterion is the number of PSO-level consecutive iterations with a relative solution improvement lower than a certain threshold ϵ. In our experiments, we set $\epsilon =$ *0-threshold*. The number of consecutive iterations without significant improvement is a parameter of IPSOLS-Stage-VI, MaxStagIter.

The list of default and tuned parameter settings for IPSOLS-Stage-VI is shown in Table 4.17. The distributions of the median objective function values obtained by the default and tuned configurations for IPSOLS-Stage-V and IPSOLS-Stage-VI are shown in Figure 4.13.

When comparing the aggregated data of IPSOLS-Stage-V and IPSOLS-Stage-VI using

[4]We remind the reader that the complete set of results can be found in (Montes de Oca, 2011).

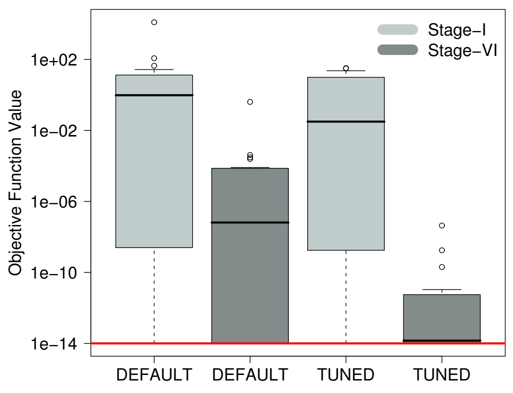


Figure 4.14: Box-plot of the median objective function values obtained by the two configurations listed in Table 4.12 of IPSOLS-Stage-I using Powell's conjugate directions method, and the two configurations listed in Table 4.17 of IPSOLS-Stage-VI. In all cases, we used the 100-dimensional versions of the benchmark functions proposed for this special issue. The line at the bottom represents the 0-threshold.

a Wilcoxon test, the null hypothesis cannot be rejected with medians or with means. While the actual performance of IPSOLS-Stage-VI is statistically equivalent to that of IPSOLS-Stage-V with the 100-dimensional versions of the benchmark functions, we prefer to keep the extra diversification layer that a strong restart offers. During scalability tests, which will be presented later, we will see that this extra diversification layer does not hurt performance and does provide extra flexibility that can be exploited in certain application scenarios, such as when a larger number of function evaluations can be used.

With IPSOLS-Stage-VI, we conclude the redesign cycle of IPSOLS. In Figure 4.14 we can clearly see the improvement with respect to the original algorithm that we were able to obtain through a tuning-in-the-loop redesign process.

It is interesting to see the positive effects that tuning had on most of the tested variants of IPSOLS. It should not be forgotten that the tuning process was performed with instances whose size was only 10% the size of the instances over which we tested the effectiveness of tuning. In the following section, we will see that the effectiveness of tuning also extends to instances that are up to 100 times larger than the ones seen during tuning.

4.5.8 Performance Scalability Study

In this section, we study the performance of IPSOLS-Stage-VI, which we refer to as IPSOLS+, on the 50-, 100-, 200-, 500-, and 1000-dimensional versions of the 19 benchmark functions shown in Table 4.8.

Reference Algorithms

The performance of the tuned version of IPSOLS+ was compared to that of 15 other algorithms, which were recently proposed in a special issue of the Soft Computing journal (Lozano et al., 2011). These algorithms include differential evolution (DE) (Storn and Price, 1997), the CHC algorithm (Eshelman and Schaffer, 1993), and a CMA-ES algorithm with increasing population size (G-CMA-ES) (Auger and Hansen, 2005). In recent years, DE and G-CMA-ES have been widely considered as representatives of the state-of-the-art of heuristic continuous optimization problems. In particular, G-CMA-ES was the best-ranked algorithm of a special session organized for the 2005 IEEE Congress on Evolutionary Computation (Smit and Eiben, 2010). The parameter settings used with DE, CHC, and G-CMA-ES were proposed by Lozano and Herrera (2010a). The parameter settings for the other algorithms were set by their respective authors. Moreover, these algorithms' source code and summary statistics of their performance on the 19 benchmark functions listed in Table 4.8 are available at Lozano and Herrera (2010b). For more information about these algorithms, we refer the reader to (Lozano et al., 2011) and the papers in which they were proposed.

Solution Quality Scalability

We use box-plots to show the distribution of the average objective function values obtained by each algorithm on the 19 benchmark functions suite (19 values per box-plot). These box-plots are shown in Figure 4.15.

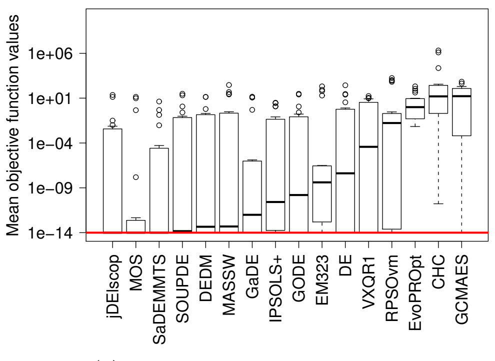
The first striking result of this comparison is that the performance of G-CMA-ES is very poor in comparison to that of the other 15 algorithms including IPSOLS+. The performance of IPSOLS+ improves with the dimensionality of the benchmark functions used for this comparison as can be seen by the ranking of IPSOLS+ based on the median value of the distribution of the 19 average objective function values. We tested the null hypothesis with a multiple pairwise Wilcoxon's test with a 0.95 confidence level. Except for the comparisons of IPSOLS+ to CHC, EvoPROpt, and G-CMA-ES, it is not possible to reject the null hypothesis of IPSOLS+ with any other algorithm. Thus, we can confidently say that IPSOLS+ obtains results that are not distinguishable from those obtained with state-of-the-art algorithms for small and large-scale continuous optimization problems.

Execution Time Scalability

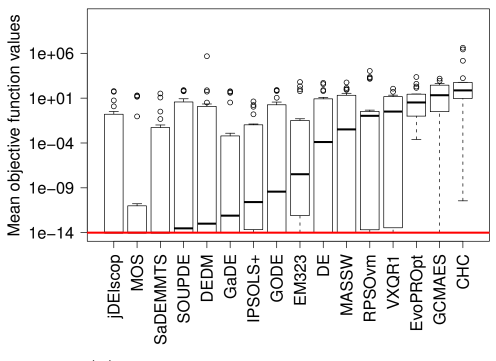
The results presented so far have focused on the solution quality obtained after some computational budget (in our case, the maximum number of function evaluations) has expired. While those results are important, it is still unclear whether the proposed algorithms are really suitable for tackling large-scale continuous optimization problems. This ambiguity is due to the fact that, in some cases, the execution time can grow so fast with the problem size that an otherwise good algorithm may become impractical. For example, the execution time of G-CMA-ES was so high that its results on 1000-dimensional problems were not even reported in the special issue of the Soft Computing journal (Lozano et al., 2011).

In this section, we study the execution time scalability of IPSOLS+. During each run of the algorithm, we recorded the number of function evaluations and the CPU time used, when the best-so-far solution was improved. With this information, we can estimate the solution quality distribution (SQD) (Hoos and Stützle, 2004) of the algorithm at, for example, the maximum number of function evalutions. To conduct the scalability study, we use the 0.9-quantile of the SQDs of IPSOLS+ on each benchmark function. We chose the 0.9-quantile as a conservative measure of the achievable solution quality. We focus on two measures: the median number of function evaluations (FES) and the median time (in seconds) needed by IPSOLS+ (in its default configuration) to find a solution that is at least as good as the 0.9-quantile of the SQD after up to $5000n$ function evaluations, where n is the dimensionality of the function tackled. Since the scalability behavior of any algorithm is problem-dependent, we show only three examples in Figure 4.16. The rest of the results can be found in (Montes de Oca, 2011).

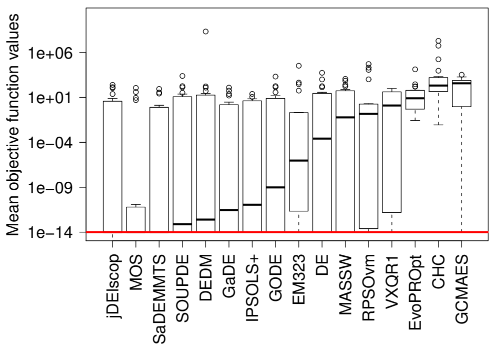
We first focus on the behavior of the algorithm with respect to the number of function evaluations consumed. For function F_1, the linear model $fes = 18.08n - 424.96$ fits the data with an adjusted r^2 score of 0.9957. The variable fes is the number of function evaluations needed by IPSOLS+ to find a solution at least as good as the 0.9-quantile of the SQD, and n is the dimensionality of the function. It is clear that the time execution scalability of IPSOLS+ with function F_1 is quite good as a consequence of the fact that Powell's conjugate directions method exploits the separability of the function. In the case of function F_8, the linear model $fes = 4699.85n + 16261.51$ fits the data with an adjusted r^2 score of 0.9999. The slope of this model is almost the same as the slope of the computational budget limit, which means that IPSOLS+ would most likely continue making progress



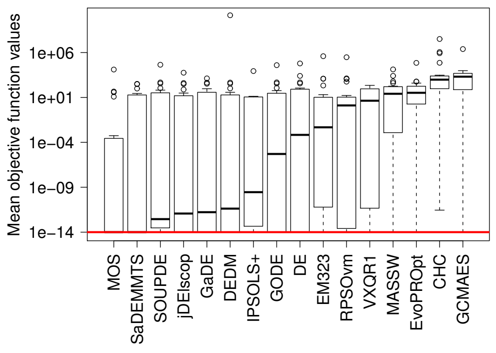
(a) $n = 50$, IPSOLS+ ranking: 8



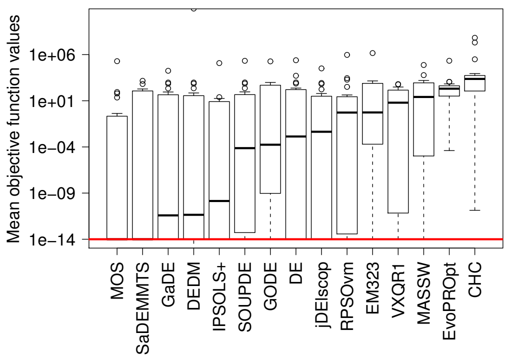
(b) $n = 100$, IPSOLS+ ranking: 7



(c) $n = 200$, IPSOLS+ ranking: 7

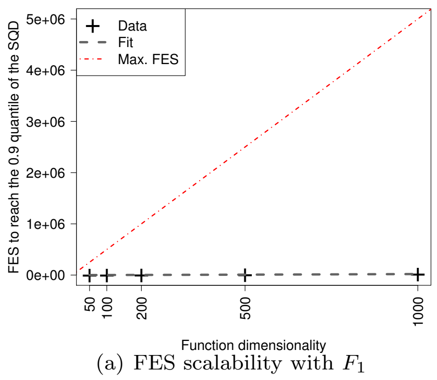


(d) $n = 500$, IPSOLS+ ranking: 7

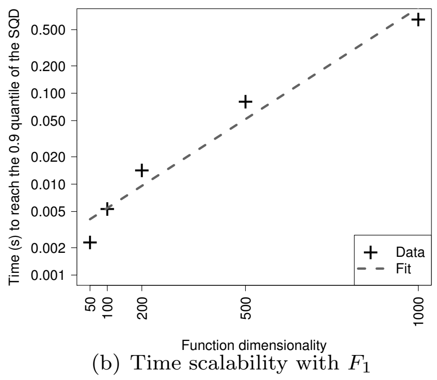


(e) $n = 1000$, IPSOLS+ ranking: 5

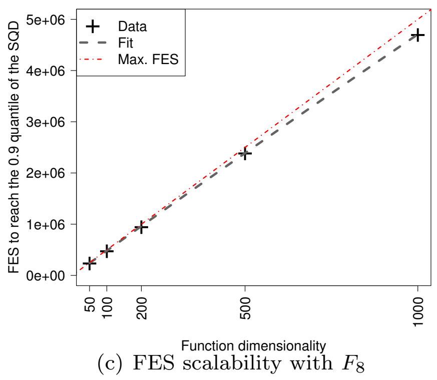
Figure 4.15: Distribution of the 19 average objective function values obtained by each of the 16 compared algorithms. The boxplots are sorted in ascending order based on the median value of the distribution. In the plot that corresponds to 1000 dimensions, the results obtained with G-CMA-ES are missing due to this algorithm's excessive computation time for this dimensionality. The line at the bottom of each plot represents the 0-threshold.



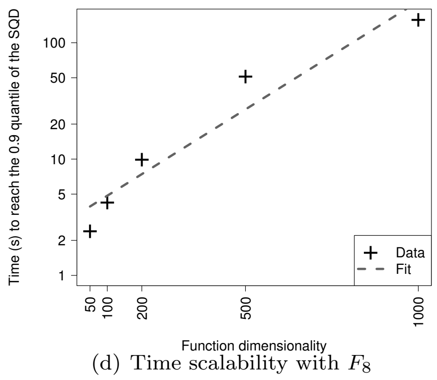
(a) FES scalability with F_1



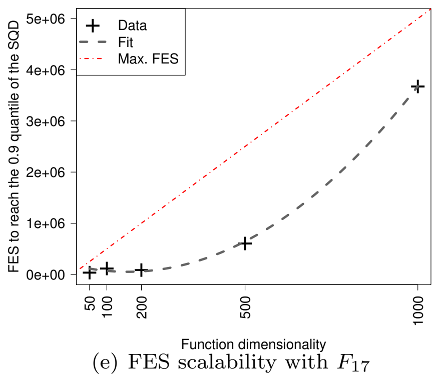
(b) Time scalability with F_1



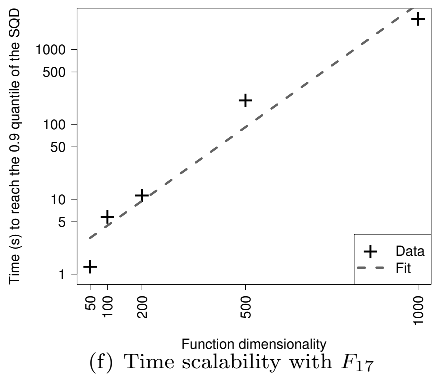
(c) FES scalability with F_8



(d) Time scalability with F_8



(e) FES scalability with F_{17}



(f) Time scalability with F_{17}

Figure 4.16: Median number of function evaluations (FES) and median number of seconds needed by IPSOLS+ to find a solution at least as good as the 0.9-quantile of the solution quality distribution after up to $5000n$ FES (i.e., the maximum computational budget represented by the thin dotted line) on functions F_1, F_8, and F_{17}. The result of a regression analysis over the observed data is shown with a dotted line.

toward better solutions if more function evaluations were allowed. This result is due to the Powell's conjugate directions method, which in the case of F_8 would align the search directions, one by one, with the axes of the hyperellipsoid that F_8 generates. Finally, we show the execution time of IPSOLS+ with function F_{17}, which scales quadratically with the size of the problem. In this case, the model is $fes = 5.14n^2 - 1643.03n + 182300$ with an adjusted r^2 score of 0.998. The extra complexity that this function represents for IPSOLS+ is evident.

In terms of CPU time, the scalability of IPSOLS+ seems to follow the form $time = Ae^{Bn}$. For F_1, the parameters of the fitted model are $A = 0.0031$ and $B = 0.0056$ with an adjusted r^2 score equal to 0.949. For F_8, the parameters are $A = 3.1598$ and $B = 0.0042$ with an adjusted r^2 score equal to 0.902. Finally, for F_{17}, the parameters are $A = 2.0851$ and $B = 0.0075$ with an adjusted r^2 score equal to 0.936. Even though the fitted model is exponential, the scalability of IPSOLS+ in seconds is very good because the coefficient B is rather small (in the order of 1e−03) for all the 19 functions studied. We ran our experiments on Intel Xeon E5410 quad core 2.33GHz computers with 2×6MB L2 cache and 8GB of RAM. All computers used Linux (kernel 2.6.9-78.0.22) as operating system, and IPSOLS+ was compiled with GCC 4.1.0.

4.6 Related Work

IPSO, IPSOLS, and IPSOLS+ are PSO-based algorithms in which the population size changes during the optimization process. IPSOLS and IPSOLS+ are also hybrid algorithms in which PSO and a local search procedure are combined. In this section, we briefly review related work on both of these topics. We highlight the differences that exist between previous approaches and IPSO, IPSOLS, and IPSOLS+.

4.6.1 PSO Algorithms with Time-Varying Population Size

Population sizing has been studied within the field of evolutionary computation for many years. From that experience, it is now generally accepted that the population size in evolutionary algorithms should be proportional to the problem's difficulty (Lobo and Lima, 2007). The issue is that it is not uncommon to know little about a problem's difficulty *a priori*. As a result, evolutionary algorithms with *time-varying* population size have been proposed (see e.g. Arabas et al. (1994); Harik and Lobo (1999); Bäck et al. (2000); Eiben et al. (2004); Auger and Hansen (2005); Eiben et al. (2006); Fernandes and Rosa (2006)). This research issue has just recently been addressed by the PSO community, and thus not many research contributions exist. Coelho and de Oliveira (2008) adapt the population resizing mechanisms used in APGA (Bäck et al., 2000) and PRoFIGA (Eiben et al., 2004) for their use in PSO algorithms. Lanzarini et al. (2008) proposed a method for varying the size of the population by assigning a maximum lifetime to groups of particles based on their performance and spatial distribution. A time-varying population size approach has been adopted by Leong and Yen (2008) for tackling multiobjective optimization problems with PSO algorithms. In the work of Chen and Zhao (2009), the optimization process is divided into a number of periods at the end of which the population size changes. The decision of whether the population size should increase or decrease depends on a diversity measure. Finally, Hsieh et al. (2009) adapt the swarm size based on the ability of the particles to improve their personal best solutions and the best-so-far solution.

All these proposals share a common problem: they eliminate the population size parameter, but introduce many others. For example, many proposals require the user to set a particle's maximum lifetime, to select the number of iterations without improvement so that a particle is added or removed, to choose particle recombination operators, etc. In contrast, our approach introduces only two parameters: the rate at which the population size should grow and how new particles should be initialized. Additionally, our approach is simple to understand and implement.

In contrast to practically all previously studied strategies, our approach, in its current form, does not consider the possibility of reducing the size of the population during an algorithm's run. The rationale behind previous approaches is that large populations require more function evaluations per iteration and thus, if the particles have converged, they can result in a waste of function evaluations. However, in at least another algorithm the population size is not decreased. Such an algorithm is G-CMA-ES (Auger and Hansen, 2005), in which the population size is doubled each time it is restarted. As we have seen, not decreasing the population size does not negatively affect the performance of G-CMA-ES, IPSO, IPSOLS, and IPSOLS+.

4.6.2 PSO Algorithms Hybridized with Local Search Procedures

The idea of combining local search techniques with PSO algorithms comes partly from the observation that particles are attracted to their own and their neighbors' previous best positions. The underlying idea is that the better the attractors of a particle are, the higher the chances that a particle finds even better solutions. The goal of most hybrid algorithms, IPSOLS and IPSOLS+ included, is thus to accelerate the placement of the particles' previous best positions in good locations. For example, Chen et al. (2005) combined a particle swarm algorithm with a hill-climbing local search procedure. Liang and Suganthan (2005) used a quasi-Newton method to improve a subset of the solutions found by a multi-swarm algorithm. Gimmler et al. (2006) experimented with PSO-based hybrids using Nelder and Mead's simplex method and Powell's conjugate directions set method. In their results, the hybrid algorithm that uses the Powell's conjugate directions set method obtained better results than the algorithm that uses Nelder and Mead's simplex method. Das et al. (2006) also used Nelder and Mead's simplex method and proposed the inclusion of an estimate of the local gradient into the particles' velocity update rule. In (Coelho and Mariani, 2006), a two-phase approach is described where a PSO algorithm is used first to find a good solution and, in a second phase, a quasi-Newton method is used to refine it. Petalas et al. (2007) report experiments with several local search-particle swarm combination schemes. Müller et al. (2009) describe in their work a hybrid PSO–CMA-ES algorithm in which a full-fledged population-based algorithm (CMA-ES Hansen et al. (2003); Hansen and Kern (2004)) is used as a local search procedure. Other PSO-local search hybrids are reported in Hao and Hu (2009) and Chen et al. (2010). Our proposal is not different from the above-mentioned approaches in the sense that it uses a local search procedure. In all cases, the goal is to accelerate the discovery of good solutions. However, our work is the first to explore the possible benefits of combining a variable population size with local search procedures in the context of PSO algorithms. We have seen that this combination allows IPSOLS and IPSOLS+ to "adapt" to the features of the objective function as discussed in Section 4.4.2.

4.7 Conclusions and Future Work

In this chapter, we have shown how the ISL framework can be used for enhancing the performance of PSO algorithms. We analyzed and empirically evaluated three algorithms that are the result of applying ISL to PSO. The first one, IPSO, is a PSO algorithm with a growing population size, in which new particles are initialized biasing their initial position toward the best-so-far solution. The second algorithm, IPSOLS, is an extension of IPSO, which implements "individual learning" through a local search procedure. The third algorithm, called IPSOLS+, resulted from a redesign process in which an automatic tuning system, iterated F-Race, was used at each design stage. IPSOLS+ is the most competitive of the three algorithms proposed. In IPSOLS+, a local search procedure is almost always invoked from the best-so-far solution. However, when this strategy is not successful, local search is invoked from the position of a randomly chosen particle. The local search procedure and the PSO algorithm are tightly coupled because the initial step size used in the local search procedure depends on the interparticle distance. Thus, the local search is naturally more focused toward the end of the optimization process. A restart mechanism is used in order to increase the chances of the algorithm of finding good quality solutions. Despite their differences, all of these three algorithms keep the two basic elements of the ISL framework: (i) incremental growth of the population and (ii) social learning at the moment a particle is added. We showed that the effects of the social learning rule are positive on a very wide range of problems.

We are not the first to use automatic parameter configuration methods in the context of heuristic algorithms for continuous optimization. Earlier approaches designed to tune numerical parameters of algorithms for continuous optimization include SPO (Bartz-Beielstein, 2006), SPO$^+$ (Hutter et al., 2009), and REVAC (Nannen and Eiben, 2007). A comparison of such methods has been presented in (Smit and Eiben, 2009). In all these approaches, however, the algorithm was tuned on the very same benchmark function on which it was later tested; thus, all these examples are prone to over-tuning. Only rather recently, an experimental comparison of tuning algorithms for calibrating numerical algorithm parameters in numerical optimization that does not incur the problems of over-tuning has been presented (Yuan et al., 2010). In this chapter, we also avoid over-tuning by applying iterated F-race to problems of much smaller dimension than the ones on which the algorithms are later tested. In addition, in contrast with previous work, we make extensive usage of automatic tuning *during* the design process of a high-performing algorithm; others have so far focused on the fine-tuning of an already developed, final algorithmic scheme.

There are a number of promising avenues for future research. First, an important issue that needs to be addressed in the future is the applicability of ISL to other population-based optimization techniques. In principle, ISL can be used with any population-based optimization algorithm. However, it is not always evident how to apply ISL to a certain algorithm. For example, it is not straightforward that one may apply ISL to ACO algorithms (Dorigo and Stützle, 2004), which have a centralized memory structure that already allows agents (in this case artificial ants) to share their search experience with others. Nevertheless, Liao et al. (2011) have recently shown that it is possible to apply ISL to ACO$_{\mathbb{R}}$ (Socha and Dorigo, 2008), which is an ACO variant for continuous optimization problems. Second, it is important to extend automatic algorithm configuration techniques to very large-scale problems in order to better deal with the scaling behavior of algorithm parameters. In fact, our algorithm tuning was done on 10-dimensional versions of the high-dimensional benchmark functions and, maybe luckily, the found parameter settings turned out to result in very high performance even on benchmark functions that had 100 times larger dimension. Further research is needed to explore better ways to find well-scaling algorithm parameters. Third, we need to further investigate methods for automatic algorithm configuration for tackling continuous optimization problems. Some approaches exist, but often these suffer from over-tuning since tuning is done on single benchmark functions. Finally, another promising direction for future work is to apply our tuning-in-the-loop algorithm engineering methodology to other algorithms for continuous function optimization. In fact, the DE algorithm proposed as a baseline for the competition would be an interesting candidate for such an undertaking.

Chapter 5

Incremental Social Learning Applied to Robot Swarms

In this chapter, we describe the application of ISL to a collective decision-making mechanism for robot swarms. This mechanism is also a contribution of this dissertation as mentioned in Section 1.3. First, we describe the collective decision-making mechanism (Sections 5.1 and 5.2). Then, in Section 5.3, we describe the integration of such a mechanism with ISL and report the results obtained. Related work is described in Section 5.4. Opportunities for future work and conclusions are presented in Section 5.5.

5.1 Majority-Rule Opinion Formation Models

When a person is immersed in a social context, her decisions are influenced by those of others. The effects of social influence on the collective-level behavior of groups of people have been studied by economists and sociologists since at least the 1970s (Schelling, 1978; Granovetter, 1978). More recently, statistical physicists have developed models to quantitatively describe social and economic phenomena that involve large numbers of interacting people (Chakrabarti et al., 2006; Castellano et al., 2009; Helbing, 2010). Some of the models that have emerged from these efforts are referred to as *opinion formation models*.

Krapivsky and Redner (2003) proposed a binary opinion formation model in which a population of agents reaches a consensus with high probability on the opinion initially favored by more than 50% of the population. The process that drives the system to consensus is based on the repeated application of the majority rule at a local level on small teams of agents (see Section 5.1.1). This model is interesting from a swarm intelligence perspective because the resulting opinion dynamics can be seen as a decentralized collective decision-making process. However, to be of practical use, the opinion dynamics induced by the majority rule need to make an initially unbiased population reach consensus on the opinion associated with the "best" alternative. In this chapter, we demonstrate how to achieve this goal in a swarm robotics context by making opinions represent actions robots need to choose from while executing a task (see Section 5.1.2). The criterion used to evaluate alternative actions is the time needed to execute them. Thus, an action that has the same effect as another one but that takes less time to perform is preferred.

We introduce a number of modifications to the majority-rule opinion formation model that capture elements of the interaction of real robots with a physical environment. One of these modifications builds on the concept of latency, which is a period of time of stochastic duration during which an agent cannot be influenced by other agents, and thus cannot change opinion (Lambiotte et al., 2009). In our model, we call this modification *differential latency* because the duration of a latency period is different for different opinions. We demonstrate, in simulation, that with the introduced modifications, a population of agents reaches consensus on the opinion associated with the shortest average latency even if that opinion is initially favored by a slight minority of the population. In Section 5.2, we propose

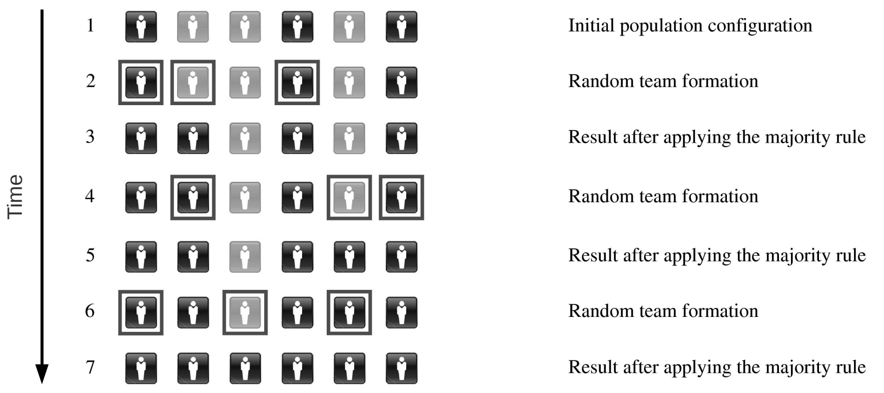

Time
1
2
3
4
5
6
7
Initial population configuration
Random team formation
Result after applying the majority rule
Random team formation
Result after applying the majority rule
Random team formation
Result after applying the majority rule

Figure 5.1: Majority-rule opinion dynamics. Initially, three agents have opinion A (represented in black) and three others have opinion B (represented in gray). After applying three times the majority rule on randomly-formed teams of three agents each (marked with squares), the population reaches consensus on one of the two opinions.

a collective decision-making mechanism for robot swarms that exploits the dynamics of the majority-rule opinion formation model with differential latency.

