Stigmergy as an Approach to Metaheuristic Optimization

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia, December 2006

Superviser: *Assist. Prof. Dr. Bogdan Filipič*

Co-superviser: *Assist. Prof. Dr. Jurij Šilc*

Evaluation board:

Assoc. Prof. Dr. Marko Bohanec, Chairman, Jožef Stefan Institute, Ljubljana, Slovenia

Assoc. Prof. Dr. Marjan Mernik, Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia

Assoc. Prof. Dr. Thiemo Krink, Department of Computer Science, University of Aarhus, Denmark

Examination board:

Assoc. Prof. Dr. Marko Bohanec, Chairman, Jožef Stefan Institute, Ljubljana, Slovenia

Assoc. Prof. Dr. Marjan Mernik, Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia

Assoc. Prof. Dr. Thiemo Krink, Department of Computer Science, University of Aarhus, Denmark

Assist. Prof. Dr. Bogdan Filipič, Jožef Stefan Institute, Ljubljana, Slovenia

Assist. Prof. Dr. Jurij Šilc, Jožef Stefan Institute, Ljubljana, Slovenia

CIP – Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004.02(043.3)

KOROŠEC, Peter, 1977–
Stigmergy as an approach to metaheuristic optimization =
Stigmergija kot pristop k metahevristični optimizaciji :
doctoral dissertation / Peter Korošec. – Ljubljana : [P. Korošec], 2006

229802496

Peter Korošec

Stigmergy as an Approach to Metaheuristic Optimization

Stigmergija kot pristop k metahevristični optimizaciji

Superviser: Assist. Prof. Dr. Bogdan Filipič

Co-superviser: Assist. Prof. Dr. Jurij Šilc

Doctoral Dissertation

December 2006

Mednarodna podiplomska šola Jožefa Stefana
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Table of Contents

Abstract

Developing metaheuristics to solve optimization problems is a rapidly growing field of research. This is due to the importance of optimization problems in the scientific as well as the industrial world. The methods developed in this dissertation are based on stigmergy: a method of communication in emergent systems, where the individual parts of the system communicate with one another by modifying their local environment. Stigmergy is the basis for all ant-colony optimization methods.

The introduction describes the components and concepts of Ant-Colony Optimization, and shows how ant-colony optimization methods have been used to solve combinatorial optimization problems like ordering, assignment, subset and grouping. In contrast, there have only been few attempts to solve numerical optimization problems in this way.

Almost all algorithms for combinatorial optimization are based on a single ant colony, while here we show how multiple ant colonies can be used to solve a grouping problem called the mesh-partitioning problem.

There is no straightforward way to implement ant-colony optimization to solve a numerical optimization problem. The main part of this dissertation is the development of two algorithms for solving numerical optimization problems. With the first one, called the Multilevel Ant-Stigmergy Algorithm, the problem is put into a discrete form, which is also a typical strategy for other ant-colony optimization algorithms. However in the second algorithm, called the Differential Ant-Stigmergy Algorithm, this is no longer required, making the Differential Ant-Stigmergy Algorithm much more versatile.

Both algorithms are evaluated on benchmark functions and compared to several other metaheuristic algorithms. They were also evaluated and compared on real-world problems from the fields of electrical engineering and metallurgical production.

The dissertation ends with the conclusion that for most numerical optimization problems addressed the Differential Ant-Stigmergy Algorithm is more suitable than the Multilevel Ant-Stigmergy Algorithm, and some suggestions for future work.

1 Introduction

In the past 20 years, new kinds of heuristic methods for solving a very general class of computational problems have evolved. They combine subordinate, more problem specific procedures—usually heuristics themselves—in what we hope is an efficient way. These methods are nowadays commonly referred to as metaheuristics. The name metaheuristic, first introduced by Glover [38], combines the Greek word *meta* ($\mu\varepsilon\tau\alpha$), meaning "beyond", here in the sense of "higher level", and the verb *heuriskein* ($\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$), which means "to find". Some representatives of this class are Iterated Local Search (ILS), Tabu Search (TS) and nature-inspired algorithms like Evolutionary Algorithms (EAs), Simulated Annealing (SA), Particle Swarm Optimization (PSO), and Ant-Colony Optimization (ACO) [78]. There is no commonly accepted definition for the term metaheuristic. In the past few years some researchers have tried to propose a definition. Some of these definitions are quoted below.

"A metaheuristic is formally defined as an iterative generation process that guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space; learning strategies are used to structure the information in order to find efficiently near-optimal solutions." — Osman and Laporte [93]

"Metaheuristics are typically high-level strategies that guide an underlying, more problem-specific heuristic, to increase their performance. The main aim is to avoid the disadvantages of iterative improvement and, in particular, multiple descent, by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high-quality solutions are produced quickly. This bias can be of various forms and can be cast

as a descent bias (based on the objective function), a memory bias (based on previously made decisions) or an experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. However, the main difference when compared to a pure random search is that in metaheuristic algorithms the randomness is not used blindly, but in an intelligent, biased form." — Stützle [108]

"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions during each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method." — Voß et al. [124]

"A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework that can be applied to different optimization problems with relatively few modifications in order to adapt them to a specific problem." — Metaheuristics Network [89]

Blum and Roli [10] summarized the fundamental properties of metaheuristics as follows:

- Metaheuristics are strategies that "guide" the search process.
- Their goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Techniques that constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.

- Metaheuristics can make use of domain-specific knowledge in the form of heuristics that are controlled by the upper-level strategy.
- Today's more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

Metaheuristics are high-level strategies for exploring search spaces by using a variety of methods. However, it is very important that a dynamic balance is maintained between diversification and intensification. The term diversification generally refers to the exploration of the search space, whereas intensification refers to the exploitation of the accumulated search experience.

Metaheuristics are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic or when it is not practical to implement such a method. Most commonly used metaheuristics are targeted on combinatorial optimization problems, but of course they can cope with any problem that can be recast in that form, such as a numerical optimization.

1.1 Optimization

The term optimization refers to the study of problems in which one seeks to minimize or maximize a function. The problem can be represented as follows. Given a function $f : A \to \mathbb{R}$ from some set A to the real numbers, we search for an element $x_0 \in A$ such that $f(x_0) \leq f(x)$ for all $x \in A$ (in the case of minimization), or such that $f(x_0) \geq f(x)$ for all $x \in A$ (in the case of maximization).

In this way we have formulated an optimization problem. Numerous theoretical and practical problems can be represented using this general formulation.

The domain A is called the search space and its elements candidate solutions. Usually, A is specified by a set of constraints that needs to be satisfied. The elements of A that satisfy the constraints are called feasible solutions. The function f is called an objective function, or a cost function. A feasible solution that optimizes (minimizes or maximizes) the objective function is called an optimal solution.

When the the objective function of the problem is not convex, there may be many local minima and maxima.

Local minimum x^* is defined as a point for which there exists some $\varepsilon > 0$ so that for all x such that $\|\mathbf{x} - \mathbf{x}^*\| \leq \varepsilon$ the expression $f(\mathbf{x}^*) \leq f(\mathbf{x})$ holds. Informally, in some region around x^* all of the function values are greater than or equal to the value at that point. The local maximum is defined analogously.

Many optimization problems of theoretical as well as practical importance consist of a search for the "best" setting of variables to achieve some goals. They seem to divide naturally into two categories: those defined over discrete variables (combinatorial optimization), and those defined over real-valued variables (numerical optimization).

Combinatorial optimization [17] algorithms are used to solve *NP*-hard optimization problems by exploring the usually large solution space. Combinatorial optimization algorithms attack these problems by reducing the effective size of the search space and by exploring the search space efficiently.

An informal definition of combinatorial optimization would be the following. It consists of optimization problems where the set of feasible solutions is discrete or can be reduced to a discrete one, and have corresponding graph representation. The goal is to find the best possible solution.

Formally, a combinatorial optimization problem $C = (X, D, \Omega, f, S, extr)$ is defined with:

- a set of variables $X = \{x_1, \ldots, x_n\}$ also called a solution space,
- a set of discrete domains $D = \{D_1, \ldots, D_n\}$ of variables, where the variable x_i is taken from the discrete domain D_i, $i = 1, \ldots, n$,
- a finite set of constraints, Ω, defined over the variables X,
- an objective function $f : D_1 \times \cdots \times D_n \to \mathbb{R}$,
- a set of feasible solutions $S = \{\mathbf{s} = (x_1, \ldots, x_n) | x_i \in D_i \wedge x_i$ satisfies constraints, Ω, for $i = 1, \ldots, n\}$,
- the extreme $extr$, which is $\min$ or $\max$.