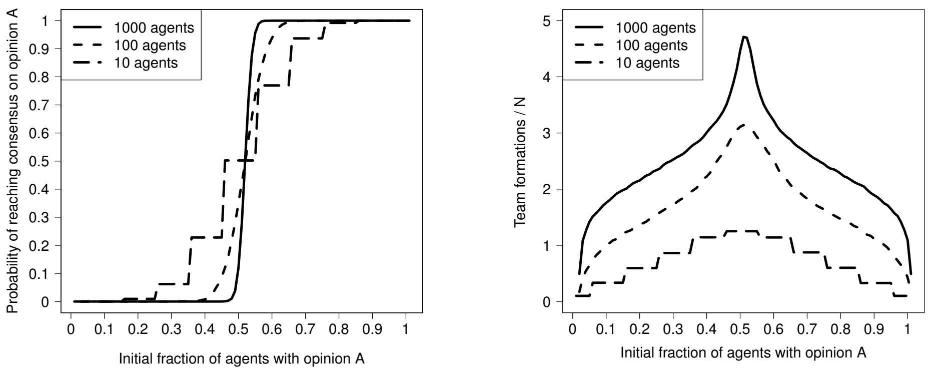
5.1.1 Majority-Rule Opinion Formation With and Without Latency

The majority rule as an element of opinion formation models was first used by Galam (1986) to study voting in hierarchical structures. Krapivsky and Redner (2003) studied the dynamics induced by the majority rule in a well-mixed population case, that is, a situation where everyone can interact with the same probability with everyone else (Nowak, 2006). In Krapivsky and Redner's model, a population of agents, each of which can assume one of two states, called opinions (A or B)[1], evolves as follows: First, a team of three randomly chosen agents is formed. Then, the team members adopt the opinion held by the majority within the team. Finally, the team members are put back in the population and the process is repeated. Figure 5.1 shows an example of the process just described.

An important aspect of the system's dynamics is the probability of reaching consensus on one opinion, say A, as a function of the initial fraction of the population favoring it (see Figure 5.2(a)). In Krapivsky and Redner's model, the value of this probability abruptly increases at a critical initial fraction equal to 0.5. If the initial fraction of the population in favor of opinion A is greater than 0.5, then the population reaches consensus on opinion A with a higher probability than on opinion B. If the initial fraction is exactly 0.5, then the probability of reaching consensus on opinion A is also 0.5. For large populations, this probability approximates a unit step function with a discontinuity at the critical initial fraction. For small populations, the probability is a step-wise function. The number of team formations required to reach consensus in the majority-rule opinion formation model also depends on both the initial fraction of the population favoring one opinion and the population size. At the critical initial fraction, the system takes the longest to reach consensus (see Figure 5.2(b)).

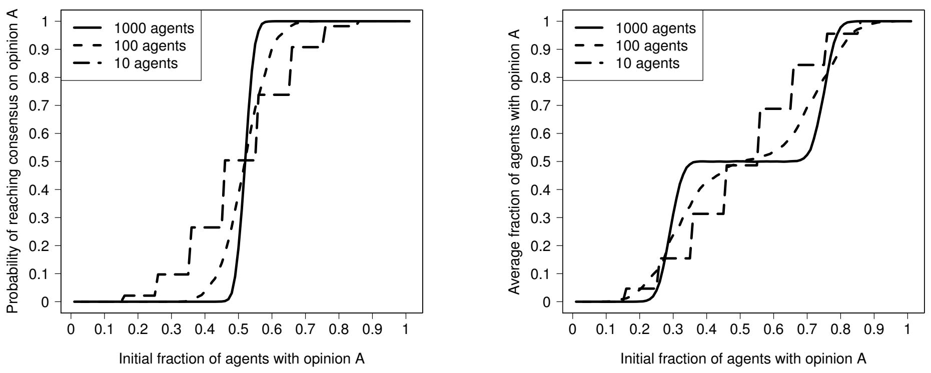
Lambiotte et al. (2009) incorporated *latency* to Krapivsky and Redner's model. In Lambiotte et al.'s model, a team is formed with three randomly picked agents that can be either latent or non-latent. The team's majority opinion is adopted only by the team's non-latent agents. If the team's non-latent agents switch opinion as a result of the majority rule, then they become latent. Otherwise, they remain non-latent. The team's latent

[1]Throughout this chapter, we use letters A and B to label the two available opinions.



(a) (b)

Figure 5.2: Dynamics of Krapivsky and Redner's model. Figure (a) shows the probability of reaching consensus on one opinion (labeled A) as a function of the initial fraction of agents in its favor. Figure (b) shows the average number of team formations needed to reach consensus on one opinion. N is the size of the population. These plots are based on results obtained through 1,000 independent runs of a Monte Carlo simulation.



(a) (b)

Figure 5.3: Dynamics of Lambiotte et al.'s model. Depending on the value of the parameter α, consensus may or may not be the only stable state of the system. When $\alpha = 1/2$ (Figure (a)) consensus is always achieved (we plot the probability of reaching consensus on opinion A). By contrast, when $\alpha = 1/20$ (Figure (b)), the population does not always achieve consensus because a third stable state, in which the fraction of agents favoring one opinion fluctuates around 0.5, arises. Thus, in this case, we plot the average fraction of agents with opinion A after 100,000 team formations. These plots are based on results obtained through 1,000 independent runs of a Monte Carlo simulation.

agents become non-latent with probability α, which is a parameter of the model. In this model, consensus may be reached for any value of α; however, for $\alpha < 1/4$, a third state in which the fraction of agents favoring one opinion fluctuates around 0.5, is also stable. The characteristic dynamics of Lambiotte et al.'s model are shown in Figure 5.3.

5.1.2 Majority-Rule Opinion Formation With Differential Latency

We introduce an opinion formation model based on Krapivsky and Redner's and Lambiotte et al.'s models. The proposed model captures some properties of real-world swarm robotics systems. Our goal is to exploit the resulting system's dynamics as a collective decision-making mechanism. In order to meet our goal, we need to interpret agents as robots, and opinions as actions or sequences of actions that robots have to choose from while solving a task. For example, an opinion can model a robot's decision whether or not to establish a physical connection with another robot—cf. Ampatzis et al. (2009). An example of an opinion representing a sequence of actions is whether to follow the rules for assembling one morphology or another for a specific task—cf. O'Grady et al. (2010a).

The modifications we introduce to Krapivsky and Redner's and Lambiotte et al.'s models are the following:

1. Robots in a swarm can operate in parallel. This property is translated into k independent teams being formed instead of just one as in the original formulations.
2. Robot actions may induce physical displacement. Thus, robots executing an action cannot simultaneously be part of two teams. As a result, robots executing an action cannot be influenced by other robots, and crucially, cannot influence other robots, unless they are in their immediate vicinity. This phenomenon is partially captured by the concept of latency as defined by Lambiotte et al.. However, in Lambiotte et al.'s model, latent agents can still influence other agents. Thus, we restrict agents to be non-latent at the moment of forming a team. This change prevents latent agents from influencing other agents.
3. Robot actions take time to perform. Moreover, the duration of an action is stochastic because there are physical interactions between robots and the environment. In addition, different actions may have different average duration. This is translated into *differential latency*, that is, the average duration of the latency period depends on the agents' adopted opinion. In contrast with Lambiotte et al.'s model in which agents become latent only if they switch opinions, in our case, agents become latent regardless of whether they switch opinion or not.

A system governed by the proposed model evolves as follows: k teams of three randomly chosen agents are formed. The majority rule is used within each team in order to update its members' opinions. Agents that belong to a team enter a latent state whose duration depends on the team's adopted opinion. When a team's latency period finishes, its members become non-latent and eligible to form a new team. When a new team is formed, its members are picked from the population of non-latent agents. The process is repeated until the population reaches a consensus. Algorithm 6 shows a pseudo-code version of the process just described.

5.2 Majority-Rule Opinion Dynamics With Differential Latency as a Collective Decision-Making Mechanism for Robot Swarms

In this section, we study the opinion dynamics induced by the majority-rule opinion formation model with differential latency. The study is performed in two stages. In the first stage, we use Monte Carlo simulation in order to study the effects of a broad range of parameters on the system's dynamics. In the second stage, we use a physics-based simulator that accurately simulates robots and their interactions with an environment in order to validate the proposed collective decision-making mechanism.

5.2.1 Monte Carlo Simulation Study

Our simulation study is performed in three steps. First, we explore the effects of different parameters on the system's dynamics. In particular, we focus on the effects of different durations of the latency periods associated with each opinion and the number of teams. Next, we study the system's dynamics when the number of teams k is equal, or very close

Algorithm 6 Majority-rule opinion formation with differential latency

```
Input: Number of agents $N$, number of teams $k$, initial fraction of the population with opinion $A$.
  /* Initialization */
  $t \leftarrow 0$
  Initialize population of agents $X$.
  /* Initial team formations */
  for $i = 1$ to $k$ do
    Form team $i$ by selecting at random three non-latent agents from $X$.
    Apply the majority rule on team $i$, updating all team members' opinions.
    Team $i$ becomes latent for a period of time whose duration depends on the adopted opinion.
  end for
  repeat
    for $i = 1$ to $k$ do
      if Team $i$ becomes non-latent then
        Form new team $i$ by selecting at random three non-latent agents from $X$.
        Apply the majority rule on team $i$, updating all team members' opinions.
        Team $i$ becomes latent for a period of time whose duration depends on the adopted opinion.
      end if
    end for
    $t \leftarrow t + 1$
  until Consensus is reached
```

to the limit $N/3$. This case is interesting because, in the continous case, the probability of two teams becoming non-latent at exaclty the same time is zero. Thus, we expect that when $k = N/3$ the system will behave differently than when $k < N/3$. However, since in our simulations we use discretization of the normal distribution, the actual probability of two teams becoming non-latent at exaclty the same time is not zero. To compensate for this difference between the continuous model and its discrete implementation, we do not allow teams that happen to finish at the same time to exchange team members. Finally, we study the system's dynamics when the durations of the latency periods are such that opinions may be difficult to distinguish.

Our simulations are based on the fact that robot actions may have a typical duration with some deviation. For example, going from one place to another cannot happen instantaneously, and, depending on the number of obstacles present in the environment, one trip may take more or less time than another. Thus, as a first approximation of such a scenario, we study the system's dynamics with normally distributed latency periods using Monte Carlo simulation.

The durations of latency periods associated with opinions A and B are modeled as two normally distributed random variables with means μ_A and μ_B, and standard deviations σ_A and σ_B, respectively. The latency period duration ratio is defined as $r = \mu_B/\mu_A$. The simulation proceeds as follows: Teams are formed at random, the majority rule is applied within each team, and the resulting opinions are adopted by the involved agents. The execution times for each team are drawn from a normal distribution with the appropriate parameters and the resulting number is rounded to the nearest integer. The time steps counter runs until a team's execution time expires. At that point, a new team is formed and the process is repeated until the maximum number of time steps is reached. In our simulations, we use populations of $N \in \{9, 90, 900\}$ agents. For each population size, we vary the number of teams: $k \in \{1, 2, 3\}$, when $N = 9$, $k \in \{1, 10, 20, 30\}$, when $N = 90$, and $k \in \{1, 100, 200, 300\}$, when $N = 900$. We also vary r by changing the value of μ_B. The explored values of r are 1, 2, 3, and 4. The reference mean μ_A is fixed to a value of 100 time steps. We set $\sigma_A = \sigma_B = 20$ time steps. With these settings, the two distributions do not significantly overlap, which allows us to see the dynamics in the absence of high levels of interference. Later in this section, we study the system's dynamics when the distributions of the latency periods overlap.

Dynamics

Figure 5.4 shows the dynamics of the proposed model with a population of 900 agents.[2] The relation between the initial configuration of the population and the probability of reaching consensus on one of the alternative opinions follows the same nonlinear pattern observed in Figure 5.2(a). However, when latency periods have a different mean duration, it is more likely that the system achieves consensus on the opinion associated with the shortest latency period. This fact is reflected by a lower critical initial fraction. In Figure 5.4(a), for example, the critical initial fraction is approximately equal to 0.35 when $r = 4$ whereas it is approximately equal to 0.42 when $r = 2$. In every case, the peak on the number of team formations needed to reach consensus occurs at the critical initial fraction (see Figure 5.4(b)). Additionally, at the critical initial fraction, the larger the latency period duration ratio, the more team formations are needed to reach consensus.

A second aspect that we study in this experiment is the effect of the number of teams on the system's dynamics. An example of the obtained results is shown in Figures 5.4(c) and 5.4(d). For a latency period duration ratio greater than one, increasing the number of teams reduces the critical initial fraction. In terms of the number of team formations to achieve consensus, the results are similar to the ones observed in Figure 5.2, that is, the maximum number of team formations occurs at the critical initial fraction. As expected, when k approaches $N/3$, the system exhibits different dynamics and stops obeying the aforementioned tendencies. Except for cases in which consensus is reached after the first team formations (e.g., with very small populations and very low or large initial densities), when $N = 3k$ the system does not reach consensus (see Figure 5.4(e)). Next, we study in detail the dynamics of the system when k approaches the value $N/3$.

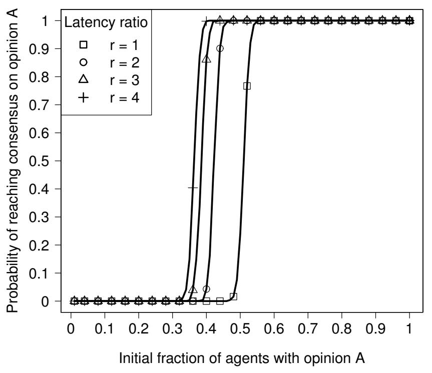
Consensus and Critical Initial Fractions

We tracked over time the proportion of latent and non-latent agents with the opinion associated with the shortest latency period (opinion A) in order to explain two phenomena: i) why the system does not always reach consensus when $k = N/3$ and ii) why, for different latency duration ratios or different number of teams, there are different critical initial fractions. Figure 5.5 shows the development of these proportions over time for three different cases: $N \in \{900, 901, 902\}$. To produce these plots, we fixed $r = 4$, $k = 300$, and the initial fraction of the population in favor of opinion A was set to 0.5.

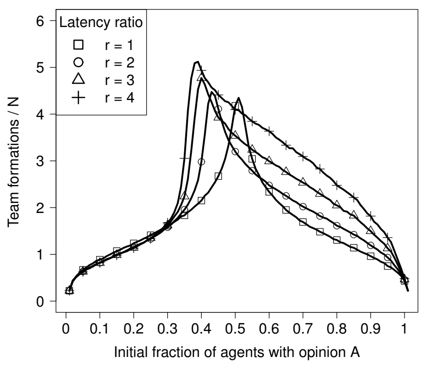
When $N = 3k$ (see Figure 5.5(a)), every time a team is destroyed and formed anew, it is composed of exactly the same members. This means that when $N = 3k$ there is no change in the number of agents with one or another opinion after the initial team formations. When $N = 3k+1$ (see Figure 5.5(b)) consensus is not reached as in the previous case. This phenomenon occurs because three of the four non-latent agents available at the moment of forming a new team have the same opinion. Thus, while there may be a different agent in a new team, the team's opinion does not change, eliminating the possibility of an eventual consensus. When $N = 3k + 2$ (see Figure 5.5(c)) the population always reaches consensus. Two non-latent agents are enough to possibly change the opinion of one agent that just switched from a latent to a non-latent state. Thus, a non-latent population of at least two non-latent agents guarantees consensus.

The "waves" depicted in Figure 5.5 are caused by the existence of two different latency periods. The valleys of the waves concur with multiples of the mean of the slowest latency period, that is, the period of these waves is μ_B. In our example, $\mu_B = 400$ because $r = 4$ and $\mu_A = 100$. The amplitude of these waves is proportional to the number of teams. These wave-like variations help explain the existence of critical initial fractions. A latency duration ratio greater than one gives, on average, more time to teams with agents with opinion A than to teams with opinion B to accumulate agents with that same opinion in the

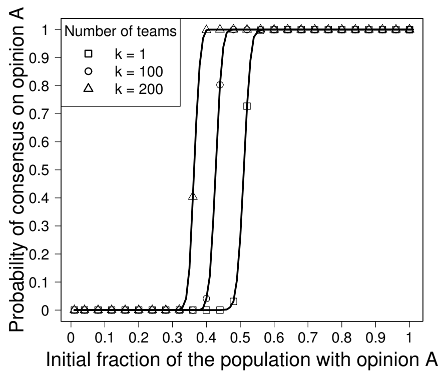
[2]In Montes de Oca (2011), the reader can find the complete set of results.



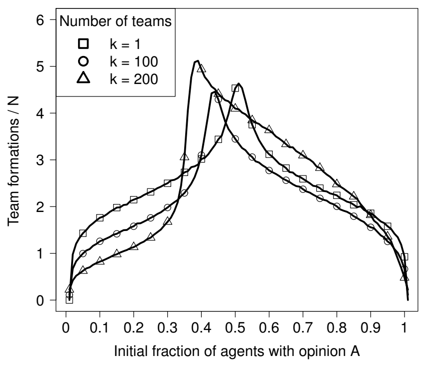
(a)



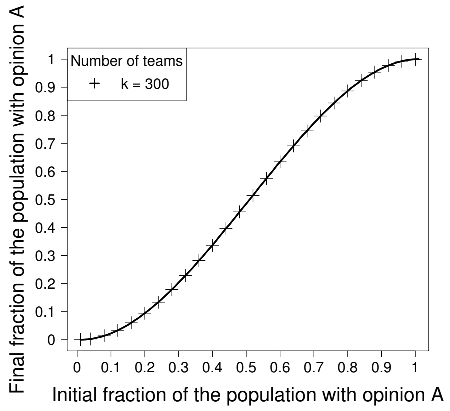
(b)



(c)



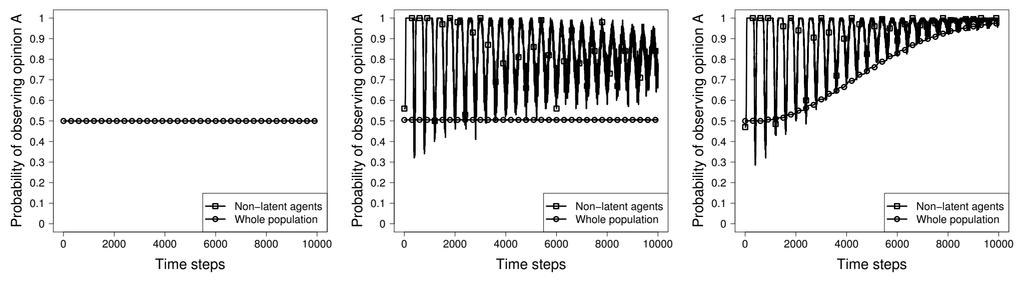
(d)



(e)

Figure 5.4: Dynamics of the majority-rule opinion formation model with normally distributed latency periods on a population of 900 agents. Figure (a) and (b) show, respectively, the probability of reaching consensus on opinion A, and the number of team formations to reach consensus for different latency period duration ratios and a fixed number of teams ($k = 200$). Figures (c) and (d) show, respectively, the probability of reaching consensus on opinion A, and the number of team formations to reach consensus for different number of teams and a fixed latency period duration ratio ($r = 4$). The plot in Figure (e) shows the case $k = N/3$ in which the system does not reach consensus. These plots are based on results obtained through 1,000 independent runs of a Monte Carlo simulation.

non-latent subpopulation. Given that $\mu_B = r\mu_A$, by the time the first teams with opinion B become non-latent, teams with opinion A will have done so approximately r times. This



(a) $N = 900$, $k = 300$ (b) $N = 901$, $k = 300$ (c) $N = 902$, $k = 300$

Figure 5.5: Development of the total proportion of agents with opinion A, and the proportion of non-latent agents with opinion A. Opinion A is associated with the shortest latency period. These plots are based on results obtained through 1,000 independent runs of a Monte Carlo simulation.

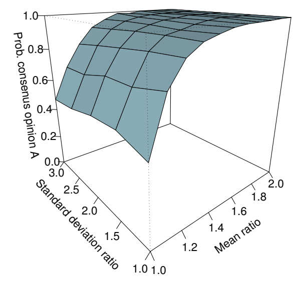
imbalance makes the population reach consensus on opinion A with higher probability than on opinion B. If the initial fraction of the population favors opinion B, that is, the initial fraction is lower than 0.5, then it is possible to balance the system. In such a situation, consensus is reached on either of the two opinions due to random fluctuations. Thus, the initial fraction that balances the opinion update process in the non-latent population is the initial critical fraction. A similar reasoning explains why the initial critical fraction decreases when the number of teams increases.

Distributions Overlap and the Discrimination Ability of the System

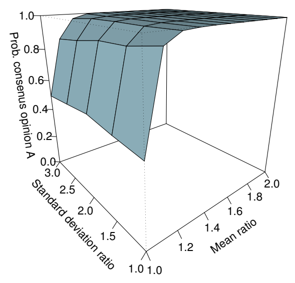
If the distributions of the duration of latency periods significantly overlap, we expect that the population of agents will not be able to consistently reach consensus on the opinion associated with the shortest latency period. Thus, it is important to assess the ability of the system to discriminate between the two distributions if the system's dynamics are to be used as a decision-making mechanism.

The following experiment is aimed at measuring the extent to which the population can still reach consensus on the opinion associated with the shortest latency period when the two latency duration distributions overlap. We assume that there is no *a priori* information about which opinion is associated with the shortest latency period. Thus, the initial fraction of agents in favor of one opinion or the other is equal to 0.5. We fix the parameters of the distribution associated with the shortest latency period (μ_A, σ_A). We vary both the mean and standard deviation of the distribution associated with the longest latency period (μ_B, σ_B). The explored ranges are: $\mu_B = r\mu_A$ with $r \in [1.0, 2.0]$ in increments of 0.1, and $\sigma_B = s\sigma_A$ with $s \in [1.0, 3.0]$ in increments of 0.5. The parameters used for the distribution associated with the shortest latency period are $\mu_A = 100$, and $\sigma_A = 10$. Other values were explored but the system does not exhibit different dynamics as long as the relations between the distributions' coefficients of variation remain the same. As discussed in Section 5.2.1, two extra non-latent agents are needed to ensure consensus. Thus, in these experiments, we increase the population size with respect to the previous experiments. The results obtained with 902 agents are shown in Figure 5.6.

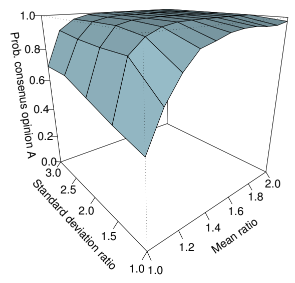
The probability of reaching consensus on the opinion associated with the shortest latency period grows more rapidly when a large number of teams and, consequently, a large population is used. For example, with 11 agents the system has great difficulties in detecting the opinion associated with the shortest latency period (results shown in (Montes de Oca, 2011)). With 11 agents, the maximum probability of reaching consensus on the opinion associated with the shortest latency period is approximately 0.8. In contrast, in the example shown in Figure 5.6, the system is able to discriminate latency periods un-



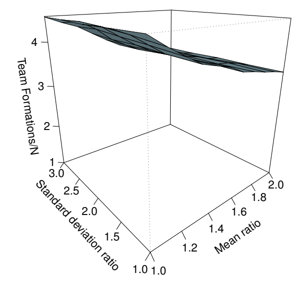
(a) $N = 902$, $k = 100$



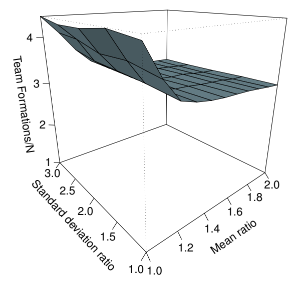
(b) $N = 902$, $k = 200$



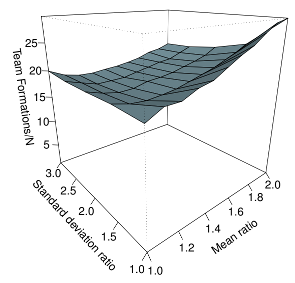
(c) $N = 902$, $k = 300$



(d) $N = 902$, $k = 100$



(e) $N = 902$, $k = 200$



(f) $N = 902$, $k = 300$

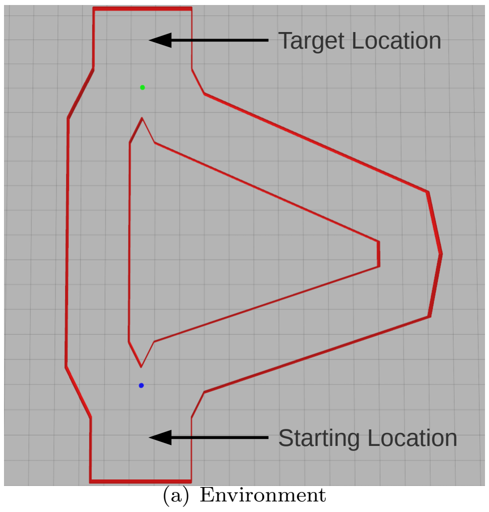
Figure 5.6: Probability of reaching consensus on the opinion associated with the shortest latency period and the average number of team formations needed to do it as a function of different levels of overlap between latency period duration distributions. These plots are based on results obtained through 1,000 independent runs of a Monte Carlo simulation.

der a wide range of combinations of means and standard deviation ratios. With 100 and 200 teams (Figures 5.6(a) and 5.6(b)), the system is mostly affected by the ratio between means. When using 200 teams, the system reaches a probability of 1 for achieving consensus on the opinion associated with the shortest latency period already from $r \geq 1.3$. At the same time, the number of team formations needed to reach consensus decreases as r increases (Figures 5.6(d) and 5.6(e)). With 300 teams (Figure 5.6(c)), the system exhibits a good discrimination ability (although not as good as with 200 teams) but at a much higher cost in terms of team formations (Figure 5.6(f)).

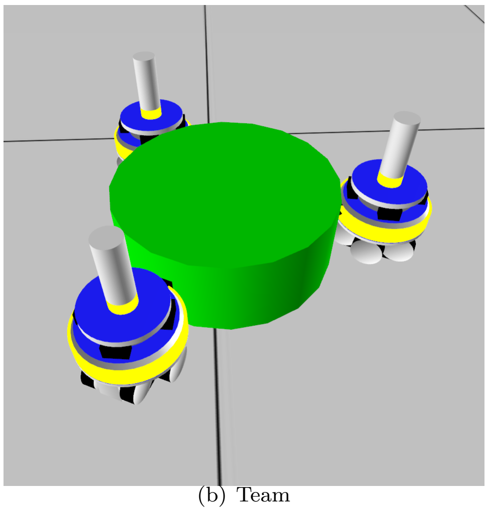
Irrespective of the size of the population, the standard deviation ratio does not have a significant impact on the probability of the system discriminating between the two distributions. We believe that this phenomenon is the result of an "averaging" effect due to the large number of team formations needed to reach a consensus. The effects of short-term fluctuations due to the high variability of one of the distributions become negligible in the long run.

5.2.2 Physics-Based Simulation Study

In the experiments described in this section, the interaction between robots and their environment determines the duration of latency periods. Moreover, the physical dimensions of the environment determines the maximum number of teams that can be used to perform a certain task. We use a scenario that resembles the well-known double bridge experiment designed by Goss et al. (1989) (see Figure 5.7(a)). The task of the robots is to transport objects from a starting location (at the bottom of the figure) to a target location (at the top



(a) Environment



(b) Team

Figure 5.7: Task Scenario. The arena is a bridge-like environment with two branches of different lengths (see Figure (a)). Teams of robots carry objects (see Figure (b)) from one end of the arena to the other. Robots must choose to take either the left or the right path.