Usually, S is called a space of feasible solutions. Each element of a set S is a candidate for a final solution. To solve a combinatorial optimization problem, a solution $\mathbf{s}^* \in S$ has to be found such that an $extr$ value is returned by the objective function f. Therefore, $f(\mathbf{s}^*) \leq f(\mathbf{s})$ must be true for $\forall \mathbf{s} \in S$ in the case when we are looking

for a minimum, or $f(\mathbf{s}^*) \geq f(\mathbf{s})$ must be true for $\forall \mathbf{s} \in S$ in the case of a search for a maximum. Solution $\mathbf{s}^*$ is called a global optimum solution of the problem C. The set of global optimum solutions for a given problem is marked with S^*.

Numerical optimization, as described by Nocedal and Wright [92], is important in decision science and in the analysis of physical systems. An important step in optimization is the identification of an objective, i.e., a quantitative measure of the performance of the system. This objective can be any quantity or combination of quantities that can be represented by a single number. The objective depends on certain characteristics of the system called parameters, which are often restricted or constrained in some way. For this reason numerical optimization is usually referred to as multi-parameter optimization. The parameters can have either continuous or discrete values. Our goal is to find the values of the parameters that optimize the objective. Depending on the type of parameters, we distinguish between *discrete* numerical optimization [17] and *continuous* numerical optimization [92].

An informal definition of numerical optimization would be the following. The domain of numerical optimization are optimization problems where the set of feasible solutions is continuous, and the goal is to find the best possible solution.

Formally, a numerical optimization problem $N = (P, D, \Omega, f, S, extr)$ is defined with:

- a set of parameters $P = \{p_1, \ldots, p_n\}$, also called a solution space,
- a set of continuous domains $D = D_1, \ldots, D_n$ of parameters, where the parameter p_i is taken from the continuous domain D_i, $i = 1, \ldots, n$,
- a finite set of constraints, Ω, defined over the parameters P,
- an objective function $f : D_1 \times \cdots \times D_n \rightarrow \mathbb{R}$,
- a set of feasible solutions $S = \{\mathbf{s} = (p_1, \ldots, p_n) | p_i \in D_i \wedge p_i$ satisfies constraints, Ω, for $i = 1, \ldots, n\}$,
- the extreme $extr$, which is $\min$ or $\max$.

Usually, S is called a space of feasible solutions. Each element of a set S is a candidate for a final solution. To solve a numerical optimization problem, a solution $\mathbf{s}^* \in S$ has to be found such that an $extr$ value is returned by the objective function f. There-

fore, $f(\mathbf{s}^*) \leq f(\mathbf{s})$ must be true for $\forall \mathbf{s} \in S$ in the case when we are looking for a minimum, or $f(\mathbf{s}^*) \geq f(\mathbf{s})$ must be true $\forall \mathbf{s} \in S$ in the case of a search for a maximum. Solution $\mathbf{s}^*$ is called a global optimum solution of the problem N. The set of global optimum solutions for a given problem is marked with S^*.

In the following, without loss of generality, we will deal only with optimization that corresponds to the minimization problems.

1.2 Motivation

Since the 1980s a lot of nature-inspired algorithms that were originally developed to solve hard combinatorial optimization problems were subsequently successfully transferred to numerical optimization problems. In the 1990s a new nature-inspired metaheuristic called Ant-Colony Optimization was introduced by Marco Dorigo [22]. Since then it has proven to be a very powerful metaheuristic for solving combinatorial optimization problems; however, there were only a few attempts to make it suitable for numerical optimization problems. This motivated us to try to develop an Ant-Colony Optimization algorithm for numerical optimization problems.

In this dissertation we will show two approaches that are usually taken when one wants to use a primarily combinatorial optimization metaheuristic and transfer it to numerical optimization problems. In the first approach, one can see the adaptation of a problem formulation to the algorithm. Here, the parameters are put into discrete form, without the loss of accuracy in real-world problems, which enables the algorithm to solve the discrete numerical optimization problem easily. In the second approach, one can see the adaptation of the algorithm to the problem. Here, a novel ant-colony-based algorithm was constructed to solve continuous numerical optimization problems.

1.3 Overview

The dissertation is further organized as follows. In Chapter 2 the background of ant-colony metaheuristics is outlined. Chapter 3 presents a multiple ant-colonies approach applied to the mesh-partitioning problem. Chapter 4 presents a new multilevel ant-colony approach applied to discrete numerical optimization problems. Chapter 5

presents the newly developed differential ant-colony approach applied to continuous numerical optimization problems. Real-world problems, from electrical engineering and metallurgical production, experiments and the obtained results are presented in Chapter 6, where the applied methods from Chapters 4 and 5 are evaluated in terms of their performance. The dissertation concludes with a summary of the findings of this study, the scientific contributions made by this dissertation, and some directions for future work.