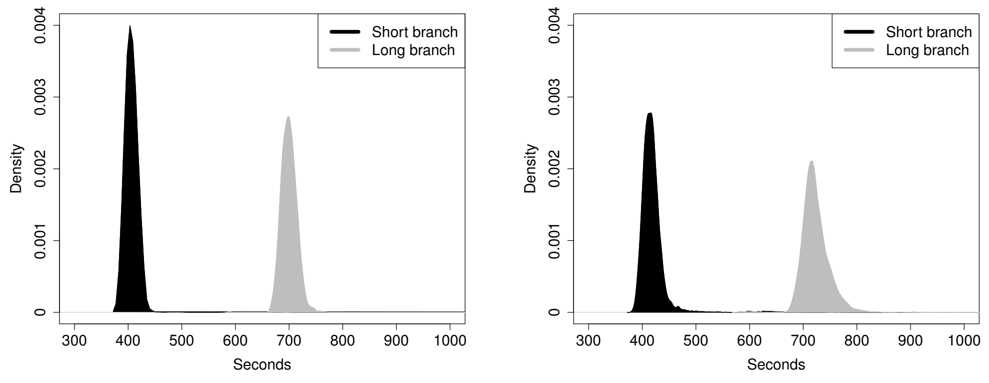
of the figure). The objects that need to be transported weigh more than what a single robot can carry. Thus, robots need to team up in order to move the objects. An assembled team ready to transport an object is shown in Figure 5.7(b). While performing this task, robots must choose to take either the left or right path to reach the target location. These two options represent the robots' "opinions." The time needed by robots to go from the starting point to the target location and back is the duration of the latency period associated with the chosen path. Robots traversing a path are latent with respect to the decision-making process because they can neither change opinion nor influence other robots to do so. On the contrary, robots that are waiting in the starting location are non-latent because they can form new teams, and thus can change or spread their opinion. Like the ants in Goss et al.'s experiment, robots do not have any knowledge about the length of the paths and do not measure distances or travel times.

Experimental Setup

We used ARGoS (Pinciroli et al., 2010), a simulator developed as part of the *SWARMANOID* project.[3] ARGoS accurately simulates physical interactions between robots and their environment. The robot models are based on the physical and electronic designs of the actual *SWARMANOID* foot-bots (Bonani et al., 2010).

In all our simulations, non-active robots are not placed inside the arena; only active robots are. The size of the environment does not allow a parallel deployment of teams. Thus, a sequential deployment strategy is adopted. From the set of non-active robots, three robots are chosen at random and placed in the starting location together with the object to be carried. These robots attach to the object using their gripper actuator. Then, the robots determine the team's majority opinion by exchanging messages using their range and bearing communication device, which allows robots to communicate locally with other robots (Roberts et al., 2009). Only robots that are located within a short range and that are in line of sight receive messages. Each robot sends its own opinion to the other two robots of the team, and once a robot receives the opinions of the others, it locally applies the majority rule to determine the opinion to adopt. Upon agreement on the path to follow, the robots start moving toward the target location. Two LEDs are placed at the bifurcations to let robot teams know in which direction they should turn. Robots detect

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



(a) Two teams (b) Ten teams

Figure 5.8: Estimated action execution time distributions for the two available actions when there are two (a) and ten (b) teams in the environment. Each density plot is based on 10,000 round trips (100 runs of 100 trips each) of a robot between the starting and goal locations in the environment shown in Figure 5.7(a).

LEDs using their omni-directional camera. When robots reach the goal area, they detach from the object they were transporting and go back, as single robots, through the same path they used when they were part of a team. On their way to the target location, robots use the collective transport controller designed by Ferrante et al. (2010). This controller allows robots to transport the object to the goal location while avoiding obstacles (walls and single robots on the way back to the starting location). Obstacles are detected using a rotating infra-red emitter and receiver. The target location is indicated by a light source located above it, which the robots perceive through their light sensors. To go back to the starting location, robots use the light source that identifies the target location as a landmark and then move away from it. To coordinate the heading direction, robots again use the range and bearing device as described by Ferrante et al. (2010). New teams are deployed every 40 simulated seconds until a specific number of teams is reached or the environment reaches its maximum capacity. The shortest branch of this environment can hold up to to ten teams.

Estimation of the Action Execution Time Distributions

In the analysis presented in Section 5.2.1, we assumed that the distributions of the latency periods are independent of the number of agents with a particular opinion. However, in a swarm robotics scenario, this assumption does not generally hold because interference between robots is likely to dynamically change the latency distributions and their ratio. In our environment, for instance, a branch could become congested if a large number of robots choose it. This increased congestion translates into longer and more variable path traversal times. To measure the effects of interference in our environment, we deploy from two to ten robot teams and make them traverse several times the environment using only one of the two branches. The estimated action execution time distributions when there are two and ten teams in the environment are shown in Figure 5.8.

The results of this experiment show that both the mean and the standard deviation of the action execution time distributions change as a result of the number of teams that choose each branch. When there are only two teams in the environment, the average time needed to traverse the left and right branches of the environment is 408.5 and 699.8 seconds, respectively. Similarly, the standard deviation is 59.3 seconds for the left path and 15.7 seconds for the right path. When there are ten teams, the average time needed to

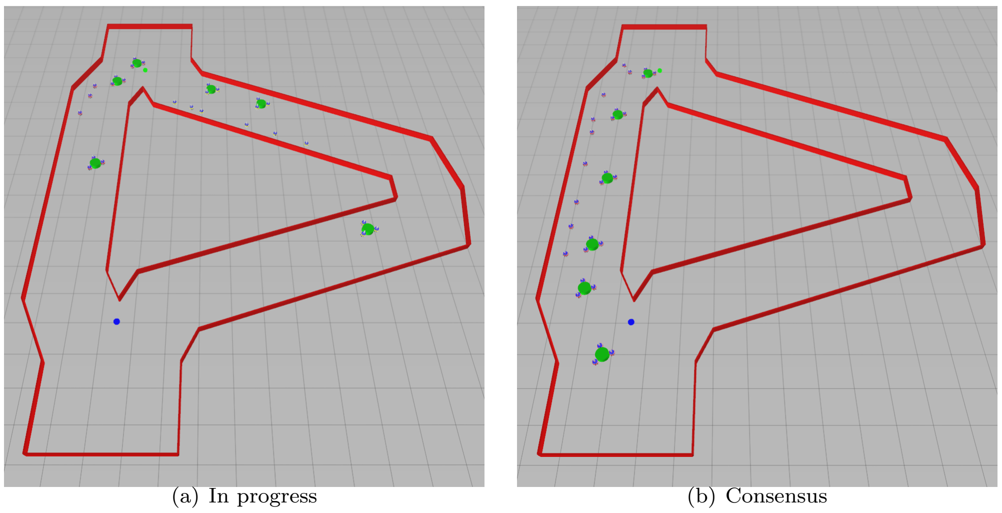


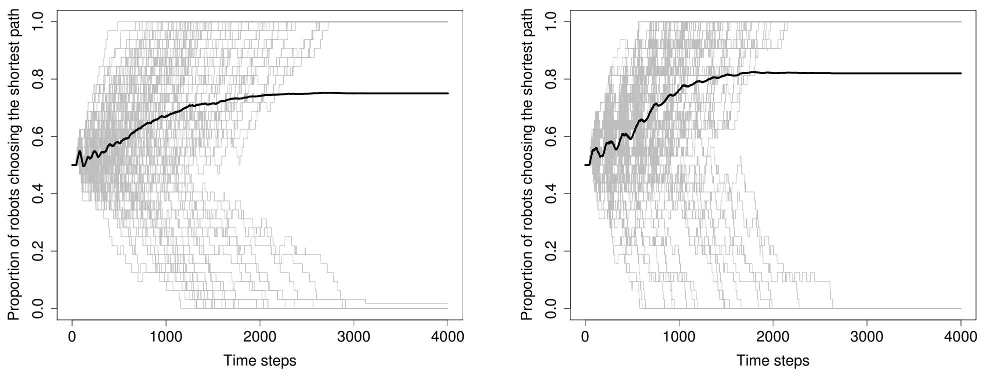
Figure 5.9: Shortest path selection process. Figure (a) shows a swarm of robots in the process of transporting objects from the starting point to the target location. Note that the robots use both branches of the environment. Figure (b) shows the state of the environment when the swarm of robots has reached consensus. The path selected by the swarm of robots is the shortest one.

traverse the left and right branches of the environment becomes 419.2 and 724.0 seconds, respectively. The standard deviation in that case becomes 29.0 seconds for the left path and 35.9 seconds for the right path. In our simulations with two agents choosing the left path, there were a few rare cases in which the time needed by a robot to perform a round trip between the starting and the target location was very long. These outliers affected the computation of the standard deviations (note that the standard deviation actually decreased when using ten teams). In our experiment, the action execution time ratio and standard deviation ratio for the two cases shown in Figure 5.8 are (1.71,0.26) for the two teams case, and (1.72, 1.23) for the ten-teams case. From two to ten teams, the mean execution ratio remained approximately the same, but the standard deviation ratio increased about five times. Additionally, the action execution distributions are right-skewed because robots that have reached the target location have to avoid collisions with incoming teams. This phenomenon occurs more frequently when the number of teams in the environment increases.

Collective Decision-Making

We now test the ability of a swarm of robots to choose the shortest path between the starting and target locations in the environment shown in Figure 5.7(a). In this experiment, the robots' decisions are governed by the dynamics of the model described in Section 5.3. We use a total of 32 robots (30 of which are executing the task at the same time plus two extra ones that are used in order to ensure consensus). The initial fraction of robots with the opinion associated with the shortest path is 0.5, that is, 16 robots initially favor the shortest path and 16 favor the longest one. In Figure 5.9, we show two snapshots of a simulation that finishes with the swarm selecting the shortest path. In the accompanying supplementary information webpage (Montes de Oca, 2011), the reader can find a video that shows the system in action.

Figure 5.10 shows two example plots of the evolution over time of the proportion of robots with the opinion associated with the shortest path. Consensus is the final state of all individual runs; however, the swarm does not reach consensus on the shortest path in all runs. The probability of reaching consensus on the shortest path depends on the number of teams deployed. In Table 5.1, we list the estimated probabilities of reaching consensus



(a) Five teams (b) Ten teams

Figure 5.10: Proportion of robots with the opinion associated with the shortest path. The evolution of the system with five and ten teams are shown in Figures (a) and (b) respectively. Single run results are shown in gray lines. The average over 100 runs is marked with a thick black line.

Table 5.1: Probability of choosing the shortest branch of the environment as a function of the number of teams k. The population size N is equal to 32 robots. The highest probability is highlighted in boldface. These results are based on statistics taken from 100 independent simulations.

	Physics-Based Simulation		Monte Carlo Simulation	
k	Probability	Avg. Team Formations	Probability	Avg. Team Formations
1	0.48	74.29	0.54	70.66
2	0.52	72.67	0.62	74.62
3	0.69	72.75	0.58	74.39
4	0.71	70.28	0.68	71.87
5	0.75	71.60	0.74	70.17
6	0.74	75.22	0.72	71.18
7	0.79	76.20	0.83	80.84
8	**0.86**	77.73	0.82	85.58
9	0.83	81.29	**0.86**	98.43
10	0.81	109.95	0.69	248.25

on the shortest path. We also include results obtained with the Monte Carlo simulator used in Section 5.2.1 for validation purposes. The simulation setup uses the data gathered in the experiment described in Section 5.2.2. Specifically, we set the mean and standard deviation of the latency period associated with the shortest path to 100 and 20 time steps, respectively. The mean of the latency period associated with the longest path was set to $1.72 \times 100 = 172$ time steps, and its standard deviation is set to $\lceil 1.23 \times 20 \rceil = 25$ time steps.

The probability of choosing the shortest path increases with the number of teams and reaches its maximum value with eight teams when using the physics-based simulator and with nine teams when using the Monte Carlo simulator. In both cases, the maximum probability is 0.86. The average team formations needed to reach consensus oscillates within the range $[70, 75]$ for most cases and grows substantially when the number of teams approaches the limit $N/3$, where N is the number of robots. These results are the consequence of three factors. First, small swarms (our 32-robot swarm can still be considered small) have difficulties in discriminating latency duration distributions whose ratio is lower than two (see Section 5.2.1). Second, as the number of teams approaches the limit $N/3$, the size of the non-latent subpopulation starts playing a role in both the quality of the decision eventually made by the swarm (lowering its quality) and the time it takes to reach consensus (increasing the number of needed team formations). Finally, the sequential deployment of teams seems to reduce the number of team formations needed to reach consensus. This phenomenon may occur because the time delay between deployments enables a better mixture of opinions in the non-latent population of robots before a new team formation.

5.3 Integration with the Incremental Social Learning Framework

The collective decision-making mechanism described in Section 5.2 potentially maximizes the amount of work performed by a swarm of robots in a given amount of time by allowing robots to select the fastest-to-execute action. However, reaching consensus on the fastest-to-execute action is a necessary but not a sufficient condition to maximize the swarm's efficiency. To maximize its efficiency, the robot swarm should also reach consensus as fast as possible. Unfortunately, the time necessary for the swarm to reach a consensus increases with the size of the population if the number of teams concurrently executing actions remains constant (see Figure 5.11). Such a situation would not be rare in environments that can hold only a certain number of teams executing a task in parallel (e.g., when the robots must travel through a corridor).

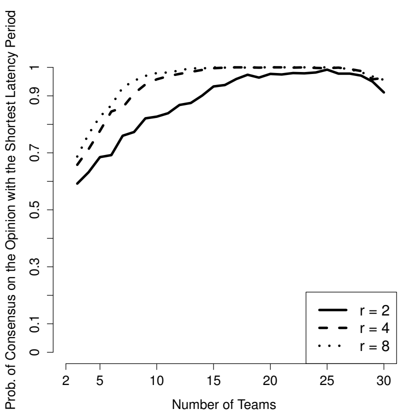
With 92 agents, the probability of reaching consensus on the opinion associated with the shortest latency period reaches the value 1.0 only with $r = 4$ and $r = 8$, and with 10 to 15 active teams. The number of team formations needed to reach consensus remains approximately the same at a value of approximately 300 team formations with up to 10 active teams. With more active teams, more team formations are needed. This number increases more rapidly with higher latency period duration ratios. For our purposes, however, the most important measure is the actual time needed to reach consensus. The number of time steps needed to reach consensus decreases rapidly as the number of active teams increases. However, past a certain value that depends on the latency period duration ratio, this time increases again. With 902 agents, the trends are similar to the ones observed with 92 agents. It is important to note that the same quality *vs.* time trade-off observed in the PSO algorithms (see Chapter 4) is observed with this system: Higher quality results, that is, reaching with high probability consensus on the opinion associated with the shortest latency period are obtained with large populations but at a higher cost in terms of the time needed to reach consensus. This characteristic trade-off makes the system suitable to be combined with the ISL framework as discussed in Chapter 3. The integration of this collective decision-making mechanism with ISL is described and evaluated in the next section.

5.3.1 Incremental Social Learning Implementation

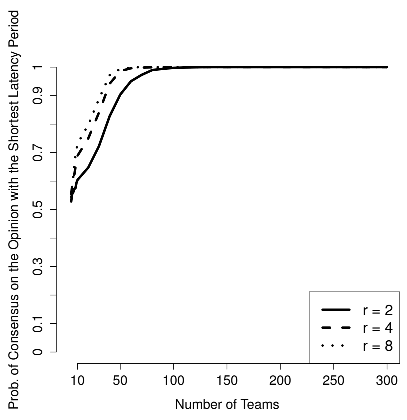
In our ISL implementation, we start with a population size $N = 6$, which means that we start with $k = 2$ teams. The reason for this choice is that the system needs at least two teams in order to detect any difference between the duration of latency periods. One team would make the population reach consensus, as demonstrated by Krapivsky and Redner (2003), but the consensus would be on a random opinion.

The agent addition schedule used is the fastest possible, that is, we add an agent to the population every time step until the maximum population size is reached. With this schedule, newly added agents are ready to form a team by the time the first team becomes non-latent. If the number of teams to build is greater than two, a new team is created as soon as there are enough free agents. Once the maximum number of teams is reached, no new teams are created even if the population is still growing.

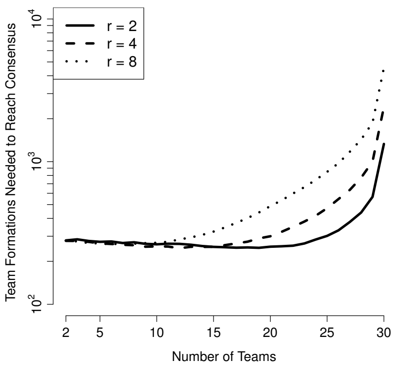
The social learning rule is implemented as follows. When a new agent is added to the population, its initial opinion is copied from one random agent chosen from the set of non-



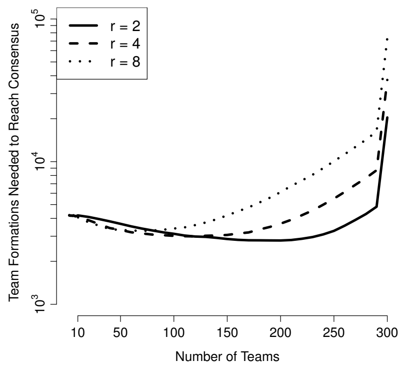
(a)



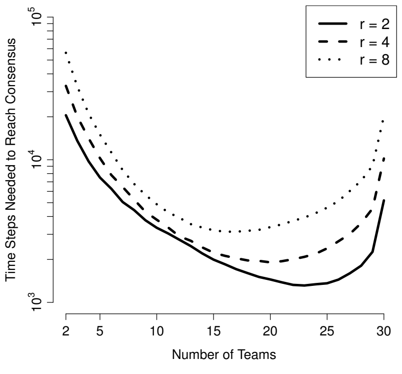
(b)



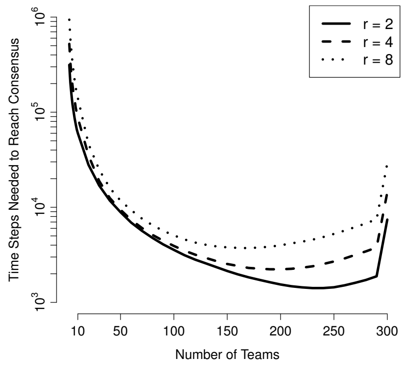
(c)



(d)



(e)



(f)

Figure 5.11: Probability of reaching consensus on the opinion associated with the shortest latency period (Figures (a) and (b)), the average number of team formations (Figures (c) and (d)), and time steps (Figures (e) and (f)) needed to reach consensus as a function of the population size and number of active teams. Figures (a), (c), and (e) correspond to the case with 92 agents. Figures (b), (d), and (f) correspond to the case with 902 agents. The data used to produce these plots were obtained through 1,000 independent runs of the Monte Carlo simulator described in Section 5.2.1. In these plots, we used three latency period duration ratios $r = 2, 4, 8$.

latent agents. If such an agent does not exist, for example, when all agents are latent, the new agent is initialized at random. A pseudo-code description of the integrated system is

Algorithm 7 Collective decision-making with incremental social learning

```
/* Initialization */
$t \leftarrow 0$
Initialize swarm $S$ with six agents /* Two teams */

/* Main loop */
while Stopping criteria not met do
  /* Agents are added according to a schedule */
  if Agent-addition criterion is not met then
    /* Execute collective decision-making mechanism as in Algorithm 6 */
  else
    Create new agent $a_{new}$
    $a_{new}$ adopts the opinion of a randomly picked non-latent agent /* Social learning */
    $\mathbf{S}^{t+1} \leftarrow \mathbf{S}^t \cup \{a_{new}\}$
  end if
  $t \leftarrow t+1$
end while
```

shown in Algorithm 7.

Evaluation Setup

Our evaluation setup is designed in order to meet the following two goals: (i) to determine whether ISL improves the performance of the collective decision-making mechanism described in Section 5.2, and if improvement is indeed achieved, (ii) to measure the magnitude of the improvement and to determine the conditions under which such an improvement occurs.

We measure the performance of the collective decision-making mechanism as the number of times agents become latent, which is equivalent to the number of times actions are executed in a given amount of time. Thus, we emphasize the amount of useful work performed by the system. Given two system settings, the one that lets agents execute more actions in the same amount of time is preferred. Additionally, we also look at the average number of times each agent in the population executes each of the two available actions. This measure allows us to observe whether ISL reduces the time agents spend trying the available alternative actions.

We use Monte Carlo simulation to carry out our experiments. As in Section 5.2.1, the durations of latency periods are modeled as two normally distributed random variables with means μ_A and μ_A, and standard deviations σ_A and σ_B, respectively. We also associate opinion A with the shortest latency period. We study the system's behavior as a function of the latency period duration ratio. Different action execution time ratios are obtained by varying μ_B. The standard deviations σ_A and σ_B are kept constant.

Two maximum population sizes are used in our simulations: $N \in \{100, 1000\}$. Different numbers of teams for each population size are used: $k_{100} \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30\}$, and $k_{1000} \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, \ldots, 90, 100, 200, 300\}$. The initial opinion of each agent is set at random. Three different values for the latency period duration ratio are tried ($r \in \{2, 4, 8\}$). The means and standard deviations of the action execution times are set as follows: $\mu_A = 10$, $\mu_B \in \{20, 40, 80\}$, and $\sigma_A = \sigma_B = 2$. This value is chosen in order to allow a clear separation of execution times between the alternative actions. Each simulation runs for 10,000 time steps. 500 simulations are run for each combination of parameters.

5.3.2 Results

The results of our simulations are reported in this section. First, we look at the relative difference of the number of times agents become latent in a given number of time steps.

Then, we look at the exploration time savings due to the use of ISL.

Amount of Work Performed

In each experiment, we count the number of times agents become latent in the original and the ISL-based system. We denote these quantities by w_{ISL} and w_{Original}, respectively. Since these quantities are in fact random variables, we use their medians, denoted by $\hat{w}_{\mathrm{ISL}}$ and $\hat{w}_{\mathrm{Original}}$, for our analysis. We then compute the medians relative difference with a normalizing factor that is equal to the expected number of times agents would become latent if the opinion associated with the shortest latency period was known from the beginning of the simulation. This number is estimated as kt/μ_A, where k is the number of active teams, t is the number of time steps (in our case, the maximum value that t can take is $T = 10000$), and μ_A is the mean of the shortest latency period. Our performance measure as a function of time is thus:

$$R_{\mathrm{ISL-Original}}(t) = \frac{\mu_A(\hat{w}_{\mathrm{ISL}} - \hat{w}_{\mathrm{Original}})}{kt}\,. \tag{5.1}$$

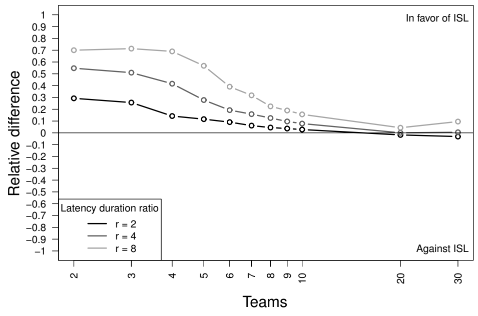
If $R_{\mathrm{ISL-Original}}(t) > 0$, then the difference is in favor of the system that uses ISL. If $R_{\mathrm{ISL-Original}}(t) < 0$, then the difference is in favor of the original system. No difference would be detected if $R_{\mathrm{ISL-Original}}(t) = 0$. The results obtained as a function of the number of active teams and latency period duration ratios are shown in Figure 5.12. We analyze the results along the following influencing factors:

- **Latency period duration ratio.** A general trend is that the greater the latency period duration ratio, the stronger the effects of ISL are on the performance of the system. This phenomenon may be due to the small population size with which the system begins. Contrary to what would happen with a constant population size system where many teams would adopt the opinion associated with the longest latency period, with ISL only one team (on average) would. If the latency period duration ratio is large, a team that adopts the opinion associated with the longest latency period does not have many chances to influence other agents once it finishes. The result is thus an accelerated convergence toward a consensus on the opinion associated with the shortest latency period.

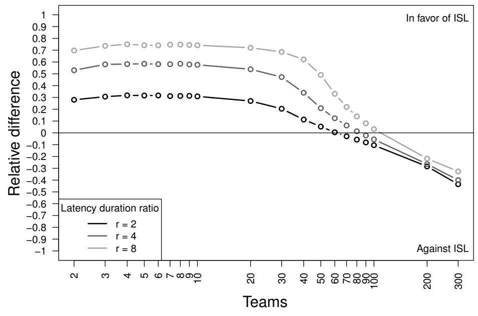
- **Number of active teams.** The effects of ISL diminish as the number of active teams increases. In fact, the differences due to different latency period duration ratios disappear when many teams are active in the environment. This result may be the consequence of the fact that increasing the number of teams in a constant population size system speeds up consensus building as seen in Figure 5.11. Nevertheless, the performance obtained by the ISL-based system is comparable to the performance of the original system.

- **Maximum population size.** The effects of ISL increase as the size of the population increases. Small populations converge rapidly as a result of the rapid amplification of fluctuations in the opinions of the population due to team formations. For example, if $N = 10$, a single team can alter the opinion of 1/10 of the population, whereas if $N = 1000$, a team can only alter the opinion of 1/1000 of the population.

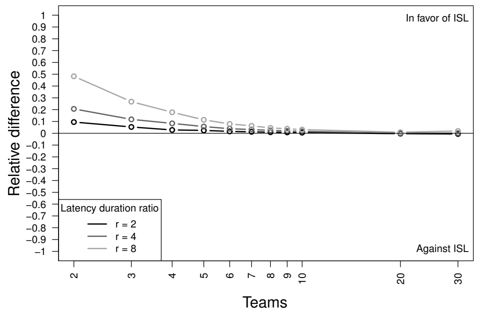
- **Available time.** The accelerated convergence that results from the application of ISL proves more useful if time constraints exist. In other words, if the time allocated for the system to perform the foraging task is limited, using the ISL framework provides benefits. This result is true even with medium-sized populations and a relatively large number of active teams.



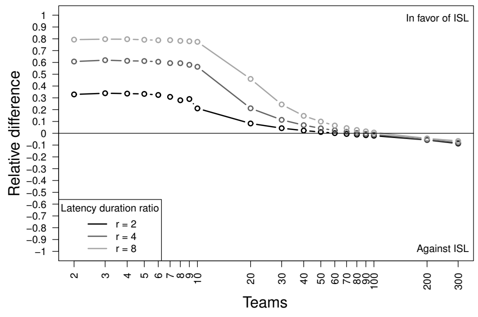
(a) N =100 after 1,000 time steps



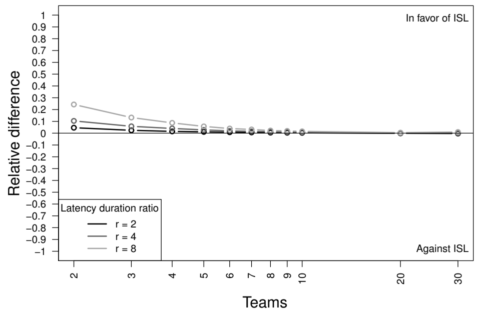
(b) N =1,000 after 1,000 time steps



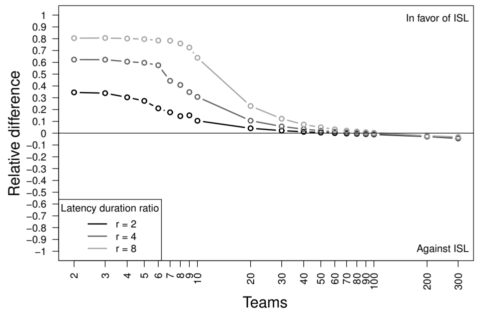
(c) N =100 after 5,000 time steps



(d) N =1,000 after 5,000 time steps



(e) N =100 after 10,000 time steps



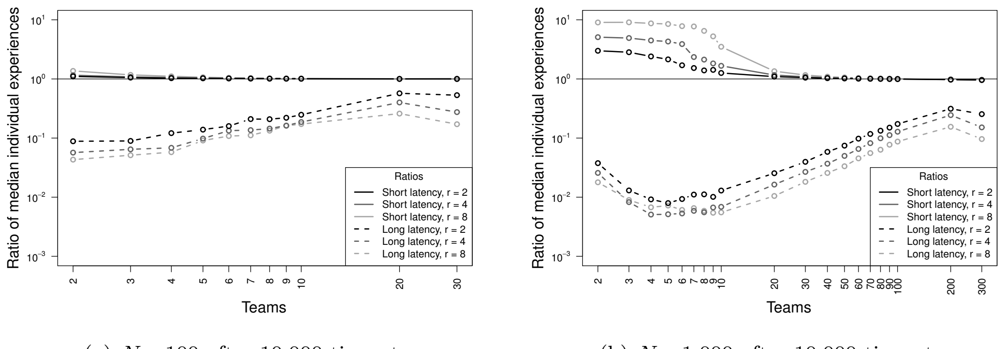
(f) N =1,000 after 10,000 time steps

Figure 5.12: Performance differences between the original collective decision-making mechanism and its combination with ISL. The normalizing factor used is the expected number of times agents would become latent if the opinion associated with the shortest latency period was chosen from the beginning of a run.

Exploration Time

As explained in Chapter 2, it is usually assumed that social learning allows agents to save time that would have otherwise been spent learning to accomplish tasks individually (Laland, 2004). As a result, social learning agents can spend more time performing more rewarding actions.

To see whether ISL allows agents to save the time otherwise needed to try the different available alternatives (that is, to learn individually), we proceed as follows. During each simulation run, we count the number of times each agent adopts each of the two available opinions. The sum of these "experiences" at the end of the simulation is then divided by



(a) N =100 after 10,000 time steps (b) N =1,000 after 10,000 time steps

Figure 5.13: Exploration *vs.* exploitation behavior. Each plot shows the ratio between the average individual experience obtained with ISL and the average individual experience obtained with a constant population size. The size of the gap between the ratios for the shortest and longest latency period is a measure of the time agents saved exploring the available opinions thanks to ISL. The solid line at ratio 1 represents the same measure but for constant population size systems.

the maximum population size. The resulting quantities can be interpreted as the average individual experience on each action, that is, the average number of times each action is tried by an agent. The difference between these quantities serve as a measure of the balance between exploration and exploitation.

To have a direct comparison between the original system and the one based on ISL, we compute the ratio of the median average individual experiences for each action and for each latency period duration ratio. The results are shown in Figure 5.13.

We conclude that ISL reduces the time spent by agents exploring the available opinions. The actual reduction depends on a number of factors including the maximum population size, the number of active teams, and the latency period duration ratio. In some cases, the reduction is substantial. For example, with a constant population size of 1000 agents and 10 active teams, agents choose the opinion associated with the longest latency period about 100 times more than agents in an ISL-based system. However, as the number of active teams increases, the advantage of using ISL is reduced.

5.4 Related Work

5.4.1 Models

In biological sciences, self-organization models have been proposed to explain the coordination of large groups of animals (Camazine et al., 2001; Couzin and Krause, 2003). Self-organization is itself the result of the interaction between several elements that include multiple direct or indirect interactions among the system's components, positive and negative feedback, and random fluctuations (Camazine et al., 2001). These models are particularly relevant for our proposal because the mechanism described in the previous section can be seen as an example of self-organization. In fact, the double-bridge experiment proposed by Goss et al. (1989) is reproduced here with the goal of pinpointing the self-organized nature of the collective decision-making mechanism introduced in this chapter. Some of the reasons that lead us to affirm this are the following: First, a large-scale spatio-temporal pattern, consensus on one branch of the environment, emerges as a result of local interactions among robots. Second, the majority rule used to control the interactions among robots does not make any reference to the pattern that emerges. Third, no single robot is capable of supervising or controlling the evolution of the system. Fourth, positive feedback occurs because robots that use the shortest path go back to the starting location before others. Thus, the probability that a new team has a majority in favor of the shortest path increases. Fifth, negative feedback is the result of the increased difficulty with which teams that adopt the opinion associated with the longest path are formed. Finally, randomness plays an important role in breaking symmetries and producing the fluctuations that are amplified by the processes described above.

In a recent study, Scheidler (2011) analyzes a simplified version of our model in which the fraction of non-latent agents is assumed to be negligible. Scheidler determines the probability of reaching consensus on the opinion associated with the shortest latency as well as the time needed to reach consensus in finite systems using Fokker-Planck equations. Moreover, an asymptotical characterization of the time to consensus is also presented.

5.4.2 Collective Decision-Making in Artificial Swarms

Many collective decision-making mechanisms in swarm robotics are based on the simulation of pheromones. Approaches range from the use of real chemicals (Russell, 1999; Fujisawa et al., 2008a,b), to the use of digital video projectors to cast images of pheromone trails on the ground (Sugawara et al., 2004; Garnier et al., 2007b; Hamman et al., 2007). There are also works in which the environment is enhanced so that it may store information. For example, Mamei and Zambonelli (2005), Herianto and Kurabayashi (2009) and Johansson and Saffiotti (2009) deploy RFID tags in the environment so that robots can read from or write in them. Mayet et al. (2010) use an environment in which the floor is covered with a paint that glows if robots activate ultraviolet LEDs. Another variant of the pheromone-inspired approach is to use actual robots as markers to form trails. Some works that use this approach are the ones by Werger and Matarić (1996); Payton et al. (2001); Nouyan et al. (2008, 2009) and Ducatelle et al. (2010). As performed to date, simulating pheromones has important limitations. For example, dealing with chemicals is problematic because very specialized sensors are needed. The level of sophistication is such that some authors have used real insects antennae (Kuwana et al., 1995; Nagasawa et al., 1999). Using video projectors is an approach that can be adopted only indoors and under controlled conditions. Furthermore, the use of video projectors requires the use of tracking cameras and a central computer to generate the images to be projected. The existence of such a central information processing unit gives the approach a single point of failure. Modifying the environment with special floors or with RFID tags is a cheap and interesting approach. However, the applicability of such an approach is limited to situations in which it is possible to design and build an environment where it is known *a priori* that robots are going to be deployed. Finally, using robots as markers allows a swarm to operate in unknown environments without central control. However, complex robot controllers are needed in order to allow individual robots to play different roles in the swarm. Although this approach is promising, the development of complex robot control software for swarms is in its infancy, since we are still trying to understand the connection between individual-level and collective-level behaviors.

Other insect behaviors have also served as sources of inspiration. For example, trophallaxis, the exchange of liquid food between insects, was first used in swarm robotics by Schmickl and Crailsheim (2008) to generate gradients through robot-to-robot communication to allow robots to find the shortest path between two locations. Gutiérrez et al. (2010) also used trophallaxis as source of inspiration for a method through which a swarm of robots can locate and navigate to the closest location of interest from a particular origin. In both of these methods, robots implicitly know that the goal is to find the shortest path between two locations. In Schmickl and Crailsheim's work, robots decrease a numerical value at a certain rate as they move. This value is communicated when there are encounters with other robots. Thus, the exchanged information gives a rough indication of the distance traveled. In Gutiérrez et al.'s work, robots actually measure the distance they have traveled and communicate this information to other robots in order to reduce the uncertainty of each robot's estimate of the location of a target. In our work, robots measure neither travel times nor distances, and still, the swarm finds the shortest path between two locations.

The aggregation behavior of cockroaches has been the source of inspiration for a site-selection mechanism with robots (Garnier et al., 2009). The nest-selection mechanism used by ants, which is based on detecting a quorum in favor of one option, has inspired the work of Parker and Zhang (2009, 2010). In these works, robots need to know whether there are enough committed robots for one of the competing options. In both cases, the more committed robots there are for one of the options, the more likely it is for a robot to commit to that option. In Garnier et al.'s work, the decision is probabilistic, and in Parker and Zhang's work, the decision depends on whether the number of committed robots is larger than a threshold. Deciding the value of this threshold or the rate at which the commitment probability increases is a critical issue because the first alternative that is identified as dominant will be the alternative chosen by the swarm. In our work, there are no thresholds or probabilities that depend on the number of robots with a specific opinion. Thus, decision-making is a continuous process that ends when the whole population reaches a consensus.

In the work of Wessnitzer and Melhuish (2003), robots use the majority rule to decide which of two "prey" to chase and immobilize. Robots capture one prey after the other. Although the decision is collective, the majority rule is used simply to break the symmetry of the decision problem.

5.4.3 Social Learning and Incremental Deployment with Robots

Work related to the integration of ISL with swarm robotics belongs to one of two categories: (i) social learning with robots, and (ii) incremental deployment of robots.

The first category has been the most productive of the two and it has been dominated by researchers interested in endowing robots with social learning capabilities so that they can naturally interact with humans. For example, in this category we can find the work of Kuniyoshi et al. (1994), Dautenhahn (1995), Billard and Dautenhahn (1999), Breazeal and Scassellati (2000), Breazeal and Scassellati (2002), Saunders et al. (2006), and Thomaz and Cakmak (2009). This kind of work is now an important aspect of the subfield of robotics research called *human-robot interaction* (Goodrich and Schultz, 2007). There is also work aimed at understanding how social learning can be exploited in multi-robot systems; however, it comprises only a small percentage of the body of literature about social learning in robotics. Matarić (1997) studied three kinds of "social reinforcement" with the goal of allowing a group of robots to learn interaction rules that reduced interference (see Chapter 3) in a foraging task. Acerbi et al. (2007) studied the influence of exploiting social cues into the effectiveness of individual and genetic (through an evolutionary computation algorithm) learning. In Acerbi et al.'s experiments, robots biased their individual learning strategy in order to induce a conformist behavior that made robots copy the behavior of other robots. This strategy proved successful in a site selection task. Pini and Tuci (2008) used artificial evolution to synthesize a neural network controller that allows a robot to use both individual and social learning in a foraging task. In their experiments, robots with the same controller can perform two different learning tasks. One of these robots learns socially from another robot that has previously learnt a task individually. Recent work has explored the relationship between a learner robot and a teacher robot. Cakmak et al. (2010) study different social learning mechanisms (see Section 2.2.1) in a two-robot scenario. The learner robot uses stimulus enhancement, mimicking, imitation, and emulation as mechanisms to exploit the information given by a demonstrator robot. In their experiments, the performance of the learner robot depends on the nature of the learning task. Thus, Cakmak et al. conclude that it might be advantageous to devise a mechanism that allows a learner robot to choose which social learning mechanism to use. One important problem that is often avoided is choosing from whom to learn. Normally, the decision is made by the experimenter. Recently, however, Kaipa et al. (2010) have tackled this problem through a self-other matching algorithm that allows a robot to choose the teacher robot based on how similar the learner and the teacher appear to be. Our work is much simpler than the ones we just mentioned. In our case, the teacher robot is a random robot. We can use this simple strategy thanks to the fact that all robots have the same capabilities and goals. Moreover, our robots copy opinions, which is equivalent to copying an observed behavior. However, copying occurs through the transfer of only one bit of information. In summary, we do not study social learning mechanisms. Instead, we exploit the effects of social learning.

Works that belong to the second category, that is, of incremental deployment of robots, are even less common than the ones dealing with social learning between robots. Howard et al. (2002) developed an incremental deployment algorithm that allowed a group of robots to cover a two-dimensional space while keeping line-of-sight contact. In a recent work, Stirling et al. (2010) show that an incremental deployment of aerial robots increases the system's energy efficiency. Both sets of authors, Howard et al. and Stirling et al., use communication between robots in order to guide the newly added robot toward its position in the environment. Their work is similar to ours in the sense that robots are deployed one at a time, and every time a new robot is deployed, information gathered by already deployed robots is exploited by the newly deployed robot.

5.5 Conclusions and Future Work

In this chapter, we introduced a collective decision-making mechanism for robot swarms that is based on the opinion dynamics induced by the majority-rule opinion formation model with differential latency. We first introduced a number of modifications to Krapivsky and Redner's and Lambiotte et al.'s majority-rule opinion formation models in order to capture some properties of real-world swarm robotics systems. Agents represent robots and opinions represent actions or sequences of actions that robots have to choose from while solving a task. One of the main modifications that we introduced is called *differential latency.* This concept models the fact that different actions that robots can perform take different amounts of time to be completed. With the proposed modifications, the population of agents reaches a consensus on the opinion associated with the shortest latency period. We demonstrated that this is the case when the duration of latency periods are normally distributed as well as when latency period distributions are the result of the interaction of the agents with their environment.

The opinion dynamics of the majority-rule opinion formation model with differential latency can be exploited in the field of swarm robotics as a self-organized collective decision-making mechanism. We believe that the proposed mechanism is promising because it enables a swarm of robots to make a decision that from an observer's point of view is intelligent without requiring intelligent individual decision makers. As an example of the potential of the new approach, we tested it on a scenario based on the well-known double-bridge experiment. The results of this experiment clearly show that through the proposed mechanism, a swarm of robots is able to find the shortest path between two locations without simulating pheromones or requiring robots to measure distance or time.

We observed that when large populations are involved, the time necessary for the system to reach consensus may make it impractical for some applications. We tackled this problem by integrating the proposed collective decision-making mechanism with ISL. By starting with a small population and increasing its size over time, the system converges faster. The social learning rule allows new agents to learn from more experienced ones, thus saving exploration time. Our simulation results show that through the application of ISL, the performance of the decision-making mechanism can be substantially improved in situations where a small fraction of the population concurrently tries the different available alternatives and when time constraints exist. This result is very positive because in many situations, reducing the number of active agents without sacrificing the amount of work performed may allow the spared agents to perform other tasks.

We believe future work should focus on the improvement of the collective decision-making mechanism in order to facilitate its use on real robotics tasks and its combination with other approaches. Some of the challenges and opportunities that the proposed mechanism offers are:

- In the proposed scenario, robots know exactly when they need to form teams and make local decisions. However, in many situations the environment is unknown to the robots. Thus, it is difficult for a robot to know its position and therefore, when it should form a team. In our experiments, teams are deployed in the team assembly area in order to avoid this problem. Additionally, we used an LED to mark the decision point. Future work should address these shortcomings.

- In the proposed scenario, robots know the number of available alternatives. We tested only the case of two opinions, but cases in which there are more than two opinions should definitely be explored. An even more flexible approach would be to let robots discover the number of available actions they can choose from as they interact with the environment.

- An interesting research direction could be the integration of opinion dynamics with task allocation methods in order to tackle problems for which consensus is a suboptimal solution.

- If the environment changes after the system has reached a consensus, the population cannot adapt. This problem could be tackled if some fixed number of robots do not change opinion. We are exploring this direction in ongoing work.

- In our work, the opinion dynamics that allow the swarm to reach consensus on one opinion are based on time-related "rewards." Thus, the proposed approach is useful when the desired collective decision is the one associated with the shortest execution time. However, there are problems for which the best collective decision is based on more qualitative aspects. Translating these qualitative aspects into latencies of different duration would be a first approach toward a more general collective decision-making mechanism. For example, if an object is more interesting than another, robots that prefer the more interesting object should spend less time in a non-latent state than the other robots. Consequently, a positive feedback process could favor that option.

- The decision quality depends on the population size. Large populations usually make better decisions. While such a property is desirable in swarm robotics systems, it also hinders its use in real robotics systems because the promise of having thousands of cheap robots has not been met yet. Thus, research is needed in order to improve the decision quality when the size of the swarm is relatively small. An option, that we are currently studying, is to simulate large swarms by making robots remember their past over long time horizons (not of just one action execution as it is currently done) and make a decision based on the opinion that has been observed more often during that period.

- The proposed approach requires robots to form teams and execute actions together. However, in some situations (e.g., when the physical dimensions of the environment does not allow robots to move together), forming teams might not be possible. Thus, a collective decision-making mechanism that works with single robots is desirable. A first attempt toward this goal is reported in (Montes de Oca et al., 2009a).

To conclude, we believe that collective decision-making in swarms based on opinion formation models is a new and exciting research direction with the potential of cross-pollinating the fields of swarm intelligence and statistical physics. On the one hand, the field of swarm intelligence may greatly benefit from ideas and tools developed in statistical physics literature. On the other hand, physicists may regard swarm intelligence as a rich source of interesting problems waiting to be modeled and solved.

Chapter 6

Conclusions and Future Work

In this chapter, we summarize the results and contributions presented in this dissertation. We also offer some ideas for future work that we believe can contribute to the further development of the swarm intelligence field.

6.1 Swarm Intelligence Systems and Interference

Swarm intelligence is the problem-solving behavior of large groups of simple entities capable of autonomous perception and action (called agents) that collectively are referred to as a *swarm*. The term swarm intelligence evokes a mental image in which a large group of insect-like entities congregates and exhibits a purposeful behavior without a central authority to supervise the actions of each individual or issue commands to govern the group's behavior. Despite its name, which makes us recall science fiction works, swarm intelligence exists in nature. Bees form swarms to collectively find and choose the best location to build a new home. Ant colonies, which can be composed of millions of ants, build complex nests, search and retrieve food, maintain the young, etc. In each case, a swarm intelligence system performs a particular task without any single individual supervising or directing the actions of other members of the swarm. Swarm intelligence can also be the product of engineering efforts. Powerful optimization techniques and control mechanisms for groups of mobile robots have been designed exploiting swarm intelligence principles.

Artificial swarm intelligence systems are composed of numerous agents that interact locally with one another and with their environment. Through different mechanisms, but predominantly through self-organization and decentralized control, these kinds of systems exhibit a collective intelligence that allows them to solve problems that their constituent agents cannot solve individually. As in any system whose constituent agents interact with each other, there are interactions among the agents that form a swarm that reduce the efficiency of the system. These interactions are collectively referred to as *interference*. One of the most visible effects of interference in a swarm intelligence system is the reduction of the system's efficiency; that is, the time required by the system to reach a desired state is increased. Interference increases with the size of the population of agents. Thus, interference is a major problem in swarm intelligence systems since many of them require large populations to perform their tasks satisfactorily. Interference is thus a fundamental problem inherent to systems composed of many agents because it negatively affects the viability of the swarm intelligence approach when solving important practical problems.

6.2 Incremental Social Learning as a Mechanism for Reducing the Effects of Interference

In this dissertation, an original framework called incremental social learning (ISL) was proposed in Chapter 3. This framework aims to reduce the negative effects of interference in swarm intelligence systems. Two components form the core of the ISL framework. The first component directly manipulates one of the factors that causes interference: the number of agents that compose a swarm. A swarm intelligence system under the control of ISL starts with a small population. Gradually, the population grows until the system performs as desired or a maximum number of agents is reached. The second component of ISL is social learning. ISL exploits the fact that learning socially is less costly, in terms of trial-and-error trials for an individual, than asocial learning. Through social learning, newly added agents acquire knowledge from agents that have been part of the swarm for some time. As a result of the combination of these two components, a growing population size, and social learning, the effects of interference is reduced. Consequently, the swarm reaches a desired state more rapidly than without using ISL.

6.2.1 Incremental Social Learning in Particle Swarms

We demonstrated the effectiveness of ISL through two case studies. In the first case study, presented in Chapter 4, we applied ISL to particle swarm optimization (PSO) algorithms. These algorithms are commonly used to tackle continuous optimization problems and are composed of a population of searching agents called *particles*. PSO algorithms with a constant population size exhibit a trade-off between solution quality and number of objective function evaluations amenable to the application of ISL. With a small population size, the solution quality improves rapidly during the first objective function evaluations until it reaches a stable value. With large populations, the same solution quality reached by a small population is reached after many more objective function evaluations. However, if more evaluations are allowed, a better solution quality may be reached. The hypothesis that supports the application of ISL to PSO algorithms is that the trade-off between solution quality and number of objective function evaluations is due, at least partially, to interference among particles. Interference in PSO algorithms is the result of particles being attracted toward the best solutions found by other particles. Interference is large in big swarms because at the beginning of the optimization process too much information flows through the network of particles. This phenomenon makes particles spend objective function evaluations in regions that do not contain the optimal solution.

As a result of the application of ISL to PSO algorithms, three new PSO variants were designed. The first one, which serves as a basis for the other two, is an incremental particle swarm optimization algorithm that we call IPSO. In IPSO, the population of particles grows over time until the optimization process returns a solution of acceptable quality or until a maximum population size is reached. The rate at which particles are added to the system is scheduled and controlled through a parameter. Each time a new particle is added, its position in the objective function's domain (usually a subset of $\mathbb{R}^n$) is generated through a rule that biases the placement of the new particle toward the best-so-far solution. Through a thorough experimental evaluation, we could show how IPSO, with the appropriate setting of the population growth, could return solutions that are comparable to those that would be returned if multiple PSO algorithms with different constant population sizes were run in parallel and only the best solution found by any of those algorithms was returned. The other two algorithms that result from the use of ISL on PSO algorithms, called IPSOLS and IPSOLS+, repeatedly call a local search procedure from a particle's best found position in order to intensify the search. Each call of the local search procedure could be seen as simulating the individual learning of a particle. IPSOLS works in the same way as IPSO with an added step that consists in calling a local search procedure from each particle's best found position. IPSOLS+ is a further refinement of IPSOLS in which the local search is called more frequently from the particle's position that represents the best-so-far solution and in which the PSO rules are modified. IPSOLS's performance is comparable with state-of-the-art PSO algorithms. IPSOLS+'s performance is comparable with state-of-the-art algorithms for large-scale continuous optimization problems.

6.2.2 Incremental Social Learning in Robot Swarms

In the second case study, presented in Chapter 5, we applied ISL to a collective decision-making mechanism for swarms of mobile robots. Two contributions are presented in that chapter. First, the collective decision-making itself, and second, the application of ISL to that mechanism. For this case study, we chose a foraging task that involves group transport in an arena that consists of two locations connected by two paths. During the execution of the task, which is to transport objects from one location to the other, a swarm of robots must choose one of the two paths. In one of the two locations, robots form teams to transport the objects, which cannot be transported by single robots. From that location, teams of robots transport objects to the other location. Robots have a preferred branch (encoded as an "opinion") and whenever they form a team, they advocate for their preferred path. The final team's decision is that of the local majority. Robots not only choose that path, but they also change their preference if it is different from the one they had before forming a team. After making a decision, a team moves from one location to the other using the chosen path. Once a team arrives at the target location, it disassembles and its component robots return as individuals using again the chosen path. Once they arrive at the initial location, robots can form new teams, repeating the process until the task is performed. The length of the paths induce a *latency* period during which robots can neither change opinion nor influence other robots. Thus, each opinion has a latency period whose duration depends on the length of the paths and on the number of robots in the environment. We showed through Monte Carlo and physics-based simulations that the dynamics of the system makes a swarm of robots reach a consensus. If the initial distribution of opinions in the swarm is such that half of the swarm prefers one opinion and the other half prefers the other opinion, the proposed collective decision-making mechanism makes the swarm reach consensus with high probability on the opinion associated with the shortest latency period. In the robotics setting described in Chapter 5, this means that a swarm reaches consensus on the opinion associated with the shortest path.

The aforementioned swarm robotics system shows a trade-off between performance and population size similar to the one observed in PSO algorithms. In this case, however, it is the population of "idle" robots, that is, those robots that are not engaged in the transportation task, that affects the system's performance. Our implementation of ISL manipulates this population. We start the process with only six robots (two teams). At each time step, we add a robot and let it copy the opinion of a randomly picked "idle" robot. If there are no robots to copy from, the opinion of the new robot is initialized at random. Because of the dynamics of the system, it is more likely for a new robot to copy the opinion associated with the shortest path. As a result, the population reaches a consensus on the opinion associated with the shortest path in fewer time steps than it would without ISL. The effectiveness of ISL, however, depends on the number of active robots in the environment. With more active teams, there are fewer "idle" robots, and thus, the effects of ISL diminish to the point at which there is practically no difference between the system that is using ISL and the system that is no using ISL.

6.2.3 Impact

One of the major challenges in swarm intelligence research is to design agent-level behaviors in order to obtain a certain desired behavior at the collective-level. Since a general methodology for achieving this goal has been elusive, most researchers in the field concentrate their efforts on specific applications. In doing so, a number of assumptions are made. One of these assumptions is that the size of a swarm of agents remains constant over time. In many cases, this assumption may not be well justified.

The framework proposed in this dissertation challenges the constant population size assumption. In the ISL framework, the population size changes over time and we have demonstrated that some benefits can be obtained with such an approach. As seen in Chapter 5, we are not the only ones to realize that an incremental deployment of agents (robots) can bring benefits and can even simplify the design of the agent-level behaviors. In fact, in many practical applications of swarm intelligence systems, in particular in swarm robotics and affine fields, such as sensor networks, it is actually more difficult to deploy hundreds of robots at once, than to deploy a few robots at different points in time. For example, consider the deployment of the 30 Galileo satellites (European Space Agency, 2010). It is not reasonable to assume that tens of satellites can be deployed at once. Rather, the deployment is painfully slow, with one or two satellites being deployed at a time. If these satellites were part of a swarm of satellites with specific tasks such as maintaining a formation in space, the rules needed to fulfill that task would be quite complex. Instead, with an incremental deployment, each satellite could take its position without disturbing the behavior of other satellites. In other words, the interference between satellites would be greatly reduced.

With our proposal, we hope that researchers in the swarm intelligence field will consider the possibility of an incremental deployment of agents in the design of new swarm intelligence systems.

The other aspect of our proposal, the use of some form of social learning, can potentially have a bigger impact in the field. Social learning can be the mechanism that enables the appearance of a form of cumulative "culture" in a swarm that passes from one "generation" of agents to another. A continuous process of addition and elimination of agents can make this process possible as long as the knowledge acquired during the lifetime of one agent is not lost, but is instead transmitted to a new agent. This new agent in turn would have time to accumulate more knowledge to pass on to another agent, and so on. Perhaps the biggest impact of this idea will be in the field of swarm robotics, in which each robot has a lifetime determined by the capacity of its batteries. Before running out of power, a robot could pass on its knowledge to another fully charged robot, which will have more time to refine and accumulate more information.

6.3 Future Work

We believe that the work presented in this dissertation opens a number of potentially fruitful research avenues. In the remainder of this section, we will briefly describe some of them. Our presentation is divided in two parts. In the first, we describe future work that is directly related to the ISL framework. In the second part, future work derived from the two case studies presented in this dissertation is proposed.

6.3.1 Future Work Related to the Incremental Social Learning Framework

Theory

Interference has been identified by some authors, notably Matarić (1997); Helbing and Vicsek (1999) and Gershenson (2007), as an influence that we need to control in order to be able to design large multiagent and self-organizing systems. Unfortunately, very little theoretical work that could help us understand how to do that has been performed. Future work in this area, we believe, could significantly impact swarm intelligence, self-organizing systems, complex systems, and other related fields.

Throughout this dissertation, we have given empirical evidence of the effectiveness of the ISL framework. However, we have not determined analytically the conditions under which the ISL framework is guaranteed to reduce interference. Future work should be directed toward achieving this goal as this would increase the impact of the proposed approach.

More Applications

The performance of the optimization algorithms presented in Chapter 4 suggests that the ISL framework can improve the performance of other swarm intelligence-based optimization algorithms. In fact, in a recent paper, we explored the application of the ISL framework to an ant colony optimization algorithm for continuous optimization problems and obtained promising results (Liao et al., 2011). Another related and potentially fruitful research direction is the application of the ISL framework to evolutionary algorithms such as differential evolution (Storn and Price, 1997) or CMA-ES (Hansen et al., 1995). However, it should be noted that these algorithms' search dynamics are different from the search dynamics of swarm intelligence algorithms. Thus, even though it is straightforward to apply ISL to these algorithms, and that the two classes of algorithms share some common features, such as a population of candidate solutions, the results of the application of ISL to evolutionary algorithms may be different from the results obtained with swarm intelligence algorithms.

In swarm robotics, more studies about the possible use and benefits of using the ISL framework should be undertaken. In particular, it would be interesting to follow and build on the work of Winfield and Griffiths (2010) who are investigating how a "robotic culture" could emerge. The ISL framework could play the role of a knowledge transfer facilitator between "generations" of robots in those settings.

6.3.2 Future Work Related to the Case Studies

Tuning-in-the-loop Design of Optimization Algorithms

In Chapter 4, we described the redesign process of IPSOLS that led to IPSOLS+. This process relied on a parameter tuning tool, iterated F-Race, as a way to measure the impact of each important design decision. The result of this process was a highly competitive algorithm in the field of large-scale continuous optimization. We believe that a methodology that integrates parameter tuning tools as part of the optimization algorithm design process can have an important role in the emerging field of engineering stochastic local search algorithms (Stützle et al., 2007, 2009).

Collective Decision-Making Mechanisms based on Opinion Formation Models

The majority-rule opinion formation model which is at the basis of the collective decision-making mechanism introduced in Chapter 5 is only one of a large number of opinion-formation models that have been proposed in the statistical physics literature (Castellano et al., 2009). Considering the promising results that we were able to obtain, we believe that the swarm intelligence field could greatly benefit if more researchers consider using similar methods to address scenarios in which agents must choose among multiple choices. Also of interest is the study of the resulting systems' dynamics in changing environments. Domains in which the agents agree on a continuous quantity instead of on a discrete one should also be explored.

6.4 Concluding Statement

In this dissertation, we have introduced the incremental social learning framework. Its design is aimed at reducing interference in systems composed of many interacting agents. To show its potential, we instantiated the framework in the context of particle swarm optimization algorithms and swarm robotics. The results obtained represent evidence that the framework indeed reduces interference, which in turn makes the systems have a better performance.