2 Optimization with ant colonies

The complex social behaviors of ants have been much studied (e.g., Eugène N. Marais in 1937 [87]), and computer scientists are now finding that these behavior patterns can provide models for solving difficult optimization problems [27]. These behavior patterns can be found in stigmergy.

The term *stigmergy*, from the Greek words stigma ($\sigma\tau\iota\gamma\mu\alpha$), meaning "goad", and ergon ($\varepsilon\rho\gamma\sigma\nu$), which means "to work", was originally defined by the French entomologist Pierre-Paul Grassé (1895–1985) as a result of his observations of a species of termite, genus *Bellicositermes* [42]. He defined it to mean *"stimulating product of labor"*. In this dissertation we will concentrate on an ant-based approach that uses ant trails—a case of stigmergy—as a means of communication between ants, which is the basic idea behind the newly created optimization algorithms.

Stigmergy is a method of communication in emergent systems where the individual parts of the system communicate with one another by modifying their local environment. Stigmergy was observed in studying natural systems. For example, ants communicate with one another by laying down pheromone[1]. This kind of interaction can be observed in many eusocial[2] beings.

However, stigmergy is not restricted to eusocial species. On the internet, there are many cases of stigmergy that arise from users interacting only by modifying local parts of their shared virtual environment. A newsgroup is just one example of this: one initial user leaves a seed of an idea or a question, which attracts other users, who then build

[1] The term *pheromone* was introduced by Karlson and Lüscher in 1959 [54], based on the Greek words pherein ($\phi\varepsilon\rho\varepsilon\iota\nu$), meaning "to transfer", and hormon ($o\rho\mu o\nu$), which means "arouse to activity".

[2] Eusocial insects are recognized by three main characteristics: (i) The mother, along with individuals that may or may not be directly related, conducts cooperative care of the young. (ii) A reproductive division of labor evolves from sterile castes, which often have certain propensities or characteristics associated with helping behavior. (iii) There is an overlapping of generations, which allows for the older generations of offspring to help related, younger generations.

upon and modify this initial idea or question, eventually constructing an elaborate structure of connected thoughts.

Stigmergy can be viewed as a clever strategy used by nature to get populations of beings to self-organize, tell each other where to find resources, create sophisticated messaging systems and build complex architectural structures.

The sections in this chapter mainly follow the works of Guntsch and Branke [43], Dorigo and Stüztle [27, 111], and Belal et al. [7].

2.1 Stigmergy in ant colonies

It is known that only 2% of all insect species are eusocial. But these 2% represent more than 50% of the total insect biomass [2]. Insect colonies are capable of solving a number of problems that none of the individual insects would be able to solve by themselves. Examples of this can be seen in finding short paths when foraging for food (see Figure 2.1), task allocation when assigning labor to workers, and clustering when orga-

Figure 2.1 Stigmergy is an organizing principle in emergent systems in which the individual parts of the system communicate with one another indirectly by modifying their local environment. Ant colonies are a classic example. The ants communicate indirectly. Information is exchanged through modifications of the environment (local gradients of pheromone).

nizing brood chambers. All of these tasks have counterparts in real-world optimization problems like routing, scheduling, and clustering.

Individuals of a colony usually communicate with each other in a more or less direct way. It all depends on the species in question. Some species, like bees, communicate through the use of a characteristic dance [122]. With this dance one bee "shows" other bees where the food source is located. This is a type of direct communication because it requires visual contact. There are also other forms of direct communication, like stimulation by physical contact or with the exchange of food or liquid.

On the other hand, in indirect communication between the individuals of a colony, one has to change the environment in a such way that it will influence the behavior of the individuals later appearing in the modified environment. An example of indirect communication can be seen at ants foraging for food, or at termites constructing a nest [85].

While an ant is foraging for food, it will mark its path by distributing an amount of pheromone on the trail it is taking. When another ant, who is also foraging for food, founds this trail, it will be encouraged, but not forced, to follow the trail. The higher is the amount of pheromone, the greater is the encouragement. As already mentioned, this communication principle is called stigmergy.

In many ant colonies stigmergy is the basis for organization. The ants in a colony are self-organized. The term self-organization is used here to describe the complex behavior that emerges from the interaction of comparatively simple agents (ants). Self-organization enables ants to solve the complex problems encountered on a daily basis. Self-organization is used as a basis for problem solving, and it is especially apparent in its distributed and robust form. Effectively, an ant colony can maintain meaningful behavior even if a large number of ants are incapable of contributing for some amount of time. This can be a really useful feature in some real-world problems where these kinds of "interruptions" are very common (e.g., the internet).