We hope that these results motivate other researchers interested in multiagent systems, swarm intelligence, and other affine fields, to integrate the incremental social framework into a set of agent deployment strategies. Such a set of strategies can indeed simplify the agent design process because, as we demonstrated, by reducing the levels of interference, it is possible to simplify the rules that govern agent interactions.

Appendices

Appendix A

Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm

Since the first PSO algorithm was introduced, many variants have been proposed. In many cases, the difference between two variants can be seen as an algorithmic component being present in one variant but not in the other. In the first part of this appendix, we present the results and insights obtained from a detailed empirical study of several PSO variants from a component difference point of view. We then describe a new PSO algorithm that combines a number of algorithmic components that showed distinct advantages in the experimental study concerning optimization speed and reliability. We call this composite algorithm *Frankenstein's PSO* in an analogy to the popular character of Mary Shelley's novel. Frankenstein's PSO performance evaluation shows that by integrating components in novel ways effective optimizers can be designed.

A.1 Compared PSO Algorithms

In this section, we describe the variants that were selected to be part of our study. For practical reasons, many variants had to be left out; however, the selection allows the study of a number of different PSO algorithmic components including those that, for us, are among the most influential or promising ones. In the description of the algorithms, we use the notation used in Chapters 2 and 4.

A.1.1 Time-Varying Inertia Weight Particle Swarm Optimizers

Shi and Eberhart (1998a, 1999) noticed that the first term of the right hand side of Eq. 2.3 plays the role of a particle's "inertia" and they introduced the idea of an *inertia weight*. The velocity-update rule was modified to

$$v_{i,j}^{t+1} = w^{\,t} v_{i,j}^{t} + \varphi_1 U_1 (pb_{i,j}^{t} - x_{i,j}^{t}) + \varphi_2 U_2 (lb_{i,j}^{t} - x_{i,j}^{t})\,, \tag{A.1}$$

where $w^{\,t}$ is the time-dependent inertia weight. Shi and Eberhart proposed to set the inertia weight according to a time-decreasing function so as to have an algorithm that initially explores the search space and only later focuses on the most promising regions. Experimental results showed that this approach is effective (Shi and Eberhart, 1998b,a, 1999). The function used to schedule the inertia weight is defined as follows:

$$w^{\,t} = \frac{wt_{\max} - t}{wt_{\max}} (w_{\max} - w_{\min}) + w_{\min}\,, \tag{A.2}$$

where $wt_{\max}$ marks the time at which $w^{\,t} = w_{\min}$; $w_{\max}$ and $w_{\min}$ are the maximum and minimum values the inertia weight can take, respectively. Normally, $wt_{\max}$ coincides with the maximum time allocated for the optimization process. We identify this variant as *decreasing-IW* PSO. The constricted PSO is a special case of this variant but with a constant inertia weight. We treat them as different variants because of their different behavior and for historical reasons.

Zheng et al. (2003b,a) experimented with a time-increasing inertia weight function obtaining, in some cases, better results than the decreasing-IW variant. Concerning the schedule of the inertia weight, Zheng et al. also used Eq. A.2, except that the values of $w_{\max}$ and $w_{\min}$ were interchanged. This variant is referred to as *increasing-IW* PSO.

Eberhart and Shi (2001) proposed a variant in which an inertia weight vector is randomly generated according to a uniform distribution in the range [0.5,1.0) with a different inertia weight for each dimension. This range was inspired by Clerc and Kennedy's constriction factor because the expected value of the inertia weight in this case is $0.75 \approx 0.729$ (see Section 4.4). Accordingly, in this *stochastic-IW* PSO algorithm, acceleration coefficients are set to the product of $\chi \cdot \varphi_i$ with $i \in \{1, 2\}$.

A.1.2 Fully Informed Particle Swarm Optimizer

Mendes et al. (2004) proposed the fully informed particle swarm (FIPS), in which a particle uses information from all its topological neighbors. Clerc and Kennedy's constriction factor is also adopted in FIPS; however, the value φ (i.e., the sum of the acceleration coefficients) is equally distributed among all the neighbors of a particle.

For a given particle i, φ is decomposed as $\varphi_k = \varphi/|\mathcal{N}_i|$, $\forall k \in \mathcal{N}_i$. The velocity-update equation becomes

$$v_{i,j}^{t+1} = \chi \left[v_{i,j}^{t} + \sum_{k \in \mathcal{N}_i} \varphi_k U_k (pb_{k,j}^{t} - x_{i,j}^{t}) \right] . \tag{A.3}$$

A.1.3 Self-Organizing Hierarchical Particle Swarm Optimizer with Time-varying Acceleration Coefficients

Ratnaweera et al. (2004) proposed the self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients (HPSOTVAC), in which the inertia term in the velocity-update rule is eliminated. Additionally, if any component of a particle's velocity vector becomes zero (or very close to zero), it is reinitialized to a value proportional to V_{max}, the maximum velocity allowed. These changes give the algorithm a local search behavior that is amplified by linearly adapting the value of the acceleration coefficients φ_1 and φ_2. The coefficient φ_1 is decreased from 2.5 to 0.5 and the coefficient φ_2 is increased from 0.5 to 2.5. In HPSOTVAC, the maximum velocity is linearly decreased during a run so as to reach one tenth of its value at the end of a run. A low reinitialization velocity near the end of a run allows particles to move slowly near the best region they have found. The resulting PSO variant is a kind of local search algorithm with occasional magnitude-decreasing unidimensional restarts.

A.1.4 Adaptive Hierarchical Particle Swarm Optimizer

The adaptive hierarchical PSO (AHPSO) (Janson and Middendorf, 2005) modifies the neighborhood topology at run time. This algorithm uses a tree-like topology structure in which particles with better objective function evaluations are located in the upper nodes of the tree. At each iteration, a child particle updates its velocity considering its own previous best performance and the previous best performance of its parent. Before the velocity-update process takes place, the previous best objective function value of a particle is compared with the previous best objective function value of its parent. If the comparison is favorable to the child particle, child and parent swap their positions in the hierarchy. Additionally, AHPSO adapts the branching degree of the tree while solving a problem in order to balance the exploration-exploitation behavior of the algorithm: a hierarchy with a low branching degree has a more exploratory behavior than a hierarchy with a high

Table A.1: Benchmark Functions

Name	Definition	Search Range
Ackley	$-20e^{-0.2\sqrt{1/n\sum_{i=1}^{n}x_i^2}} - e^{1/n\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e$	$[-32.0, 32.0]^n$
Griewank	$\frac{1}{4000}\sum_{i=1}^{n}x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600.0, 600.0]^n$
Rastrigin	$10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$	$[-5.12, 5.12]^n$
Salomon	$1 - \cos(2\pi\sqrt{\sum_{i=1}^{n}x_i^2}) + 0.1\sqrt{\sum_{i=1}^{n}x_i^2}$	$[-100.0, 100.0]^n$
Schwefel (sine root)	$418.9829n + \sum_{i=1}^{n} -x_i\sin(\sqrt{\lvert x_i\rvert})$	$[-512.0, 512.0]^n$
Step	$6n + \sum_{i=1}^{n}\lfloor x_i\rfloor$	$[-5.12, 5.12]^n$
Rosenbrock	$\sum_{i=1}^{n-1}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$[-30.0, 30.0]^n$
Sphere	$\sum_{i=1}^{n}x_i^2$	$[-100.0, 100.0]^n$

branching degree. In AHPSO, the branching degree is decreased by k_{adapt} degrees (one at a time) until a certain minimum degree d_{min} is reached. This process takes place every f_{adapt} number of iterations. For more details, see (Janson and Middendorf, 2005).

A.2 Experimental Setup

The complete experimental design examines five factors:

1. **PSO algorithm**. This factor considers the differences between PSO variants. Specifically, we focused on (i) different strategies for updating inertia weights, (ii) the use of static and time-varying population topologies, and (iii) different strategies for updating a particle's velocity.

2. **Problem**. We selected some of the most commonly used benchmark functions in experimental evolutionary computation. Since most of these functions have their global optimum located at the origin, we shifted it to avoid any possible search bias as suggested by Liang et al. (2005). In most cases, we used the shift values proposed in the set of benchmark functions used for the special session on real parameter optimization of the IEEE CEC 2005 (Suganthan et al., 2005). Table A.1 lists the benchmark functions used in our study. In all cases, we used their 30-dimensional versions, that is, $n = 30$. All algorithms were run 100 times on each problem.

3. **Population topology**. We use three of the most commonly used population topologies: The fully connected topology, in which every particle is a neighbor of any other particle in the swarm; the von Neumann topology, in which each particle is a neighbor of 4 other particles; and the ring topology, in which each particle is a neighbor of another 2 particles. In our setup, all particles are also neighbors to themselves. These three topologies are tested with all variants except in the case of AHPSO which uses a time-varying topology. The selected topologies provide different degrees of connectivity between particles. The goal is to favor exploration in different degrees: The less connected is a topology, the more it delays the propagation of the best-so-far solution. Thus, low connected topologies result in more exploratory behavior than highly connected ones (Mendes, 2004). Although recent research suggests that random topologies can be competitive with predefined ones (Mohais et al., 2005), they are not included in our setup in order not to have an unmanageable number of free variables.

4. **Population size**. We considered three population sizes: 20, 40 and 60 particles. With low connected topologies and large populations, the propagation of information is slower and thus it is expected that a more "parallel" search takes place. The

Table A.2: Parameter settings

Algorithm	Settings
Constricted	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.05$. Constriction factor $\chi = 0.729$. Maximum velocity $V_{max} = \pm X_{max}$, where X_{max} is the maximum of the search range.
Decreasing-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.0$. Linearly-decreasing inertia weight from 0.9 to 0.4. The final value is reached at the end of the run. Maximum velocity $V_{max} = \pm X_{max}$.
Increasing-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.0$. Linearly-increasing inertia weight from 0.4 to 0.9. The final value is reached at the end of the run. Maximum velocity $V_{max} = \pm X_{max}$.
Stochastic-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 1.494$. Uniformly distributed random inertia weight in the range $[0.5, 1.0]$. Maximum velocity $V_{max} = \pm X_{max}$.
FIPS	Acceleration parameter $\varphi = 4.1$. Constriction factor $\chi = 0.729$. Maximum velocity $V_{max} = \pm X_{max}$.
HPSOTVAC	Acceleration coefficient φ_1 linearly decreased from 2.5 to 0.5 and coefficient φ_2 linearly increased from 0.5 to 2.5. Linearly decreased reinitialization velocity from V_{max} to $0.1 \cdot V_{max}$. Maximum velocity $V_{max} = \pm X_{max}$.
AHPSO	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.05$. Constriction factor $\chi = 0.729$. Initial branching factor is set to 20, d_{min}, f_{adapt}, and k_{adapt} were set to 2, $1000 \cdot m$, and 3 respectively, where m is the number of particles.

configurations of the von Neumann topologies for 20, 40 and 60 particles were, respectively, 5×4, 5×8 and 6×10 particles. The population is initialized uniformly at random over the ranges specified in Table A.1. Since the problems' optima were shifted, the initialization range is asymmetric with respect to them.

5. **Maximum number of function evaluations**. This factor determined the stopping criterion. The limit was set to 10^6 function evaluations. However, data were collected during a run to determine relative performances for shorter runs. The goal was to find variants that are well suited for different application scenarios. The first two cases (10^3 and 10^4 function evaluations) model scenarios in which there are scarce resources and the best possible solution is sought given a restrictive time limit. The other two cases (10^5 and 10^6 function evaluations) model scenarios in which the main concern is to find high quality solutions without paying too much attention to the time it takes to find them.

In our experimental setup, each algorithm was run with the same parameter settings across all benchmark problems. When possible, we use the most commonly used parameter settings found in the literature. These parameter settings are listed in Table A.2.

In our experimental analysis, we examined the algorithms' performance at different levels of aggregation. At a detailed level, we analyze the algorithms' qualified run-length distributions (RLDs, for short). At a more aggregate level, we use the median solution quality reached by the algorithms at different stopping criteria. The most important elements of the RLD methodology are explained below (for a detailed exposition, see (Hoos and Stützle, 2004)).

The number of function evaluations a stochastic optimization algorithm needs to find a solution of a certain quality on a given problem can be modeled as a random variable. Its associated cumulative probability distribution $RL_q(l)$ is the algorithm's RLD, defined as

$$RL_q(l) = P(L_q \leq l)\,, \tag{A.4}$$

where L_q is the random variable representing the number of function evaluations needed to find a solution of quality q, and $P(L_q \leq l)$ is the probability that L_q takes a value less than or equal to l function evaluations. Theoretical RLDs can be estimated empirically using multiple independent runs of an algorithm.

An empirical RLD provides a graphical view of the development of the probability of finding a solution of a certain quality as a function of time. When this probability does not increase or it does but very slowly, the algorithm is said to stagnate. In this appendix we use the word "stagnation" to refer to the phenomenon of slow or no increment of the probability of finding a solution of a specific quality. Note that no reference to the state of the optimization algorithm is implied (e.g. in active search or otherwise).

In stagnation cases, the probability of finding a solution of a certain quality may be increased by restarting the algorithm at fixed cut-off times without carrying over information from the previous runs (Hoos and Stützle, 2004). These *independent* restarts entail re-running the algorithm using a different random seed. However, the output of the algorithm with restarts is always the overall best-so-far solution across all independent runs.

The RLD of the algorithm with periodic restarts will approximate, in the long run, an exponential distribution. However, independent restarts can be detrimental if an algorithm's original RLD grows faster than an exponential distribution. Given an algorithms RLD, it is possible to estimate the number of function evaluations needed for finding a solution of required quality with a probability greater than or equal to z supposing an optimal restart policy. This estimation is sometimes called *computational effort* (Niehaus and Banzhaf, 2003) and is defined as

$$\mathit{effort} = \min_{l} \left\{ l \cdot \frac{ln(1-z)}{ln(1-RL_q(l))} \right\}. \tag{A.5}$$

We use this measure to account for the possibility of restarting the compared algorithms with optimal restart policies.

Another measure that will be used in the description of the results is the *first hitting time* H_q for a specific solution quality q. H_q is an estimation of the minimum number of evaluations that an algorithm needs for finding a solution of a quality level q. It is defined as

$$H_q = \min\{l \geq 0; RL_q(l) > 0\}\,. \tag{A.6}$$

A.3 Performance Comparison of Particle Swarm Optimization Algorithms

The comparison is carried out in three phases. In the first one, a problem-dependent run-time behavior comparison based on RLDs is performed. In the second phase, data from all the problems of our benchmark suite are aggregated and analyzed. In the third phase, we study the effects of using different inertia weight schedules on the performance of the concerned variants. Results that are valid for all the tested problems are explicitly summarized.

A.3.1 Results: Run-Length Distributions

The graphs presented in this section show a curve for each of the compared algorithms corresponding to a particular combination of a population topology and a population size.

Since AHPSO does not use a fixed topology, its RLDs are the same across topologies and its results can therefore be used as a reference across plots for a same problem. The RLDs we present here were obtained using swarms of 20 and 60 particles.

We present only one representative example of the results we obtained. Fig. A.1 shows some of the algorithms' RLDs when solving Griewank's function.[1] These plots are given with respect to a bound of 0.001% above the optimum value, corresponding to an absolute error of 0.0018. The smallest first hitting times for the same algorithm across different population size and topology settings are obtained with a population size of 20 and the fully connected topology. Conversely, the largest ones are obtained with a population size of 60 and the ring topology. With 20 particles, the right tails of the RLDs show a slowly-increasing or a non-increasing slope. This means that, for the Griewank's function, all the PSO variants included in our study, when using 20 particles and the parameter settings shown in Table II, have a strong stagnation tendency. In fact, no variant is capable of finding a solution of the required quality with probability 1.0 with this population size. With 60 particles and a ring topology, only FIPS finds the required solution quality with probability 1.0, while the constricted PSO and HPSOTVAC reach a solution of the required quality with probability 0.99.

Result 1: *Depending on the problem and required solution quality, PSO algorithms exhibit a stagnation tendency with different degrees of severity. This tendency is smaller when using large population sizes and/or low connected topologies than it is when using small population sizes and/or highly connected topologies; however, even though the probability of solving the problem increases, first hitting times are normally delayed.*

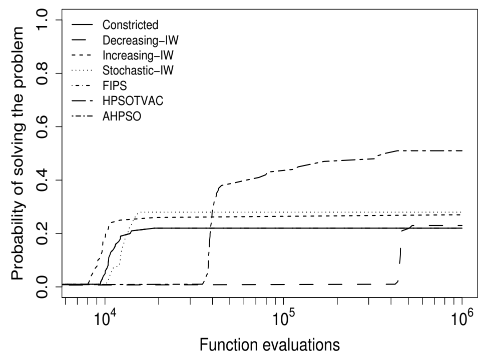
An interesting fact is the strong influence of the topology on the algorithms' performance. For example, FIPS with a fully connected topology does not find a single solution of the required quality; however, with a ring topology, it is among the fastest algorithms (in terms of first hitting time). AHPSO seems to profit from a highly connected topology at the beginning of a run. It is also among the fastest variants when the rest of the algorithms use a von Neumann or ring topology. However, it is unable to solve the problem with a high probability.

Result 2: *PSO algorithms are sensitive to changes in the population topology in different degrees. Among those tested, FIPS is the most sensitive variant to a change of this nature. On the contrary, HPSOTVAC and the decreasing inertia weight PSO algorithm are quite robust to topology changes.*

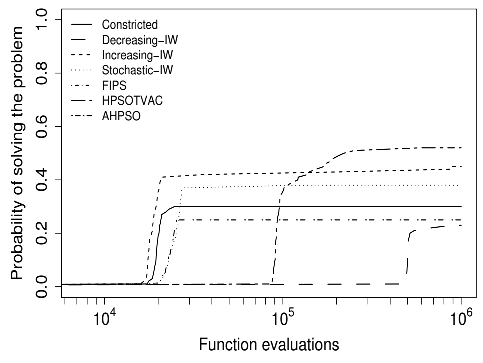
As a best-case analysis, we now consider the possibility of restarting the algorithms with an optimal cut-off period. In Table A.3, we show the best configuration of each algorithm to solve Griewank's problem (at 0.001% above the global optimum) with probability 0.99. The best performing configurations of FIPS and the constricted PSO, both with 60 particles and the ring topology, do not benefit from restarts under these conditions, and they are the two best variants for the considered goal. In this case, the joint effect of choosing the right algorithm, with an appropriate population size and with the right topology, cannot be outperformed by configurations that benefit the most from restarts (i.e., those that stagnate). Similar analyses were performed on all the problems of our benchmark suite but different results were obtained in each case.

Result 3: *Independent restarts can improve the performance of various PSO algorithms. In some cases, configurations that favor an exploitative behavior can outperform those that favor an exploratory one if optimal restart policies are used. However, the optimal restart policy is algorithm- and problem-dependent and therefore cannot be defined a priori.*

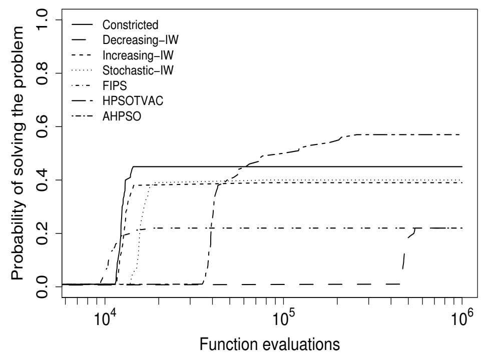
[1]The complete set of results can be found at Montes de Oca et al. (2007).



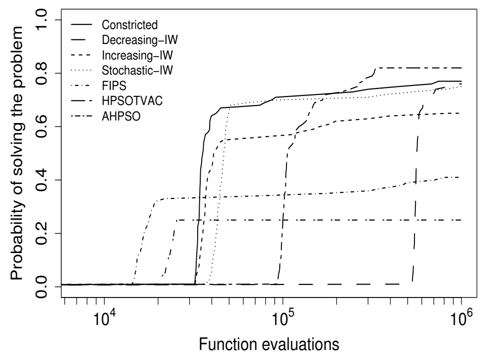
(a) 20 particles, Fully connected topology



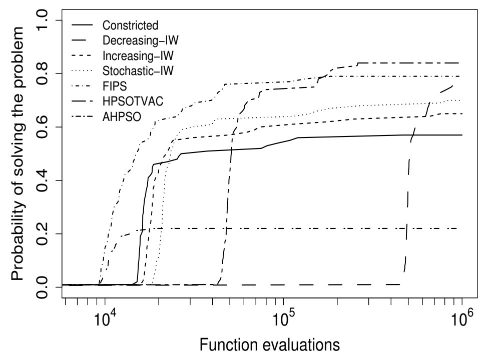
(b) 60 particles, Fully connected topology



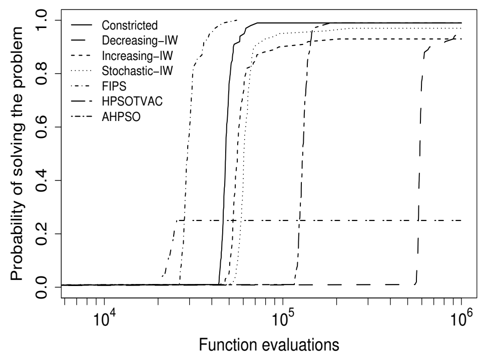
(c) 20 particles, von Neumann topology



(d) 60 particles, von Neumann topology



(e) 20 particles, Ring topology



(f) 60 particles, Ring topology

Figure A.1: RLDs on the Griewank function. The solution quality bound is set to 0.001% above the global optimum (equivalent to an absolute error of 0.0018). Plots (a), (c), and (e) in the left column show the RLDs obtained with 20 particles. Plots (b), (d), and (f) in the right column show the RLDs obtained with 60 particles. The effect of using different population topologies can be seen by comparing plots in different rows. The effect of using a different number of particles can be seen by comparing columns.

Table A.3: Best performing configurations of each algorithm using independent restarts on Griewank's function[1, 2]

Algorithm	Pop. Size	Topology	Cut-off	Effort	Restarts
FIPS	60	Ring	46440	**46440**	0
Constricted	60	Ring	71880	71880	0
Sto-IW	40	Ring	52160	131075	2
Inc-IW	20	Ring	24040	138644	5
HPSOTVAC	40	Ring	132080	155482	1
AHPSO	40	Dynamic	17360	207295	11
Dec-IW	60	Ring	663000	1326000	1

[1] Probabilities taken from the RLDs.
[2] Cut-off and effort measured in function evaluations. The effort is computed using Eq. A.5.

A.3.2 Results: Aggregated Data

The analysis that follows is based on the median solution quality achieved by an algorithm after some specific number of function evaluations. This analysis considers only the 40 particles case, which represents the intermediate case in terms of population size in our experimental setup. For each problem, we ranked 19 configurations (6 PSO algorithms $\times$ 3 topologies + AHPSO) and selected only those that were ranked in the first three places (what we call the top-three group). For this analysis, we assume that the algorithms are neither restarted nor fine-tuned for any specific problem.

Table A.5 shows the distribution of appearances of the compared PSO algorithms in the top-three group. The table shows configurations ranked among the three best algorithms for different numbers of function evaluations (FES). The topology used by a particular configuration is shown in parenthesis. If two or more configurations found solutions with the same quality level (differences smaller than 10^{-15} are not considered) and they were among the three best solution qualities, these configurations were considered to be part of the top-three group. In fact, we observed that, as the number of function evaluations increases, more and more algorithms appear in the top-three group. This indicates that the difference in the solution quality achieved by different algorithms decreases and that many algorithms find solutions of the same quality level.

Table A.4 shows the algorithms that most often appear in the top-three group in Table A.5 for different termination criteria. The column labeled "Σ" shows the total number of times each algorithm appeared in the top-three group. The rightmost column shows the distribution of appearances in the top-three group between multi- and unimodal functions.

Table A.4: Best PSO variants for different termination criteria

Budget (in FES)	Algorithm(Topology)	Σ	multi/unimodal
10^3	Inc-IW(F), FIPS(F,vN)	6	5/1
10^4	Inc-IW(F)	7	6/1
10^5	Constricted(vN)	5	4/1
10^6	Dec-IW(vN), FIPS(R)	6	5/1

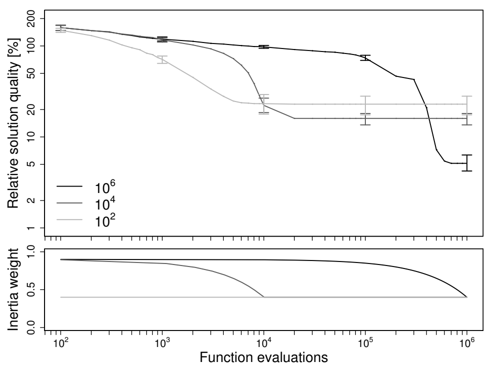
Note that the connectivity of the topology used by the best ranked variants decreases as the maximum number of function evaluations increases. Note also that FIPS is among the best ranked variants: for the shortest runs, using a fully-connected or a von Neumann topology and, for the longest runs, using a ring topology. Even though these results may seem counterintuitive at first inspection, they can be understood by looking at the convergence behavior of the algorithm when topologies of different connectivity degree are used. In FIPS, highly connected topologies induce a strongly convergent behavior that, depending on the features of the objective function, can result in a very fast solution improvement during the first iterations (Montes de Oca and Stützle, 2008a). Indeed, it has been shown that under stagnation, the moments of the sampling distribution of FIPS become more and more stable (over time) as the topology connectivity increases (Poli, 2007). This means that in FIPS the more connected the population topology, the lower the stochasticity in the behavior of a particle. By observing the behavior of FIPS over different run lengths, our results extend those of Mendes (2004) who studied the behavior of FIPS using only a fixed number of function evaluations as stopping criterion.

Result 4: *When a limited number of function evaluations are allowed, configurations that favor an exploitative behavior (i.e., those with highly connected topologies and/or low inertia weights) obtain the best results. When solution quality is the most important aspect, algorithms with exploratory properties are the best performing.*

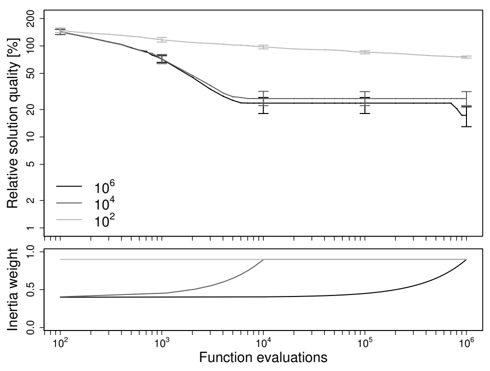
A.3.3 Results: Different Inertia Weight Schedules

With very few exceptions, e.g., (Wang and Wang, 2004), the change of the inertia weight value in the time-decreasing/increasing inertia weight variants is normally scheduled over the whole optimization process. Here we present a study on the effects of using different schedules on both the time-decreasing and time increasing inertia weight variants. To do so, we modified the inertia weight schedule, which is based on Eq. A.2, so that whenever the inertia weight reaches its limit value, it remains there. We experimented with five inertia weight schedules of $wt_{\max} \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$ function evaluations each. The remaining parameters were set as shown in Table A.2.

As an example of the effects of different inertia weight schedules, consider Fig. A.2, which shows the development of the solution quality over time (using both the time-decreasing and time-increasing inertia weight variants) for different inertia weight schedules on the Rastrigin function.



(a) Decreasing inertia weight



(b) Increasing inertia weight

Figure A.2: Solution quality and inertia weight development over time for different inertia weight schedules on the Rastrigin function. The solution quality development plots are based on the medians of the algorithms' RLDs. The first and third quartiles are shown at selected points. These results correspond to configurations of 20 particles in a fully connected topology. The results obtained with the schedules of 10^5 and 10^3 function evaluations (not shown) are intermediate with respect to the results obtained with the other schedules.

In the case of the time-decreasing inertia weight variant, slow schedules ($wt_{\max} = 10^5$ or 10^6 function evaluations) perform poorly during the first phase of the optimization process; however, they are the ones that are capable of finding the best quality solutions. On the other hand, fast schedules ($wt_{\max} = 10^2$ or 10^3 function evaluations) produce rapid improvement but at the cost of stagnation later in the optimization process.

With the time-increasing inertia weight variant, slow schedules provide the best performance. Fast schedules make the time-increasing inertia weight variant strongly stagnant. For both variants, the severity of the stagnation tendency induced by different schedules is alleviated by both an increase in the number of particles and the use of a low connected topology.

Table A.5: Distribution of appearances of different PSO algorithms in the top-three group[1]

FES	Ackley	Griewank	Rastrigin	Salomon	Schwefel	Step	Rosenbrock	Sphere
10^3	FIPS (F,vN) Inc-IW (F)	FIPS (F,vN) Inc-IW (F)	FIPS (F,vN) Inc-IW (F)	FIPS (F,vN) HPSOTVAC	Inc-IW (F,vN,R)	FIPS (F,vN) Inc-IW (F)	AHPSO Constricted (F) Sto-IW (F)	FIPS (F,vN) Inc-IW (F)
10^4	FIPS (vN,R) Inc-IW (F)	Constricted (F) FIPS (vN) Inc-IW (F)	AHPSO Constricted (F) Inc-IW (F)	Constricted (F) Inc-IW (F) Sto-IW (F)	AHPSO Inc-IW (F) Sto-IW (F)	AHPSO Constricted (F) Inc-IW (F) Sto-IW (F)	AHPSO Constricted (F) Sto-IW (F)	AHPSO Constricted (F) Inc-IW (F)
10^5	Constricted (vN) FIPS (R) Inc-IW (F)	Constricted (vN,R) FIPS (R) Inc-IW (vN,R) Sto-IW (vN,R)	FIPS (vN) Inc-IW (vN) Sto-IW (vN)	Constricted (vN,R) FIPS (R) Inc-IW (F,vN) Sto-IW (F,vN,R)	HPSOTVAC (F,vN,R)	Constricted (vN) Inc-IW (F) Sto-IW (F)	AHPSO Constricted (F) Sto-IW (F)	AHPSO Constricted (F,vN,R) FIPS (R) Inc-IW (F,vN,R) Sto-IW (F,vN,R)
10^6	Constricted (vN,R) Dec-IW (F,vN,R) FIPS (R) Inc-IW (vN,R) Sto-IW (vN,R)	Constricted (vN,R) Dec-IW (vN,R) FIPS (R) HPSOTVAC (F,vN,R) Inc-IW (vN,R) Sto-IW (vN,R)	HPSOTVAC (F,vN,R)	Constricted (vN,R) Dec-IW (F,vN,R) FIPS (R) HPSOTVAC (F,vN,R) Inc-IW (vN,R) Sto-IW (vN,R)	Dec-IW (vN) FIPS (R) HPSOTVAC (R)	Constricted (vN,R) Dec-IW (F,vN,R) FIPS (R) HPSOTVAC (F,vN,R) Inc-IW (F,vN,R) Sto-IW (F,vN,R)	AHPSO Constricted (F) Sto-IW (F)	AHPSO Constricted (F,vN,R) Dec-IW (F,vN,R) FIPS (R) HPSOTVAC (vN) Inc-IW (F,vN,R) Sto-IW (F,vN,R)

[1] F, vN and R stand for fully connected, von Neumann and ring, respectively. FES stands for function evaluations.

Result 5: *By varying the inertia weight schedule, it is possible to control the convergence speed of the time-varying inertia weight variants. In the case of the time-decreasing inertia weight variant, faster schedules induce a faster convergence speed, albeit at the cost of increasing the algorithm's stagnation tendencies. In the time-increasing inertia weight variant, slow schedules provide the best performance both in terms of speed and quality.*

A.3.4 Summary

The goal of the comparison presented in this section was to identify algorithmic components that provide good performance under different operating conditions (specially run-lengths). The five main results give insight into what factors should be taken into account when trying to solve effectively a problem using a PSO algorithm.

Among other results, we have seen that the stagnation tendency of PSO algorithms can be alleviated by using a large population and/or a low connected topology. Another approach to reduce stagnation in some cases is to use restarts. However, optimal restart schedules are algorithm- and problem-dependent and determining them requires previous experimentation. We have also seen how different inertia weight schedules affect the performance of the time-decreasing/increasing inertia weight variants.

A.4 Frankenstein's Particle Swarm Optimization Algorithm

Insights gained from experimental studies ideally guide toward the definition of new, better performing algorithms. In this section, a composite algorithm called *Frankenstein's PSO* is assembled from algorithmic components that are taken from the PSO algorithms that we have examined or that are derived from the analysis of the comparison results.

A.4.1 The Algorithm

Frankenstein's PSO is composed of three main algorithmic components, namely (i) a time-varying population topology that reduces its connectivity over time, (ii) FIPS's mechanism for updating a particle's velocity, and (iii) a decreasing inertia weight. These components are taken from AHPSO, FIPS and the time-decreasing inertia weight variant, respectively. The first component is included as a mechanism for improving the trade-off between speed and quality associated with topologies of different connectivity degrees. The second component is used because the analysis showed that FIPS is the only algorithm that can outperform the others using topologies of different connectivity degree (see Table A.4). Finally, the decreasing inertia weight component is included as a mean to balance the exploration-exploitation behavior of the algorithm.

The time-varying topology starts as a fully connected one and, as the optimization process evolves, decreases its connectivity until it ends up being a ring topology. Interestingly, it is the opposite approach than the one taken by Suganthan (1999). Note, however, that our approach is entirely based on the results of the empirical analysis presented in the previous section. Specifically, our choice is based on the fact that a highly connected topology during the first iterations gives an algorithm the opportunity to find good quality solutions early in a run (see Table A.4 and Results 1 and 4 in Section A.3). The topology connectivity is then decreased, so that the risk of getting trapped somewhere in the search space is reduced and, hence, exploration is enhanced. Including this component into the algorithm allows it to achieve good performance across a wider range of run lengths as it will be shown later. As we said before, this component is taken from AHPSO. Information flow in AHPSO is very fast during the first iterations because the topology connectivity is high. As the optimization process evolves, its connectivity decreases.

In Frankenstein's PSO, we do not use a hierarchical topology as it is not clear from our results how it contributes to a good performance. Instead, the topology is changed

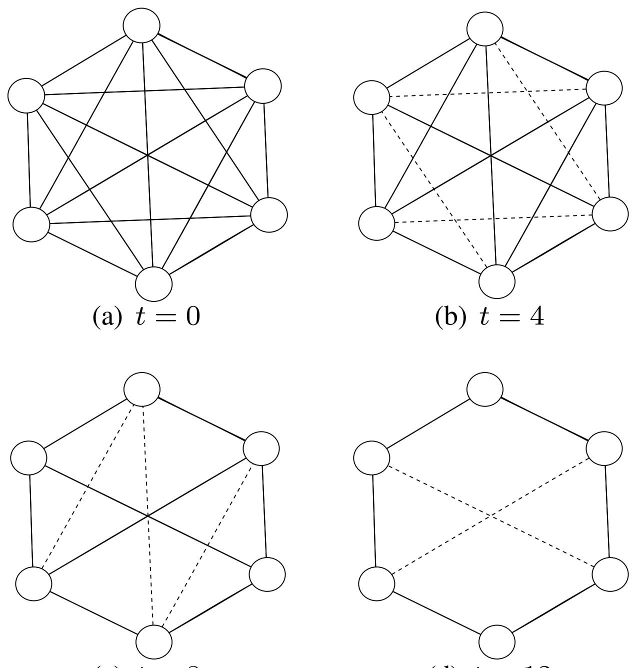


Figure A.3: Topology change process. Suppose $n = 6$ and $k = 12$. Then, every $\lceil 12/(6-3) \rceil = 4$ iterations we remove some edges from the graph. In $6-3 = 3$ steps, the elimination process will be finished. (a). At $t = 0$ a fully connected topology is used. (b). At $t = 4$ the $6-2 = 4$ edges to be removed are shown in dashed lines. (c). At $t = 8$ the $6-3 = 3$ edges to be removed are shown in dashed lines. (d). At $t = 12$ the remaining $6-4 = 2$ edges to be removed are shown in dashed lines. From $t = 12$ on, the algorithm uses a ring topology.

as follows. Suppose we have a particle swarm composed of n particles. We schedule the change of the topology so that in k iterations (with $k \geq n$), we transform a fully connected topology with $n(n-1)/2$ edges into a ring topology with n edges. The total number of edges that have to be eliminated is $n(n-3)/2$. Every $\lceil k/(n-3) \rceil$ iterations we remove m edges, where m follows an arithmetic regression pattern of the form $n-2, n-3, \ldots, 2$. We sweep m nodes removing one edge per node. The edge to be removed is chosen uniformly at random from the edges that do not belong to the exterior ring, which is predefined in advance (just as it is done when using the normal ring topology). The transformation from the initially fully connected to the final ring topology is performed in $n-3$ elimination steps. Fig. A.3 shows a graphical example of how the process just described is carried out.

Changes in the population topology must be exploited by the underlying particles' velocity-update mechanism. In Frankenstein's PSO we included the mechanism used by FIPS. The reason for this is that we need a component that offers good performance across different topology connectivities. According to Table A.4, the only velocity-update mechanism that is ranked among the best variants when using different topologies is the one used by FIPS. For short runs, FIPS's best performance is obtained with the fully connected topology (the way Frankenstein's PSO topology starts); for long runs, FIPS reaches very high performance with a low connected topology (the way Frankenstein's PSO topology ends).

The constriction factor originally used in FIPS is substituted by a decreasing inertia weight. A decreasing inertia weight was chosen because it is a parameter that can be used to control the algorithm's exploration/exploitation capabilities. In Section A.3.3, we saw that a proper selection of the inertia weight schedule can dramatically change the performance of a PSO algorithm. A decreasing inertia weight would counterbalance the exploratory behavior that the chosen topology change scheme could induce.

The pseudocode of Frankenstein's PSO is shown in Algorithm A.4.1. The main loop cycles through the three algorithmic components: topology update, inertia weight update, and the particles' velocity and position updates. The topology update mechanism is only executed while the algorithm's current number of iterations is lower than or equal to a parameter k, which specifies the topology update schedule. Since it is guaranteed that the ring topology is reached after iteration k, there is no need to call this procedure thereafter. In Algorithm 1, a variable *esteps* is used to ensure that the number of eliminated edges in the topology follows an arithmetic regression pattern. Note that the elimination of neighborhood relations is symmetrical, that is, if particle r is removed from the neighborhood of particle i, particle i is also removed from the neighborhood of particle r. The inertia weight is then updated, and finally, the velocity-update mechanism is applied in the same way as in FIPS.

A.4.2 Parameterization Effects

We studied the impact of using different schedules for the topology and inertia weight updates on the algorithm's performance. The remaining parameters were set as follows: the maximum velocity $V_{\max}$ is set to $\pm X_{\max}$ (the maximum of the search range), the linearly-decreasing inertia weight is varied from 0.9 to 0.4, and the sum of the acceleration coefficients, φ, is set to 4.0.

The experimental conditions described in Section A.2 are used. Three swarm sizes ($n = 20,\ 40,\ 60$), four schedules of the topology update (measured in iterations; $k = n,\ 2n,\ 3n,\ 4n$) and four schedules of the inertia weight (measured in function evaluations; $wt_{\max} = n^2,\ 2n^2,\ 3n^2,\ 4n^2$) were tried. Note that the values of k and $wt_{\max}$ are independent of each other.

As an illustrative example of the results, consider Fig. A.4. It shows the RLDs obtained by Frankenstein's PSO algorithm on Griewank's function. These distributions correspond, as before, to a solution quality 0.001% above the optimum value. Only the results obtained with 4 out of the 12 possible combinations of topology schedules and population sizes are shown.

A combination of a slow topology update schedule ($3n$ or $4n$) and a fast inertia weight schedule (n^2 or $2n^2$) promotes the stagnation of the algorithm. This can be explained if we recall that FIPS has a strong stagnation tendency when using a highly connected topology: A slow topology update schedule maintains a high topology connectivity for more iterations and a fast inertia weight schedule quickly reduces the exploration capabilities of the particle swarm. These two effects also increase the algorithm's stagnation tendency. To counteract a fast stagnation tendency, the two possibilities are to slow down the inertia weight schedule or to speed up the change of the topology.

Increasing the number of particles increases the amount of information available to the algorithm during the first iterations. The exploitation of this information depends on the topology update and inertia weight schedules. The configurations that appear to better exploit it are those in which these two schedules are slow.

To compare the configurations' relative performance across problems that have different scales, we look at the average (over the 8 benchmark problems of the experimental setup) of the standardized median solution quality (i.e., for each group, the mean is equal to zero and the standard deviation is equal to one) as a function of the topology update and the inertia weight schedules for different termination criteria. The results are shown in Fig. A.5. Since we work with minimization problems, a lower average standard solution quality means that the specific configuration found better solutions.

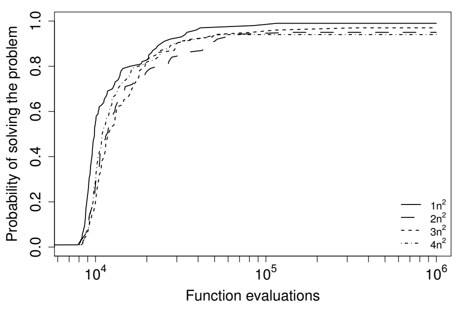
According to Fig. A.5, the algorithm needs more exploratory configurations (i.e., fast topology update schedules and slow inertia weight schedules) for long runs. For short runs, configurations with slow topology update schedules and fast inertia weight schedules yield the best results. For runs of 10^4 and 10^5 function evaluations, the best configurations are intermediate ones (i.e., fast or slow schedules for both the topology and inertia weight updates).

Algorithm 8 Frankenstein's particle swarm optimization algorithm

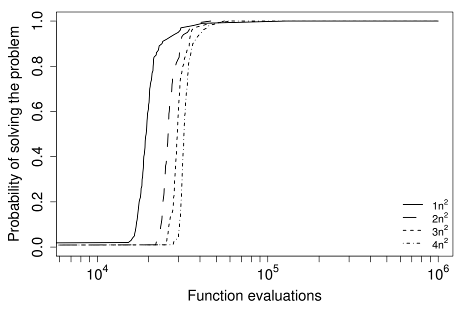
```
/* Initialization */
for $i = 1$ to $n$ do
  Create particle $i$ and add it to the set of particles $\mathcal{P}$
  Initialize its vectors $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$ to random values within the search range and maximum allowed velocities
  Set $\boldsymbol{pb}_i = \boldsymbol{x}_i$
  Set $\mathcal{N}_i = \mathcal{P}$
end for

/* Main Loop */
Set $t = 0$
Set $esteps = 0$
repeat
  /* Evaluation Loop */
  for $i = 1$ to $n$ do
    if $f(\boldsymbol{x}_i)$ is better than $f(\boldsymbol{pb}_i)$ then
      Set $\boldsymbol{pb}_i = \boldsymbol{x}_i$
    end if
  end for
  /* Topology Update */
  if $t > 0 \wedge t <= k \wedge t \bmod \lceil k/(n-3) \rceil = 0$ then
    /* $t > 0$ ensures that a fully connected topology is used first */
    /* $t <= k$ ensures that the topology update process is not called after iteration $k$ */
    /* $t \bmod \lceil k/(n-3) \rceil = 0$ ensures the correct scheduling of the topology update process */
    for $i = 1$ to $n - (2 + esteps)$ do
      /* $n - (2 + esteps)$ ensures the arithmetic regression pattern */
      if $|\mathcal{N}_i| > 2$ then
        /* $|\mathcal{N}_i| > 2$ ensures proper node selection */
        Select at random particle $r$ from $\mathcal{N}_i$ such that $r$ is not adjacent to $i$
        Eliminate particle $r$ from $\mathcal{N}_i$
        Eliminate particle $i$ from $\mathcal{N}_r$
      end if
    end for
    Set $esteps = esteps + 1$
  end if
  /* Inertia Weight Update */
  if $t \leq wt_{max}$ then
    Set $w(t) = \frac{wt_{max} - t}{wt_{max}}(w_{max} - w_{min}) + w_{min}$
  else
    Set $w(t) = w_{min}$
  end if
  /* Velocity and Position Update */
  for $i = 1$ to $n$ do
    Generate $\boldsymbol{U}_m \ \forall m \in \mathcal{N}_i$
    Set $\varphi_m = \varphi / |\mathcal{N}_i| \ \forall m \in \mathcal{N}_i$
    Set $\boldsymbol{v}_i^{t+1} = w^t \boldsymbol{v}_i^t + \sum_{m \in \mathcal{N}_i} \varphi_k \boldsymbol{U}_k (\boldsymbol{pb}_k^t - \boldsymbol{x}_i^t)$
    Set $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}$
  end for
  Set $t = t + 1$
  Set $\boldsymbol{sol} = \underset{i \in \mathcal{P}}{\operatorname{argmin}} f(\boldsymbol{pb}_i^t)$
until $f(\boldsymbol{sol})$ value is good enough or $t = t_{max}$
```

The more exploratory behavior that a large population provides needs to be counterbalanced by the chosen configuration. For example, at 10^3 function evaluations, the best configuration tends to have faster inertia weight schedules for larger swarms. With 20 particles, the best configuration is at point $(4, 3)$ while with 40 and 60 particles, the best configurations are at $(4, 2)$ and $(4, 1)$, respectively. These results are consistent with those of the experimental comparison.

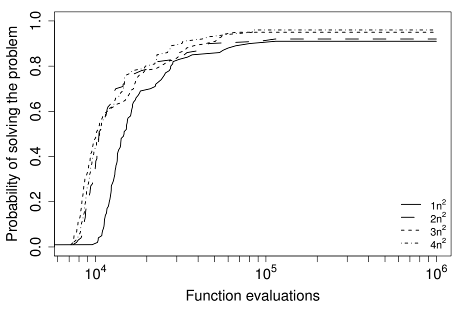
Like any other algorithm, Frankenstein's PSO has its own set of parameters that need to be set by the practitioner before trying to solve a problem. The final parameter settings will depend on the class of problems one is trying to solve and on the application scenario



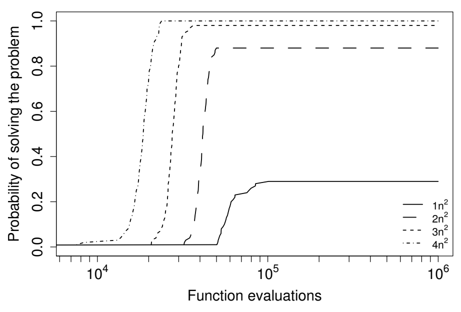
(a) 20 particles, $1 \times n$ iterations



(b) 60 particles, $1 \times n$ iterations

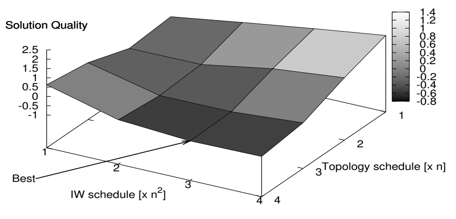


(c) 20 particles, $4 \times n$ iterations

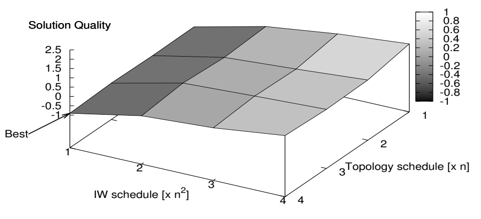


(d) 60 particles, $4 \times n$ iterations

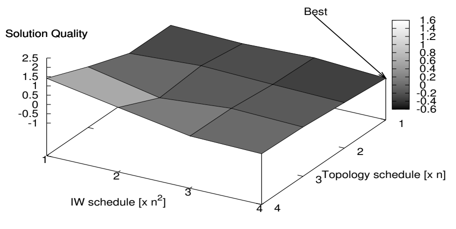
Figure A.4: RLDs obtained by Frankenstein's PSO algorithm on the Griewank function. The solution quality demanded is 0.001% above the global optimum. Each graph shows four RLDs that correspond to different inertia weight schedules.



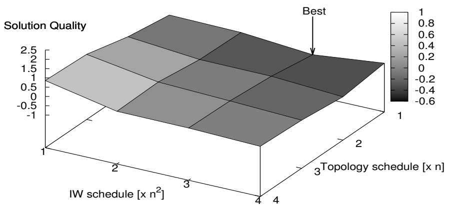
(a) 20 particles, 10^3 evaluations



(b) 60 particles, 10^3 evaluations



(c) 20 particles, 10^6 evaluations



(d) 60 particles, 10^6 evaluations

Figure A.5: Average standard solution quality as a function of the topology update and the inertia weight schedules for different termination criteria. In each case, the best configuration is pointed by an arrow.

requirements. Based on the results presented in this section we can derive the following guidelines for choosing the topology and the inertia weight schedules. If the number of function evaluations is restricted, a configuration with 20 particles, a slow topology change schedule ($\approx 4n$) and an intermediate inertia weight schedule ($\approx 3n^2$) would be the first one to try. If solution quality is the main concern, a configuration with 60 particles, a fast topology update schedule ($\approx n$), and a slow inertia weight ($\approx 4n^2$) should be preferred.

A.5 Performance Evaluation

The performance of Frankenstein's PSO is evaluated by comparing its best configurations with those of the PSO algorithms described in Section A.3. The best configurations of each variant were selected using the same ranking scheme as in Section A.3.2. The list of selected configurations is available at Montes de Oca et al. (2007).

Table A.6 shows the standardized median solution quality obtained by each configuration (identified only by the algorithm's name) for each termination criterion.

For runs of 10^3, 10^5 and 10^6 function evaluations, the best overall configuration is the one of Frankenstein's PSO. For runs of 10^4 function evaluations, the configuration of Frankenstein's PSO is ranked in the fourth place. However, with this same number of function evaluations, the configuration of Frankenstein's PSO is the best configuration in 6 of the 8 benchmark problems. The average rank of Frankenstein's PSO after 10^4 function evaluations can be explained with the results on Schwefel's function: FIPS (of which a component is used in Frankenstein's PSO) is the worst algorithm for this termination criterion (and also for the one of 10^3 function evaluations) on Schwefel's function.

The performance of Frankenstein's PSO suggests that indeed it is possible and profitable to integrate different existing algorithmic components into a single PSO variant. The results show that by composing existing algorithmic components, new high-performance variants can be built. At the same time, it is possible to gain insights into the effects of the interactions of different components on the algorithm's final performance. Of course, just as it is possible to take advantage of the strengths of different components, it is also possible that their weaknesses are passed on: the performance of Frankenstein's PSO on Schwefel's function is an example of this.

A.6 Conclusions and Future Work

Many PSO variants are proposed in the current literature. This is a consequence of the great attention that PSO has received since its introduction. However, it is also a sign of the lack of knowledge about which algorithmic components provide good performance on particular types of problems and under different operating conditions.

In an attempt to gain insight into the performance advantages that different algorithmic components provide, we compared what we consider to be some of the most influential or promising PSO variants. For practical reasons, many variants were left out of this study. Future studies should consider other variants as well as other components that are not necessarily present in existing PSO algorithms. In fact, some works are already exploring these issues (Mendes and Kennedy, 2007; Jordan et al., 2008; Yisu et al., 2008; Ramana Murthy et al., 2009; García-Villoria and Pastor, 2009). Recently, an alternative way of composing algorithmic components has been proposed by Montes de Oca et al. (2009b) and Engelbrecht (2010). The approach consists in shifting the integration of components from the particle level to the swarm level by creating heterogeneous swarms, that is, swarms composed of particles that move using different rules (i.e., algorithmic components). An avenue of research that seems promising is to experiment with random topologies that satisfy some constraints (e.g., a desired average connection degree). These works would help in improving our understanding of the interactions among PSO algorithmic components.

As it may have been expected, the results of our experimental comparison showed that no variant dominates all the others on all the problems of our benchmark suite over different run lengths. Nevertheless, we were able to identify general trends on the influence that various PSO algorithmic components and their parameters have on performance.

Based on these insights, we explored the possible advantages of combining algorithmic components that provided good performance into a single PSO variant by assembling a composite algorithm that we call *Frankenstein's PSO*. This new PSO algorithm is composed of

Table A.6: Best overall configurations of different PSO variants for different termination criteria. Each group is sorted by the average standard solution quality in ascending order, so the best overall configuration is listed first. The best values for each individual problem and stopping criterion are highlighted in boldface.

FES	Algorithm	Ackley	Griewank	Rastrigin	Salomon	Schwefel	Step	Rosenbrock	Sphere	Average
10^3	Frankenstein's PSO	**-2.024**	**-0.955**	-0.975	**-0.517**	1.378	**-1.315**	-0.302	**-1.108**	-0.727
	Increasing-IW	-0.013	-0.393	-0.950	-0.323	**-1.229**	-0.645	-0.367	-0.371	-0.536
	Decreasing-IW	-0.002	-0.386	**-1.067**	-0.316	-1.199	-0.359	-0.474	-0.425	-0.528
	FIPS	-0.765	-0.430	-0.080	-0.457	1.432	-0.932	0.206	-0.538	-0.195
	Constricted	0.476	-0.156	0.287	-0.276	-0.213	0.406	**-0.491**	-0.057	-0.003
	Stochastic-IW	0.656	0.124	0.652	-0.237	-0.046	0.693	-0.488	0.304	0.207
	AHPSO	0.476	-0.156	0.287	2.464	-0.213	0.406	**-0.491**	-0.057	0.340
	HPSOTVAC	1.198	2.353	1.847	-0.338	0.090	1.745	2.406	2.251	1.444
10^4	Increasing-IW	-0.129	-0.564	-0.593	-0.349	**-0.797**	-0.539	-0.348	-0.359	-0.460
	Constricted	-0.212	-0.616	-0.591	-0.373	-0.459	-0.539	-0.376	-0.359	-0.441
	Decreasing-IW	-0.065	-0.518	**-0.962**	-0.341	-0.754	-0.085	-0.370	-0.358	-0.431
	Frankenstein's PSO	**-1.061**	**-0.761**	0.056	**-0.386**	1.332	**-0.993**	**-0.414**	**-0.361**	-0.324
	Stochastic-IW	-0.131	0.443	-0.512	-0.361	-0.541	-0.085	-0.290	-0.359	-0.230
	FIPS	-1.056	-0.718	1.567	-0.378	1.760	-0.539	-0.364	**-0.361**	-0.011
	AHPSO	0.569	0.656	-0.512	2.474	-0.641	0.596	-0.312	-0.316	0.314
	HPSOTVAC	2.086	2.077	1.546	-0.287	0.101	2.185	2.473	2.475	1.582
10^5	Frankenstein's PSO	**-0.354**	**-0.883**	-1.192	**-0.359**	**-1.548**	-0.487	0.782	**-0.354**	-0.549
	Decreasing-IW	**-0.354**	0.631	-0.709	-0.355	-0.311	**-0.787**	-0.983	**-0.354**	-0.402
	Increasing-IW	**-0.354**	0.631	0.108	-0.355	-0.271	**-0.787**	-0.441	**-0.354**	-0.228
	Constricted	**-0.354**	**-0.883**	0.313	**-0.359**	0.729	-0.487	0.216	**-0.354**	-0.147
	Stochastic-IW	**-0.354**	0.631	1.130	**-0.359**	0.649	**-0.787**	-1.013	**-0.354**	-0.057
	FIPS	**-0.354**	**-0.883**	1.060	-0.355	1.372	0.712	1.008	**-0.354**	0.276
	AHPSO	**-0.354**	1.639	0.721	2.475	0.529	0.712	**-1.019**	**-0.354**	0.544
	HPSOTVAC	2.475	**-0.883**	**-1.431**	-0.334	-1.149	1.911	1.449	2.475	0.564
10^6	Frankenstein's PSO	**-0.354**	**-0.354**	-0.787	**-0.358**	-1.257	**-0.661**	-0.058	**-0.504**	-0.542
	Increasing-IW	**-0.354**	**-0.354**	0.002	-0.354	0.019	**-0.661**	0.039	**-0.504**	-0.271
	Decreasing-IW	**-0.354**	**-0.354**	0.472	-0.354	0.367	**-0.661**	**-0.778**	**-0.504**	-0.271
	FIPS	**-0.354**	**-0.354**	-0.546	-0.354	**-1.349**	0.661	0.685	**-0.504**	-0.264
	Stochastic-IW	**-0.354**	**-0.354**	0.415	**-0.358**	0.705	**-0.661**	-0.529	**-0.504**	-0.205
	Constricted	**-0.354**	**-0.354**	0.815	**-0.358**	1.072	**-0.661**	-0.717	**-0.504**	-0.132
	HPSOTVAC	2.475	**-0.354**	**-1.760**	-0.341	-0.705	0.661	2.129	2.184	0.536
	AHPSO	**-0.354**	2.475	1.388	2.475	1.149	1.984	-0.771	0.840	1.148

three main algorithmic components: (i) a time-varying population topology that decreases its connectivity as the optimization process evolves; (ii) a particles' velocity-update mechanism that exploits every stage of the topology change process, and (iii) a time-decreasing inertia weight that allows the user to tune the algorithm's exploration/exploitation capabilities. In many cases, Frankenstein's PSO is capable of performing better than the variants from which its components were taken.

As a methodological approach, in-depth experimental studies can help in identifying positive and negative (in terms of performance) interactions among algorithmic components and provide strong guidance for the informed design of new composite algorithms. Another selection of PSO variants would have probably ended up in a different Frankenstein's PSO algorithm. For this reason, further research is needed to understand which components are better suited for particular classes of problems and operating conditions and whether some components can be integrated into the same composite algorithm or not. Methods to quantify the contribution of each component on the composite algorithms' final performance are also needed to achieve this goal.

Bibliography

Acerbi, A., Marocco, D., and Nolfi, S. (2007). Social facilitation on the development of foraging behaviors in a population of autonomous robots. In NewEditor1 et al., editors, *LNCS 4648. Proceedings of the European Conference on Artificial Life (ECAL 2007)*, pages 625–634. Springer, Berlin, Germany.

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

Ampatzis, C., Tuci, E., Trianni, V., Christensen, A. L., and Dorigo, M. (2009). Evolving self-assembly in autonomous homogeneous robots: Experiments with two physical robots. *Artificial Life*, 15(4):465–484.

Anderson, C., Theraulaz, G., and Deneubourg, J.-L. (2002). Self-assemblages in insect societies. *Insectes Sociaux*, 49(2):99–110.

Annunziato, M. and Pierucci, P. (2003). The emergence of social learning in artificial societies. In Cagnoni, S. et al., editors, *LNCS 2611. Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003*, pages 467–478. Springer, Berlin, Germany.

Arabas, J., Michalewicz, Z., and Mulawka, J. J. (1994). GAVaPS – A genetic algorithm with varying population size. In *Proceedings of the IEEE Conference on Evolutionary Computation (CEC 1994)*, pages 73–78. IEEE Press, Piscataway, NJ.

Auger, A. and Hansen, N. (2005). A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1769–1776. IEEE Press, Piscataway, NJ.

Auger, A., Hansen, N., Zerpa, J. M. P., Ros, R., and Schoenauer, M. (2009). Experimental comparisons of derivative free optimization algorithms. In Vahrenhold, J., editor, *LNCS 5526. Proceedings of the Symposium on Experimental Algorithmics (SEA 2009)*, pages 3–15. Springer, Berlin, Germany.

Bäck, T., Eiben, A. E., and van der Vaart, N. A. L. (2000). An empirical study on GAs "without parameters". In *LNCS 1917. Parallel Problem Solving from Nature - PPSN VI, 6th International Conference*, pages 315–324. Springer, Berlin, Germany.

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T. et al., editors, *LNCS 4771. Proceedings of the International Workshop on Hybrid Metaheuristics (HM 2007)*, pages 108–122. Springer, Berlin, Germany.

Bandura, A. (1977). *Social Learning Theory.* Prentice-Hall, Englewood Cliffs, NJ.

Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation. The New Experimentalism.* Springer, Berlin, Germany.

Bayindir, L. and Şahin, E. (2007). A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering*, 15(2):115–147.

Beekman, M., Sword, G. A., and Simpson, S. J. (2008). Biological foundations of swarm intelligence. In Blum, C. and Merkle, D., editors, *Swarm Intelligence. Introduction and Applications*, pages 3–41. Springer, Berlin, Germany.

Beni, G. (2005). From swarm intelligence to swarm robotics. In Şahin, E. and Spears, W. M., editors, *LNCS 3342. Swarm Robotics: SAB 2004 International Workshop*, pages 1–9. Springer, Berlin, Germany.

Beni, G. and Wang, J. (1993). Swarm intelligence in cellular robotic systems. In Paolo, D. et al., editors, *Proceedings of the NATO Advanced Workshop on Robots and Biological Systems*, page 2. Springer, Berlin, Germany.

Bergman, A. and Feldman, M. W. (1995). On the evolution of learning: Representation of a stochastic environment. *Theoretical Population Biology*, 48(3):251–276.

Biederman, G. B., Davey, V. A., Suboski, M. D., Muir, D., Hall, D., and Fiorito, G. (1993). Social learning in invertebrates. *Science*, 259(5101):1627–1629.

Billard, A. and Dautenhahn, K. (1999). Experiments in learning by imitation – Grounding and use of communication in robotic agents. *Adaptive Behavior*, 7(3–4):415–438.

Birattari, M. (2009). *Tuning Metaheuristics: A machine learning perspective*. Springer, Berlin, Germany.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W. B. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 11–18. Morgan Kaufmann, San Francisco, CA.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-Race and iterated F-Race: An overview. In Bartz-Beielstein, T. et al., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, Berlin, Germany.

Blackwell, T. and Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472.

Blum, C. (2004). *Theoretical and Practical Aspects of Ant Colony Optimization*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

Blum, C. (2005). Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591.

Bonabeau, E. (1998). Social insect colonies as complex adaptive systems. *Ecosystems*, 1(5):437–443.

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

Bonabeau, E., Dorigo, M., and Theraulaz, G. (2000). Inspiration for optimization from social insect behaviour. *Nature*, 406(6791):39–42.

Bonabeau, E., Theraulaz, G., Deneubourg, J.-L., Aron, S., and Camazine, S. (1997). Self-organization in social insects. *Trends in Ecology & Evolution*, 12(5):188–193.

Bonabeau, E., Theraulaz, G., Deneubourg, J.-L., Franks, N. R., Rafelsberger, O., Joly, J.-L., and Blanco, S. (1998). A model for the emergence of pillars, walls and royal chambers in termite nests. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 353(1375):1561–1576.

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 *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, pages 4187–4193. IEEE Press, Piscataway, NJ.

Bonnie, K. E. and de Waal, F. B. M. (2007). Copying without rewards: socially influenced foraging decisions among brown capuchin monkeys. *Animal Kingdom*, 10(3):283–292.

Boyd, R. and Richerson, P. J. (1985). *Culture and the Evolutionary Process.* University of Chicago Press, Chicago, IL.

Brambilla, M., Pinciroli, C., Birattari, M., and Dorigo, M. (2009). A reliable distributed algorithm for group size estimation with minimal communication requirements. In Prassel, E. et al., editors, *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009)*, pages 1–6. IEEE Press, Piscataway, NJ. Proceedings on CD-ROM, paper ID 137.

Bratton, D. and Kennedy, J. (2007). Defining a standard for particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2007)*, pages 120 – 127. IEEE Press, Piscataway, NJ.

Breazeal, C. and Scassellati, B. (2000). Infant-like social interactions between a robot and a human caregiver. *Adaptive Behavior*, 8(1):49–74.

Breazeal, C. and Scassellati, B. (2002). Robots that imitate humans. *Trends in Cognitive Sciences*, 6(11):481–487.

Brent, R. P. (1973). *Algorithms for Minimization Without Derivatives.* Prentice-Hall, Englewood Cliffs, NJ.

Brown, C. and Laland, K. N. (2003). Social learning in fishes: a review. *Fish and Fisheries*, 4(3):280–288.

Bruecker, S. A., Di Marzo Serugendo, G., Hales, D., and Zambonelli, F., editors (2006). *LNCS 3910. Engineering Self-Organising Systems. Third International Workshop, ESOA 2005*, Berlin, Germany. Springer.

Bruecker, S. A., Di Marzo Serugendo, G., Karageorgos, A., and Nagpal, R., editors (2005). *LNCS 3464. Engineering Self-Organising Systems. Methodologies and Applications, ESOA 2004*, Berlin, Germany. Springer.

Bruecker, S. A., Hassas, S., Jelasity, M., and Yamins, D., editors (2007). *LNCS 4335. Engineering Self-Organising Systems. 4th International Workshop, ESOA 2006*, Berlin, Germany. Springer.

Buhl, J., Sumpter, D. J. T., Couzin, I. D., Hale, J. J., Despland, E., Miller, E. R., and Simpson, S. J. (2006). From disorder to order in marching locusts. *Science*, 312(5778):1402–1406.

Bullnheimer, B., Hartl, R. F., and Strauss, C. (1999). A new rank-based version of the Ant System: A computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38.

Cakmak, M., DePalma, N., Arriaga, R. I., and Thomaz, A. L. (2010). Exploiting social partners in robot learning. *Autonomous Robots*, 29(3–4):309–329.

Caldwell, C. A. and Millen, A. E. (2009). Social learning mechanisms and cumulative cultural evolution: Is imitation necessary? *Psychological Science*, 20(12):1478–1483.

Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2001). *Self-Organization in Biological Systems.* Princeton University Press, Princeton, NJ.

Carere, C., Montanino, S., Moreschini, F., Zoratto, F., Chiarotti, F., Santucci, D., and Alleva, E. (2009). Aerial flocking patterns of wintering starlings, *Sturnus vulgaris*, under different predation risk. *Animal Behaviour*, 77(1):101–107.

Castellano, C., Fortunato, S., and Loreto, V. (2009). Statistical physics of social dynamics. *Reviews of Modern Physics*, 81(2):591–646.

Cavalli-Sforza, L. L. and Feldman, M. W. (1981). *Cultural Transmission and Evolution. A Quantitative Approach.* Princeton University Press, Princeton, NJ.

Cavalli-Sforza, L. L. and Feldman, M. W. (1983). Cultural versus genetic adaptation. *Proceedings of the National Academy of Sciences*, 80(16):4993–4996.

Chakrabarti, B., Chakraborti, A., and Chatterjee, A., editors (2006). *Econophysics and sociophysics: Trends and perspectives.* Wiley-VCH, Wienheim, Germany.

Chamley, C. P. (2004). *Rational Herds. Economic Models of Social Learning.* Cambridge University Press, New York.

Chatterjee, A., Chatterjee, R., Matsuno, F., and Endo, T. (2007). Neuro-fuzzy state modeling of flexible robotic arm employing dynamically varying cognitive and social component based PSO. *Measurement*, 40(6):628–643.

Chatterjee, A. and Siarry, P. (2006). Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Computers & Operations Research*, 33(3):859–871.

Chaturvedi, K. T., Pandit, M., and Srivastava, L. (2009). Particle swarm optimization with time varying acceleration coefficients for non-convex economic power dispatch. *International Journal of Electrical Power & Energy Systems*, 31(6):249–257.

Chen, D. and Zhao, C. (2009). Particle swarm optimization with adaptive population size and its application. *Applied Soft Computing*, 9(1):39–48.

Chen, J., Qin, Z., Liu, Y., and Lu, J. (2005). Particle swarm optimization with local search. In *Proceedings of the International Conference on Neural Networks and Brain (ICNN&B 2005)*, pages 481–484. IEEE Press, Piscataway, NJ.

Chen, M.-R., Li, X., Zhang, X., and Lu, Y.-Z. (2010). A novel particle swarm optimizer hybridized with extremal optimization. *Applied Soft Computing*, 10(2):367–373.

Chiarandini, M., Birattari, M., Socha, K., and Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432.

Christensen, A. L., O'Grady, R., and Dorigo, M. (2009). From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766.

Clerc, M. (2006). *Particle Swarm Optimization.* ISTE, London, UK.

Clerc, M. and Kennedy, J. (2002). The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73.

Coelho, A. L. V. and de Oliveira, D. G. (2008). Dynamically tuning the population size in particle swarm optimization. In *Proceedings of the ACM Symposium on Applied Computing (SAC'08)*, pages 1782–1787. ACM Press, New York.

Coelho, L. D. S. and Mariani, V. C. (2006). Particle swarm optimization with quasi-Newton local search for solving economic dispatch problem. In *Proceedings of the IEEE Congress on Systems, Man, and Cybernetics (SMC 2006)*, pages 3109–3113. IEEE Press, Piscataway, NJ.

Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences.* Lawrence Earlbaum Associates, Hillsdale, NJ.

Conn, A. R., Gould, N. I. M., and Toint, P. L. (2000). *Trust-Region Methods.* MPS-SIAM Series on Optimization. MPS-SIAM, Philadelphia, PA.

Conte, R., Gilbert, N., and Sichman, J. S. (1998). MAS and social simulation: A suitable commitment. In Sichman, J. S. et al., editors, *LNAI 1534. International Workshop on Multi-agent systems and agent-based simulation (MABS 1998)*, pages 1–9. Springer, Berlin, Germany.

Cordón, O., de Viana, I. F., and Herrera, F. (2002). Analysis of the best-worst Ant System and its variants on the TSP. *Mathware and Soft Computing*, 9(2–3):177–192.

Cordón, O., de Viana, I. F., Herrera, F., and Moreno, L. (2000). A new ACO model integrating evolutionary computation concepts: The best-worst Ant System. In Dorigo, M., Middendorf, M., and Stützle, T., editors, *Abstract proceedings of ANTS 2000 – From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms*, pages 22–29. IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Couzin, I. D. and Krause, J. (2003). Self-organization and collective behavior in vertebrates. *Advances in the Study of Behavior*, 32:1–75.

Couzin, I. D., Krause, J., Franks, N. R., and Levin, S. A. (2005). Effective leadership and decision-making in animal groups on the move. *Nature*, 433(7025):513–516.

Das, S., Koduru, P., Gui, M., Cochran, M., Wareing, A., Welch, S. M., and Babin, B. R. (2006). Adding local search to particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 428–433. IEEE Press, Piscataway, NJ.

Dautenhahn, K. (1995). Getting to know each other–artificial social intelligence for autonomous robots. *Robotics and Autonomous Systems*, 16(2–4):333–356.

Deloach, S. A., Wood, M. F., and Sparkman, C. H. (2001). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258.

Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J.-M. (1990a). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168.

Deneubourg, J.-L., Goss, S., Franks, N. R., Sendova-Franks, A., Detrain, C., and Chrétien, L. (1990b). The dynamics of collective sorting robot-like ants and ant-like robots. In *Proceedings of the first international conference on simulation of adaptive behavior. From animals to animats*, pages 356–363. MIT Press, Cambridge, MA.

Di Marzo Serugendo, G., Karageorgos, A., Rana, O. F., and Zambonelli, F., editors (2004). *LNCS 2977. Engineering Self-Organising Systems. Nature-Inspired Approaches to Software Engineering*, Berlin, Germany. Springer.

Divina, F. and Vogt, P. (2006). A hybrid model for learning word-meaning mappings. In Vogt, P. et al., editors, *LNCS 4211. Symbol Grounding and Beyond, Third International Workshop on the Emergence and Evolution of Linguistic Communication (EELC 2006)*, pages 1–15. Springer, Berlin, Germany.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms.* PhD thesis, Dipartimento di Elettronica, Politecnico de Milano, Italy. In Italian.

Dorigo, M. (2007). Ant colony optimization. *Scholarpedia*, 2(3):1461.

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

Dorigo, M. and Di Caro, G. (1999). The ant colony optimization meta-heuristic. In Corne, D. et al., editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK.

Dorigo, M., Di Caro, G., and Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1991a). The Ant System: An autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M., Maniezzo, V., and Colorni, A. (1991b). Positive feedback as a search strategy. Technical Report 91–016, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 26(1):29–41.

Dorigo, M., Montes de Oca, M. A., and Engelbrecht, A. P. (2008). Particle swarm optimization. *Scholarpedia*, 3(11):1486.

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

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

Dorigo, M., Tuci, E., Groß, R., Trianni, V., Labella, T. H., Nouyan, S., Ampatzis, C., Deneubourg, J.-L., Baldassarre, G., Nolfi, S., Mondada, F., Floreano, D., and Gambardella, L. M. (2004). The SWARM-BOTS project. In Şahin, E. and Spears, W. M., editors, *LNCS 3342. Swarm Robotics: SAB 2004 International Workshop*, pages 31–44. Springer, Berlin, Germany.

dos Santos Coelho, L. (2008). A quantum particle swarm optimizer with chaotic mutation operator. *Chaos, Solitons & Fractals*, 37(5):1409–1418.

Ducatelle, F., Di Caro, G., and Gambardella, L. M. (2010). Cooperative self-organization in a heterogeneous swarm robotic system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, pages 87–94. ACM Press, New York.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pages 39–43. IEEE Press, Piscataway, NJ.

Eberhart, R. and Shi, Y. (2001). Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2001)*, pages 94–100. IEEE Press, Piscataway, NJ.

Eiben, A. E., Marchiori, E., and Valkó, V. A. (2004). Evolutionary algorithms with on-the-fly population size adjustment. In *LNCS 3242. Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference*, pages 41–50. Springer, Berlin, Germany.

Eiben, A. E., Shut, M. C., and de Wilde, A. R. (2006). Is self-adaptation of selection pressure and population size possible? – A case study. In *LNCS 4193. Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, pages 900–909. Springer, Berlin, Germany.

Ellison, G. and Fudenberg, D. (1995). Word-of-mouth communication and social learning. *Quarterly Journal of Economics*, 110(1):93–125.

Engelbrecht, A. P. (2005). *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, Ltd., West Sussex, UK.

Engelbrecht, A. P. (2010). Heterogeneous particle swarm optimization. In Dorigo, M. et al., editors, *LNCS 6234. Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, pages 191–202. Springer, Berlin, Germany.

Eshelman, L. J. and Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In Whitley, D. L., editor, *Foundation of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, San Mateo, CA.

European Space Agency (2010). What is Galileo? URL: `http://www.esa.int/esaNA/GGGMX650NDC_galileo_0.html` Last accessed: April 2011.

Feldman, M. W. and Laland, K. N. (1996). Gene-culture coevolutionary theory. *Trends in Ecology & Evolution*, 11(11):453–457.

Fernandes, C. and Rosa, A. (2006). Self-regulated population size in evolutionary algorithms. In *LNCS 4193. Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, pages 920–929. Springer, Berlin, Germany.

Fernández Martínez, J. L. and García Gonzalo, E. (2009). The PSO family: deduction, stochastic analysis and comparison. *Swarm Intelligence*, 3(4):245–273.

Ferrante, E., Brambilla, M., Birattari, M., and Dorigo, M. (2010). Socially-mediated negotiation for obstacle avoidance in collective transport. In *Proceedings of the 10th International Symposium on Distributed Autonomous Robotic Systems (DARS 2010)*, page to appear. Springer, Berlin, Germany.

Finin, T., Fritzson, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the third international conference on information and knowledge management (CIKM 1994)*, pages 456–463. ACM Press, New York.

Flinn, M. V. (1997). Culture and the evolution of social learning. *Evolution and Human Behavior*, 18(1):23–67.

Franz, M. and Matthews, L. J. (2010). Social enhancement can create adaptive, arbitrary and maladaptive cultural traditions. *Proceedings of the Royal Society B: Biological Sciences*, 277(1698):3363–3372.

Fujisawa, R., Dobata, S., Kubota, D., Imamura, H., and Matsuno, F. (2008a). Dependency by concentration of pheromone trail for multiple robots. In *LNCS 5217. Proceedings of the Sixth International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS 2008)*, pages 283–290. Springer, Berlin, Germany.

Fujisawa, R., Imamura, H., Hashimoto, T., and Matsuno, F. (2008b). Communication using pheromone field for multiple robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, pages 1391–1396. IEEE Press, Piscataway, NJ.

Galam, S. (1986). Majority rule, hierarchical structures, and democratic totalitarianism: A statistical approach. *Journal of Mathematical Psychology*, 30(4):426–434.

Galef Jr., B. G. (1990). A historical perspective on recent studies of social learning about foods by norway rats. *Canadian Journal of Psychology*, 44(3):311–329.

Galef Jr., B. G. (1996). Social enhancement of food preferences in Norway rats: A brief review. In Heyes, C. M. and Galef Jr., B. G., editors, *Social learning and imitation: The roots of culture*, pages 49–64. Academic Press, San Diego, CA.

Galef Jr., B. G. (2009). Strategies for social learning: Testing predictions from formal theory. *Advances in the Study of Behavior*, 39:117–151.

Gambardella, L. M. and Dorigo, M. (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. In Prieditis, A. and Russell, S., editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 252–260. Morgan Kaufmann Publishers, Palo Alto, CA.

Gambardella, L. M. and Dorigo, M. (1996). Solving symmetric and asymmetric TSPs by ant colonies. In *ICEC'96 Proceedings of the IEEE Conference on Evolutionary Computation*, pages 622–627. IEEE Press, Piscataway, NJ.

García-Pardo, J. A., Soler, J., and Carrascosa, C. (2010). Social conformity and its convergence for reinforcement learning. In Dix, J. and Witteveen, C., editors, *LNAI 6251. Proceedings of the 8th German conference on Multiagent system technologies (MATES 2010)*, pages 150–161. Springer, Berlin, Germany.

García-Villoria, A. and Pastor, R. (2009). Introducing dynamic diversity into a discrete particle swarm optimization. *Computers & Operations Research*, 36(3):951–966.

Garnier, S., Gautrais, J., Asadpour, M., Jost, C., and Theraulaz, G. (2009). Self-organized aggregation triggers collective decision making in a group of cockroach-like robots. *Adaptive Behavior*, 17(2):109–133.

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

Garnier, S., Tache, F., Combe, M., Grimal, A., and Theraulaz, G. (2007b). Alice in pheromone land: An experimental setup for the study of ant-like robots. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2007)*, pages 37–44. IEEE Press, Piscataway, NJ.

Gershenson, C. (2007). *Design and Control of Self-organizing Systems.* PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium.

Ghosh, S., Das, S., Kundu, D., Suresh, K., and Abraham, A. (2011). Inter-particle communication and search-dynamics of *lbest* particle swarm optimizers: An analysis. *Information Sciences.* In press.

Gimmler, J., Stützle, T., and Exner, T. E. (2006). Hybrid particle swarm optimization: An examination of the influence of iterative improvement algorithms on performance. In Dorigo, M. et al., editors, *LNCS 4150. Ant Colony Optimization and Swarm Intelligence. 5th International Workshop, ANTS 2006*, pages 436–443. Springer, Berlin, Germany.

Gmytrasiewicz, P. J. and Durfee, E. H. (2000). Rational coordination in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 3(4):319–350.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.

Goodrich, M. A. and Schultz, A. C. (2007). Human-robot interaction: A survey. *Foundations and Trends in Human-Computer Interaction*, 1(3):203–275.

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

Granovetter, M. (1978). Threshold models of collective behavior. *The American Journal of Sociology*, 83(6):1420–1443.

Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chez *Bellicositermes natalensis* et *Cubitermes sp.* La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80.

Groß, R. and Dorigo, M. (2008). Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285–305.

Grushin, A. and Reggia, J. A. (2008). Automated design of distributed control rules for the self-assembly of prespecified artificial structures. *Robotics and Autonomous Systems*, 56(4):334–359.

Guérin, B. (1993). *Social facilitation.* European monographs in social psychology. Maison des Sciences de l'Homme and Cambridge University Press, Paris, France and Cambridge, UK.

Guntsch, M. and Middendorf, M. (2002). A population based approach for ACO. In Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., and Raidl, G. R., editors, *LNCS 2279. Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, pages 71–80. Springer, Berlin, Germany.

Gutiérrez, Á., 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.

Haken, H. (2008). Self-organization. *Scholarpedia*, 3(8):1401.

Hamman, H., Szymanski, M., and Worn, H. (2007). Orientation in a trail network by exploiting its geometry for swarm robotics. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2007)*, pages 310–315. IEEE Press, Piscataway, NJ.

Handl, J., Knowles, J., and Dorigo, M. (2005). Ant-based clustering and topographic mapping. *Artificial Life*, 12(1):35–61.

Handl, J. and Meyer, B. (2007). Ant-based and swarm-based clustering. *Swarm Intelligence*, 1(2):95–113.

Hansen, N. and Kern, S. (2004). Evaluating the CMA evolution strategy on multimodal test functions. In Yao, X. et al., editors, *LNCS 3242. Parallel Problem Solving from Nature - PPSN VIII*, pages 282–291. Springer, Berlin, Germany.

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18.

Hansen, N., Ostermeier, A., and Gawelczyk, A. (1995). On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 57–64. Morgan Kaufmann, San Francisco, CA.

Hao, L. and Hu, L. (2009). Hybrid particle swarm optimization for continuous problems. In *Proceedings of the ISECS International Colloquium on Computing, Communication, Control and Management. CCCM 2009*, pages 283–286. IEEE Press, Piscataway, NJ.

Harik, G. R. and Lobo, F. G. (1999). A parameter-less genetic algorithm. In Banzhaf, W. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, pages 258–265. Morgan Kaufmann, San Francisco, CA.

He, M., Jennings, N. R., and Leung, H.-F. (2003). On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):985–1003.

Helbing, D. (2010). *Quantitative Sociodynamics. Stochastic Methods and Models of Social Interaction Processes*. Springer, Berlin, Germany, second edition.

Helbing, D. and Vicsek, T. (1999). Optimal self-organization. *New Journal of Physics*, 1:13.1–13.17.

Heppner, F. and Grenander, U. (1990). A stochastic nonlinear model for coordinated bird flocks. In Krasner, S., editor, *The Ubiquity of Chaos.*, pages 233–238. AAAS Publications, Washington, DC.

Herianto, H. and Kurabayashi, D. (2009). Realization of an artificial pheromone system in random data carriers using RFID tags for autonomous navigation. In *IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 2288–2293. IEEE Press, Piscataway, NJ.

Herrera, F., Lozano, M., and Molina, D. (2010). Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. URL: `http://sci2s.ugr.es/eamhco/updated-functions1-19.pdf` Last accessed: July 2010.

Heyes, C. M. (1994). Social learning in animals: Categories and mechanisms. *Biological Reviews*, 69(2):207–231.

Heyes, C. M., Ray, E. D., Mitchell, C. J., and Nokes, T. (2000). Stimulus enhancement: Controls for social facilitation and local enhancement. *Learning and Motivation*, 31(2):83–98.

Holland, O. and Melhuish, C. (1999). Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202.

Holland, O., Melhuish, C., and Hoddell, S. E. J. (1999). Convoying: using chorusing for the formation of travelling groups of minimal agents. *Robotics and Autonomous Systems*, 28(2–3):207–216.

Hölldobler, B. and Wilson, E. (1990). *The Ants*. Harvard University Press, Cambridge, MA.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann, San Francisco, CA.

Höppner, F., Klawonn, F., Kruse, R., and Runkler, T. (1999). *Fuzzy Cluster Analysis. Methods for classification, data analysis and image recognition.* John Wiley & Sons, Ltd., West Sussex, UK.

Howard, A., Matarić, M. J., and Sukhatme, G. S. (2002). An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2):113–126.

Hsieh, S.-T., Sun, T.-Y., Liu, C.-C., and Tsai, S.-J. (2009). Efficient population utilization strategy for particle swarm optimizer. *IEEE Transactions on Systems, Man, and Cybernetics. Part B: Cybernetics*, 39(2):444–456.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Murphy, K. P. (2009). An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*, pages 271–278. ACM Press, New York.

IEEE Foundation for Intelligent Physical Agents (2011). Foundation for intelligent physical agents. URL: `http://www.fipa.org/` Last accessed: May 2011.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys*, 31(3):316–323.

Jang, M. and Cho, S. (2002). Observational learning algorithm for an ensemble of neural networks. *Pattern Analysis & Applications*, 5(2):154–167.

Janson, S. and Middendorf, M. (2003). A hierarchical particle swarm optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 770–776. IEEE Press, Piscataway, NJ.

Janson, S. and Middendorf, M. (2005). A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man and Cybernetics. Part B: Cybernetics*, 35(6):1272–1282.

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

Johnson, B. R. (2009). Pattern formation on the combs of honeybees: increasing fitness by coupling self-organization with templates. *Proceedings of the Royal Society B: Biological Sciences*, 276(1655):255–261.

Johnson, S. G. (2008). The NLopt nonlinear-optimization package. URL: `http://ab-initio.mit.edu/nlopt` Last Accessed: July 2010.

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, San Francisco, CA.

Jordan, J., Helwig, S., and Wanka, R. (2008). Social interaction in particle swarm optimization, the ranked FIPS and adaptive multi-swarms. In Keijzer, M. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 49–56. ACM Press, New York.

Kadirkamanathan, V., Selvarajah, K., and Fleming, P. J. (2006). Stability analysis of the particle dynamics in particle swarm optimizer. *IEEE Transactions on Evolutionary Computation*, 10(3):245–255.

Kaebling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4(1):237–285.

Kaipa, K. N., Bongard, J. C., and Meltzoff, A. N. (2010). Self discovery enables robot social cognition: Are you my teacher? *Neural Networks*, 23(8–9):1113–1124.

Karaboga, D. and Akay, B. (2009). A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, 31(1–4):61–85.

Karaboga, D. and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):459–471.

Kennedy, J. (1997). The particle swarm: Social adaptation of knowledge. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1997)*, pages 303–308. IEEE Press, Piscataway, NJ.

Kennedy, J. (1999). Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 1931–1938. IEEE Press, Piscataway, NJ.

Kennedy, J. (2003). Bare bones particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2003)*, pages 80–87. IEEE Press, Piscataway, NJ.

Kennedy, J. (2006). Swarm intelligence. In Zomaya, A. Y., editor, *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*, pages 187–219. Springer US, Secaucus, NJ.

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

Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Swarm Intelligence*. The Morgan Kaufmann Series in Evolutionary Computation. Morgan Kaufmann, San Francisco, CA.

Kennedy, J. and Mendes, R. (2002). Population structure and particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 1671–1676. IEEE Press, Piscataway, NJ.

Kennedy, J. and Mendes, R. (2006). Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications & Reviews*, 36(4):515–519.

Kernbach, S., Meister, E., Schlachter, F., Jebens, K., Szymanski, M., Liedke, J., Laneri, D., Winkler, L., Schmickl, T., Thenius, R., Corradi, P., and Ricotti, L. (2008). Symbiotic robot organisms: REPLICATOR and SYMBRION projects. In Madhavan, R. and Messina, E., editors, *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems (PerMIS 2008)*, pages 62–69. ACM Press, New York.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2009). SATenstein: Automatically building local search SAT solvers from components. In Boutilier, C. et al., editors, *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 517–524. AAAI Press, Menlo Park, CA.

Kopp, S. and Graeser, O. (2006). Imitation learning and response facilitation in embodied agents. In Gratch, J., Young, M., Aylett, R., Ballin, D., and Olivier, P., editors, *LNAI 4133. Proceedings of the 6th International Conference on Intelligent Virtual Agents (IVA 2006)*, pages 28–41. Springer, Berlin, Germany.

Krapivsky, P. L. and Redner, S. (2003). Dynamics of majority rule in two-state interacting spin systems. *Physical Review Letters*, 90(23):238701, 4 pages.

Kube, C. R. and Bonabeau, E. (2000). Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1–2):85–101.

Kuniyoshi, Y., Inaba, M., and Inoue, H. (1994). Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822.

Kuwana, Y., Shimoyama, I., and Miura, H. (1995). Steering control of a mobile robot using insect antennae. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 530–535 vol.2. IEEE Press, Piscataway, NJ.

Laland, K. N. (2004). Social learning strategies. *Learning & Behavior*, 32(1):4–14.

Laland, K. N. and Kendal, J. R. (2003). What the models say about animal social learning. In Fragaszy, D. M. and Perry, S., editors, *The Biology of Traditions. Models and Evidence*, pages 33–55. Cambridge University Press, Cambridge, UK.

Lambiotte, R., Saramäki, J., and Blondel, V. D. (2009). Dynamics of latent voters. *Physical Review E*, 79(4):046107, 6 pages.

Lanzarini, L., Leza, V., and De Giusti, A. (2008). Particle swarm optimization with variable population size. In Goebel, R. et al., editors, *LNAI 5097. Proceedings of the International Conference on Artificial Intelligence and Soft Computing. ICAISC 2008*, pages 438–449. Springer, Berlin, Germany.

Latané, B. (1981). The psychology of social impact. *American Psychologist*, 36(4):343–356.

Leong, W.-F. and Yen, G. G. (2008). PSO-based multiobjective optimization with dynamic population size and adaptive local archives. *IEEE Transactions on Systems, Man, and Cybernetics. Part B: Cybernetics*, 38(5):1270–1293.

Liang, J. J. and Suganthan, P. N. (2005). Dynamic multi-swarm particle swarm optimizer with local search. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 522–528. IEEE Press, Piscataway, NJ.

Liang, J. J., Suganthan, P. N., and Deb, K. (2005). Novel composition test functions for numerical global optimization. In *Proceedings of IEEE Swarm Intelligence Symposium (SIS 2005)*, pages 68–75. IEEE Press, Piscataway, NJ.

Liao, T., Montes de Oca, M. A., Aydın, D., Stützle, T., and Dorigo, M. (2011). An incremental ant colony algorithm with local search for continuous optimization. In Krasnogor, N. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*. ACM Press, New York. To appear. Preprint available at `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-005r002.pdf`.

Lobo, F. G. and Lima, C. F. (2007). *Parameter Setting in Evolutionary Algorithms*, volume 54/2007 of *Studies in Computational Intelligence*, chapter Adaptive Population Sizing Schemes in Genetic Algorithms, pages 185–204. Springer, Berlin, Germany.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles. URL: `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004r001.pdf`.

López-Ibáñez, M. and Stützle, T. (2010). Automatic configuration of multi-objective ACO algorithms. In Dorigo, M. et al., editors, *LNCS 6234. Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, LNCS, pages 96–107. Springer, Berlin, Germany.

Lozano, M. and Herrera, F. (2010a). Call for papers: Special issue of soft computing: A fusion of foundations, methodologies and applications on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. Last accessed: July 2010. `http://sci2s.ugr.es/eamhco/CFP.php`.

Lozano, M. and Herrera, F. (2010b). Complementary material: SOCO special issue on large scale continuous optimization problems. URL: `http://sci2s.ugr.es/EAMHCO/` Last accessed: March 2011.

Lozano, M., Molina, D., and Herrera, F. (2011). Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing*. Forthcoming. DOI:10.1007/s00500-010-0639-2.

Lumer, E. D. and Faieta, B. (1994). Diversity and adaptation in populations of clustering ants. In Cliff, D. et al., editors, *Proceedings of the third international conference on Simulation of adaptive behavior : From animals to animats 3 (SAB 1994)*, pages 501–508. MIT Press, Cambridge, MA.

Mamei, M. and Zambonelli, F. (2005). Physical deployment of digital pheromones through RFID technology. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 1353–1354. ACM Press, New York.

Maniezzo, V. (1998). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. Technical Report CSR 98-1, Scienze dell'Informazione, Universitá di Bologna.

Maniezzo, V. (1999). Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369.

Martens, D., Baesens, B., and Fawcett, T. (2011). Editorial survey: swarm intelligence for data mining. *Machine Learning*, 82(1):1–42.

Matarić, M. J. (1997). Learning social behavior. *Robotics and Autonomous Systems*, 20(2–4):191–204.

Mayet, R., Roberz, J., Schmickl, T., and Crailsheim, K. (2010). Antbots: A feasible visual emulation of pheromone trails for swarm robots. In Dorigo, M. et al., editors, *LNCS 6234. Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, pages 84–94. Springer, Berlin, Germany.

Melhuish, C. and Kubo, M. (2007). Collective energy distribution: Maintaining the energy balance in distributed autonomous robots using trophallaxis. In Alami, R., Chatila, R., and Asama, H., editors, *Distributed Autonomous Robotic Systems 6*, pages 275–284. Springer, Berlin, Germany.

Melhuish, C., Sendova-Franks, A. B., Scholes, S., Horsfield, I., and Welsby, F. (2006). Ant-inspired sorting by robots: the importance of initial clustering. *Journal of the Royal Society Interface*, 3(7):235–242.

Mellinger, D., Shomin, M., Michael, N., and Kumar, V. (2010). Cooperative grasping and transport using multiple quadrotors. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*.

Mendes, R. (2004). *Population Topologies and Their Influence in Particle Swarm Performance*. PhD thesis, Escola de Engenharia, Universidade do Minho, Portugal.

Mendes, R. and Kennedy, J. (2007). Stochastic barycenters and beta distribution for gaussian particle swarms. In *LNAI 4874. Proceedings of the Portuguese Conference on Artificial Intelligence (EPIA 2007)*, pages 259–270. Springer, Berlin, Germany.

Mendes, R., Kennedy, J., and Neves, J. (2004). The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210.

Mohais, A., Mendes, R., Ward, C., and Posthoff, C. (2005). Neighborhood re-structuring in particle swarm optimization. In Zhang, S. and Jarvis, R., editors, *LNCS 3809. Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, pages 776–785. Springer, Berlin, Germany.

Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I. W., Floreano, D., Deneubourg, J.-L., Nolfi, S., Gambardella, L. M., and Dorigo, M. (2004). Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17(2–3):193–221.

Montes de Oca, M. A. (2011). Incremental social learning in swarm intelligence systems: Supplementary information webpage. URL: `http://iridia.ulb.ac.be/supp/IridiaSupp2011-014/`.

Montes de Oca, M. A., Aydın, D., and Stützle, T. (2011a). An incremental particle swarm for large-scale optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. *Soft Computing*. Forthcoming. DOI: 10.1007/s00500-010-0649-0.

Montes de Oca, M. A., Ferrante, E., Mathews, N., Birattari, M., and Dorigo, M. (2009a). Optimal collective decision-making through social influence and different action execution times. In Curran, D. and O'Riordan, C., editors, *Proceedings of the Workshop on Organisation, Cooperation and Emergence in Social Learning Agents of the European Conference on Artificial Life (ECAL 2009)*. No formal proceedings published.

Montes de Oca, M. A., Ferrante, E., Mathews, N., Birattari, M., and Dorigo, M. (2010a). Opinion dynamics for decentralized decision-making in a robot swarm. In Dorigo, M. et al., editors, *LNCS 6234. Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, pages 251–262. Springer, Berlin, Germany.

Montes de Oca, M. A., Ferrante, E., Scheidler, A., Pinciroli, C., Birattari, M., and Dorigo, M. (2010b). Majority-rule opinion dynamics with differential latency: A mechanism for self-organized collective decision-making. Technical Report TR/IRIDIA/2010-023, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Montes de Oca, M. A., Peña, J., Stützle, T., Pinciroli, C., and Dorigo, M. (2009b). Heterogeneous particle swarm optimizers. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 698–705. IEEE Press, Piscataway, NJ.

Montes de Oca, M. A. and Stützle, T. (2008a). Convergence behavior of the fully informed particle swarm optimization algorithm. In Keijzer, M. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 71–78. ACM Press, New York.

Montes de Oca, M. A. and Stützle, T. (2008b). Towards incremental social learning in optimization and multiagent systems. In Rand, W. et al., editors, *Workshop on Evolutionary Computation and Multiagent Systems Simulation of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 1939–1944. ACM Press, New York.

Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2006a). A comparison of particle swarm optimization algorithms based on run-length distributions. In Dorigo, M. et al., editors, *LNCS 4150. Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2006)*, pages 1–12. Springer, Berlin, Germany.

Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2006b). On the performance analysis of particle swarm optimisers. *AISB Quarterly*, 124:6–7.

Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2007). Frankenstein's PSO: Complete data. Supplementary information page at `http://iridia.ulb.ac.be/supp/IridiaSupp2007-002/`.

Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2009c). Frankenstein's PSO: A composite particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 13(5):1120–1132.

Montes de Oca, M. A., Stützle, T., Birattari, M., and Dorigo, M. (2010c). Incremental social learning applied to a decentralized decision-making mechanism: Collective learning made faster. In Gupta, I., Hassas, S., and Rolia, J., editors, *Proceedings of the Fourth IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 243–252. IEEE Computer Society Press, Los Alamitos, CA.

Montes de Oca, M. A., Stützle, T., Van den Enden, K., and Dorigo, M. (2011b). Incremental social learning in particle swarms. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 41(2):368–384.

Montes de Oca, M. A., Van den Enden, K., and Stützle, T. (2008). Incremental particle swarm-guided local search for continuous optimization. In Blesa, M. J. et al., editors, *LNCS 5296. Proceedings of the International Workshop on Hybrid Metaheuristics (HM 2008)*, pages 72–86. Springer, Berlin, Germany.

More, J. and Wild, S. (2009). Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191.

Moussaid, M., Garnier, S., Theraulaz, G., and Helbing, D. (2009). Collective information processing and pattern formation in swarms, flocks, and crowds. *Topics in Cognitive Science*, 1(3):469–497.

Müller, C. L., Baumgartner, B., and Sbalzarini, I. F. (2009). Particle swarm CMA evolution strategy for the optimization of multi-funnel landscapes. In Haddow, P. et al., editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 2685–2692. IEEE Press, Piscataway, NJ.

Nagasawa, S., Kanzaki, R., and Shimoyama, I. (1999). Study of a small mobile robot that uses living insect antennae as pheromone sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1999)*, pages 555–560. IEEE Press, Piscataway, NJ.

Nakagawa, S. and Cuthill, I. C. (2007). Effect size, confidence interval and statistical significance: a practical guide for biologists. *Biological Reviews*, 82(4):591–605.

Nannen, V. and Eiben, A. E. (2007). Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 975–980. AAAI Press, Menlo Park, CA.

Nehaniv, C. L. and Dautenhahn, K., editors (2007). *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions.* Cambridge University Press, Cambridge, UK.

Niehaus, J. and Banzhaf, W. (2003). More on computational effort statistics for genetic programming. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E., editors, *LNCS 2610. Genetic Programming: 6th European Conference, EuroGP 2003*, pages 164–172. Springer, Berlin, Germany.

Noble, J. and Franks, D. W. (2003). Social learning in a multi-agent system. *Computers and Informatics*, 22(6):561–574.

Nouyan, S., Campo, A., and Dorigo, M. (2008). Path formation in a robot swarm: Self-organized strategies to find your way home. *Swarm Intelligence*, 2(1):1–23.

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.

Nowak, A., Szamrej, J., and Latané, B. (1990). From private attitude to public opinion: A dynamic theory of social impact. *Phsycological Review*, 97(3):362–376.

Nowak, M. A. (2006). Five rules for the evolution of cooperation. *Science*, 314(5805):1560–1563.

Nuttall, Z. (1930). Documentary evidence concerning wild maize in Mexico. A contribution towards the solution of the origin of cultivated maize. *Journal of Heredity*, 21(5):217–220.

O'Brien, P. D. and Nicol, R. C. (1998). FIPA–Towards a standard for software agents. *BT Technology Journal*, 16(3):51–59.

O'Grady, R., Christensen, A. L., Pinciroli, C., and Dorigo, M. (2010a). Robots autonomously self-assemble into dedicated morphologies to solve different tasks. In van der Hoek, W. et al., editors, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1517–1518. IFAAMAS, Richland, SC.

O'Grady, R., Groß, R., Christensen, A. L., and Dorigo, M. (2010b). Self-assembly strategies in a group of autonomous mobile robots. *Autonomous Robots*, 28(4):439–455.

Ozcan, E. and Mohan, C. K. (1999). Particle swarm optimization: Surfing the waves. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*. IEEE Press, Piscataway, NJ.

Parker, C. A. C. and Zhang, H. (2009). Cooperative decision-making in decentralized multiple-robot systems: The best-of-N problem. *IEEE/ASME Transactions on Mechatronics*, 14(2):240–251.

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.

Partridge, B. L. (1982). Structure and function of fish schools. *Scientific American*, 246(6):114–123.

Pasteels, J.-M., Deneubourg, J.-L., and Goss, S. (1987). Self-organization mechanisms in ant societies (I): Trail recruitment to newly discovered food sources. *Experientia Supplementum*, 54:155–175.

Payton, D., Daily, M., Estowski, R., Howard, M., and Lee, C. (2001). Pheromone robotics. *Autonomous Robots*, 11(3):319–324.

Petalas, Y. G., Parsopoulos, K. E., and Vrahatis, M. N. (2007). Memetic particle swarm optimization. *Annals of Operations Research*, 156(1):99–127.

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, Á., Gambardella, L.-M., and Dorigo, M. (2010). ARGoS: a pluggable, multi-physics engine simulator for heterogeneous swarm roboticsAn integrated, cooperative development framework for heterogeneous swarm robotics. Technical Report TR/IRIDIA/2010-026, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

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*, 20(2–3):211–230.

Poli, R. (2007). On the moments of the sampling distribution of particle swarm optimisers. In *Workshop on Particle Swarm Optimization: The Second Decade of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, pages 2907–2914. ACM Press, New York.

Poli, R. (2009). Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. *IEEE Transactions on Evolutionary Computation*, 13(4):712–721.

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. An overview. *Swarm Intelligence*, 1(1):33–57.

Portugali, J. (2000). *Self-organization and the city*. Springer, Berlin, Germany.

Powell, M. J. D. (2006). *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, chapter The NEWUOA software for unconstrained optimization, pages 255–297. Springer, Berlin, Germany.

Powell, M. J. D. (2009). The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report NA2009/06, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, New York, second edition.

Priesterjahn, S. and Eberling, M. (2008). Imitation learning in uncertain environments. In Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors, *LNCS 5199. Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN X)*, pages 950–960. Springer, Berlin, Germany.

Ramana Murthy, G., Senthil Arumugam, M., and Loo, C. K. (2009). Hybrid particle swarm optimization algorithm with fine tuning operators. *International Journal of Bio-Inspired Computation*, 1(1/2):14–31.

Ratnaweera, A., Halgamuge, S. K., and Watson, H. C. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255.

Reeves, W. T. (1983). Particle systems–a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108.

Rendell, L., Boyd, R., Cownden, D., Enquist, M., Eriksson, K., Feldman, M. W., Fogarty, L., Ghirlanda, S., Lillicrap, T., and Laland, K. N. (2010a). Why copy others? Insights from the social learning strategies tournament. *Science*, 328(5975):208 – 213.

Rendell, L., Fogarty, L., Hoppit, W. J. E., Morgan, T. J. H., Webster, M. M., and Laland, K. N. (2011). Cognitive culture: theoretical and empirical insights into social learning strategies. *Trends in Cognitive Sciences*, 15(2):68–76.

Rendell, L., Fogarty, L., and Laland, K. N. (2010b). Rogers' paradox recast and resolved: Population structure and the evolution of social learning strategies. *Evolution*, 64(2):534–548.

Rescorla, R. A. (1988). Behavioral studies of pavlovian conditioning. *Annual Review of Neuroscience*, 11(1):329–352.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *ACM Computer Graphics*, 21(4):25–34.

Roberts, J., Stirling, T., Zufferey, J., and Floreano, D. (2009). 2.5D infrared range and bearing system for collective robotics. In Hamel, W. R., editor, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3659–3664. IEEE Press, Piscataway, NJ.

Rogers, A. R. (1988). Does biology constrain culture? *American Anthropologist*, 90(4):819–831.

Russell, R. A. (1999). Ant trails – An example for robots to follow? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 1999)*, pages 2968–2703. IEEE Press, Piscataway, NJ.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In Şahin, E. and Spears, W. M., editors, *LNCS 3342. Swarm Robotics: SAB 2004 International Workshop*, pages 10–20. Springer, Berlin, Germany.

Saunders, J., Nehaniv, C. L., and Dautenhahn, K. (2006). Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction (HRI 2006)*, pages 118–125. ACM Press, New York.

Scheidler, A. (2011). Dynamics of majority rule with differential latencies. *Physical Review E*, 83(3):031116, 4 pages.

Schelling, T. (1978). *Micromotives and Macrobehavior.* Norton & Co., New York.

Schloler, H. and Ngo, T. D. (2008). Trophallaxis in robotic swarms – beyond energy autonomy. In *Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV 2008)*, pages 1526–1533. IEEE Press, Piscataway, NJ.

Schmickl, T. and Crailsheim, K. (2008). Trophallaxis within a robotic swarm: Bio-inspired communication among robots in a swarm. *Autonomous Robots*, 25(1–2):171–188.

Seeley, T. D. (2010). *Honeybee Democracy.* Princeton University Press, Princeton, NJ.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200.

Sheskin, D. J. (2000). *Handbook of parametric and nonparametric statistical procedures.* Chapman & Hall/CRC, Boca Raton, FL, second edition.

Shi, Y. and Eberhart, R. (1998a). A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73. IEEE Press, Piscataway, NJ.

Shi, Y. and Eberhart, R. (1998b). Parameter selection in particle swarm optimization. In *LNCS 1447. Evolutionary Programming VII: 7th International Conference, EP98*, pages 591–600. Springer, Berlin, Germany.

Shi, Y. and Eberhart, R. (1999). Empirical study of particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 1945–1950. IEEE Press, Piscataway, NJ.

Shi, Y. and Eberhart, R. (2001). Fuzzy adaptive particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2001)*, pages 101–106. IEEE Press, Piscataway, NJ.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1–2):231–252.

Smit, S. K. and Eiben, A. E. (2009). Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 399–406. IEEE Press, Piscataway, NJ.

Smit, S. K. and Eiben, A. E. (2010). Beating the 'world champion' evolutionary algorithm via REVAC tuning. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–8. IEEE Press, Piscataway, NJ.

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173.

Stewart, R. L. and Russell, R. A. (2006). A distributed feedback mechanism to regulate wall construction by a robotic swarm. *Adaptive Behavior*, 14(1):21–51.

Stirling, T., Wischmann, S., and Floreano, D. (2010). Energy-efficient indoor search by swarms of simulated flying robots without global information. *Swarm Intelligence*, 4(2):117–143.

Storn, R. M. and Price, K. V. (1997). Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

Stützle, T., Birattari, M., and Hoos, H. H., editors (2007). *Engineering Stochastic Local Search Algorithms. Designing Implementing and Analizing Effective Heuristics. International Workshop, SLS 2007.* LNCS 4638. Springer, Berlin, Germany.

Stützle, T., Birattari, M., and Hoos, H. H., editors (2009). *Engineering Stochastic Local Search Algorithms. Designing Implementing and Analizing Effective Heuristics. Second International Workshop, SLS 2009.* LNCS 5752. Springer, Berlin, Germany.

Stützle, T. and Hoos, H. H. (1996). Improving the Ant System: A detailed report on the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Technical Report AIDA–96–12, FG Intellektik, FB Informatik, TU Darmstadt, Germany.

Stützle, T. and Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and local search for the traveling salesman problem. In Bäck, T., Michalewicz, Z., and Yao, X., editors, *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 309–314. IEEE Press, Piscataway, NJ.

Stützle, T. and Hoos, H. H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914.

Suganthan, P. N. (1999). Particle swarm optimiser with neighbourhood operator. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999)*, pages 1958–1962. IEEE Press, Piscataway, NJ.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, Nanyang Technological University, Singapore and IIT Kanpur, India.

Sugawara, K., Kazama, T., and Watanabe, T. (2004). Foraging behavior of interacting robots with virtual pheromone. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, pages 3074–3079. IEEE Press, Piscataway, NJ.

Teodorović, D. (2009). Bee colony optimization (BCO). In Lim, C. P., Jain, L. C., and Dehuri, S., editors, *Innovations in Swarm Intelligence*, pages 39–60. Springer, Berlin, Germany.

Theraulaz, G. and Bonabeau, E. (1995). Coordination in distributed building. *Science*, 269(5224):686–688.

Theraulaz, G., Bonabeau, E., and Deneubourg, J.-L. (1998). Response threshold reinforcements and division of labour in insect societies. *Proceedings of the Royal Society B: Biological Sciences*, 265(1393):327–332.

Thomaz, A. L. (2006). *Socially Guided Machine Learning.* PhD thesis, Massachussetts Institute of Technology, Cambridge, MA.

Thomaz, A. L. and Cakmak, M. (2009). Learning about objects with human teachers. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction (HRI 2009)*, pages 15–22. ACM Press, New York.

Tomasello, M. (2004). Learning through others. *Daedalus*, 133(1):51–58.

Trelea, I. C. (2003). The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325.

Trianni, V. and Nolfi, S. (2009). Self-organising sync in a robotic swarm. A dynamical system view. *IEEE Transactions on Evolutionary Computation*, 13(4):722–741.

Tuci, E., Groß, R., Trianni, V., Mondada, F., Bonani, M., and Dorigo, M. (2006). Cooperation through self-assembly in multi-robot systems. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):115–150.

Turgut, A. E., Çelikkanat, H., Gökçe, F., and Şahin, E. (2008). Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2–4):97–120.

van den Bergh, F. and Engelbrecht, A. P. (2001). Effects of swarm size on cooperative particle swarm optimisers. In Spector, L. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 469–476. Morgan Kaufmann, San Francisco, CA.

van der Post, D. J. and Hogeweg, P. (2004). Learning what to eat: Studying inter-relations between learning, grouping, and environmental conditions in an artificial world. In Sloot, P. M. A., Chopard, B., and Hoekstra, A. G., editors, *LNCS 3305. Proceedings of the 6th International Conference on Cellular Automata for Research and Industry (ACRI 2004)*, pages 492–501. Springer, Berlin, Germany.

Vogt, P. (2006). Cumulative cultural evolution: Can we ever learn more? In Nolfi, S., Baldasarre, G., Calabretta, R., Hallam, J. C. T., Marocco, D., Meyer, J.-A., Miglino, O., and Parisi, D., editors, *LNCS 4095. Proceedings of the 9th International Conference on Simulation of Adaptive Behavior (SAB 2006)*, pages 738–749. Springer, Berlin, Germany.

Wakano, J. Y., Aoki, K., and Feldman, M. W. (2004). Evolution of social learning: a mathematical analysis. *Theoretical Population Biology*, 66(3):249–258.

Wang, J. and Wang, D. (2004). Experiments and analysis on inertia weight in particle swarm optimization. In *Proceedings of the International Conference on Service Systems and Service Management*, pages 655–659. International Academic Publishers/World Publishing Corporation, Beijing, China.

Webb, B. (2000). What does robotics offer animal behaviour? *Animal Behaviour*, 60(5):545–558.

Werfel, J. and Nagpal, R. (2008). Three-dimensional construction with mobile robots and modular blocks. *International Journal of Robotics Research*, 27(3–4):463–479.

Werger, B. and Matarić, M. (1996). Robotic "food" chains: Externalization of state and program for minimal-agent foraging. In *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats (SAB 1996)*, pages 625–634. MIT Press, Cambridge, MA.

Wessnitzer, J. and Melhuish, C. (2003). Collective decision-making and behaviour transitions in distributed ad hoc wireless networks of mobile robots: Target-hunting. In NewEditor1 et al., editors, *LNCS 2801. Proceedings of the European Conference on Artificial Life (ECAL 2003)*, pages 893–902. Springer, Berlin, Germany.

Wilson, M., Melhuish, C., Sendova-Franks, A. B., and Scholes, S. (2004). Algorithms for building annular structures with minimalist robots inspired by brood sorting in ant colonies. *Autonomous Robots*, 17(2–3):115–136.

Winfield, A. F. T. and Griffiths, F. (2010). Towards the emergence of artificial culture in collective robot systems. In Levi, P. and Kernbach, S., editors, *Symbiotic Multi-robot Organisms*, pages 425–433. Springer, Berlin, Germany.

Witkowski, M. (2007). Energy sharing for swarms modeled on the common vampire bat. *Adaptive Behavior*, 15(3):307–328.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems.* John Wiley & Sons, Ltd., West Sussex, UK, second edition.

Yisu, J., Knowles, J., Hongmei, L., Yizeng, L., and Kell, D. B. (2008). The landscape adaptive particle swarm optimizer. *Applied Soft Computing*, 8(1):295–304.

Yuan, Z., Montes de Oca, M. A., Stützle, T., and Birattari, M. (2010). Modern continuous optimization algorithms for tuning real and integer algorithm parameters. In Dorigo, M. et al., editors, *LNCS 6234. Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, pages 204–215. Springer, Berlin, Germany.

Zajonc, R. B. (1965). Social facilitation. *Science*, 149(3681):269–274.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370.

Zhabotinsky, A. B. (2007). Belousov-Zhabotinsky reaction. *Scholarpedia*, 2(9):1435.

Zheng, Y.-L., Ma, L.-H., Zhang, L.-Y., and Qian, J.-X. (2003a). Empirical study of particle swarm optimizer with an increasing inertia weight. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2003)*, pages 221–226. IEEE Press, Piscataway, NJ.

Zheng, Y.-L., Ma, L.-H., Zhang, L.-Y., and Qian, J.-X. (2003b). On the convergence analysis and parameter selection in particle swarm optimization. In *Proceedings of the 2003 IEEE International Conference on Machine Learning and Cybernetics*, pages 1802–1807. IEEE Press, Piscataway, NJ.