To better understand the mechanism behind the ability of an ant colony to converge to good solutions when looking for a short path from the nest to a food source, Deneubourg et al. [21] made the following experiment: between a nest of the Argentine ant *Linepithema humile* and a food source two paths of identical length were put.

After some time had passed, it was observed that the ants had converged to one of the paths. Almost none of the ants chosen the other path. To see whether this type of ant would converge to the shorter of two alternative paths, the experimental setup shown in Figure 2.2 was evaluated by Goss et al. [41].

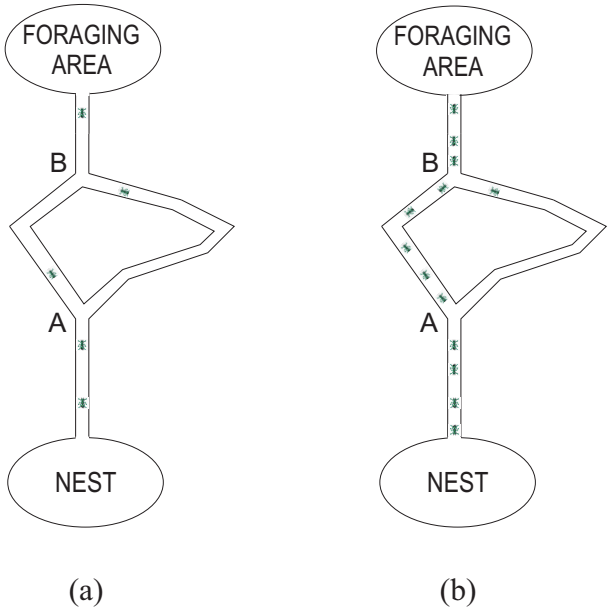

Figure 2.2 Bridge experiment: (a) at the start of the experiment, (b) after some time.

The *Linepithema humile* is practically blind and does not use any other sense to "visualize" its surroundings. So it cannot identify the shorter path using any of its senses. However, even without these abilities, a colony of ants is capable of finding the shorter path, as the experiment shows. In the beginning, all the ants are located at the nest. A number of ants start from the nest in search for food. While the ants walk, they lay pheromone on their paths. When they reach the first intersection at point A, they have no information about which way to go, due to the fact that no ant has walked this way before and left a pheromone trail for them to follow. Each ant will choose the left or right path with equal probability. As a consequence, about one half of the ants will take the shorter path and the other half the longer one. The ants that choose the shorter path will reach junction B first. Again, there is no information for the ants to use as orientation, so half of the ants will turn back toward the nest (longer branch), while the rest continue toward the food. When ants that chose the longer branch reach

junction B, the situation is a little different. Due to the fact that the first group of ants walked here, they have already laid down pheromone trails. Since the intensity of the pheromone trail heading back toward the nest is roughly twice as high as that of the trail heading to the food, the majority of ants will turn back toward the nest. They arrive there at approximately the same time as the ants that took the longer branch back. Now, if we look at the number of ants that walked on the shorter path compared to those on the longer one, we notice that the numbers differ. Since more ants have walked on the shorter path in comparison to the longer one, the amount of pheromone deposited on the shorter path is greater than on the longer path. When the next ants come to intersection A, they will be more inclined to choose the shorter path due to the larger amount of pheromone deposited on it. This is the first indication of optimization in progress.

The ants that reached the food, pick it up and carry it back to the nest. When arriving at junction B, the ants will prefer the shorter path with the same argument as used before for new ants that reached intersection A from the nest. Since the amount of pheromone at intersection A on the path back to the nest is (roughly) equal to the sum of the pheromone amounts on the two paths leading away from the nest, the path to the nest is most likely to be chosen by the returning ants. Because ants continually distribute pheromone as they walk, the shorter path is continually reinforced, until the amount of pheromone placed on it in relation to the alternative route is so high that practically all the ants use the shorter path, i.e., the system converges to the shortest path through self-reinforcement.

In the mean time the pheromone deposited by the ants to mark their trails slowly evaporates. Due to the evaporation of pheromone, a path that has not been chosen for some time or is chosen less frequently, will contain almost no traces of pheromone after a sufficient amount of time. This only further increases the likelihood that the ants will take the path identified by the continually updated pheromone.

Another interesting experiment was done to find out what happens when the ant colony is offered a new shorter connection between the nest and the food, after already walking on the longer one for some time. This situation was studied in an additional experiment where only the longer path was initially offered to the colony and

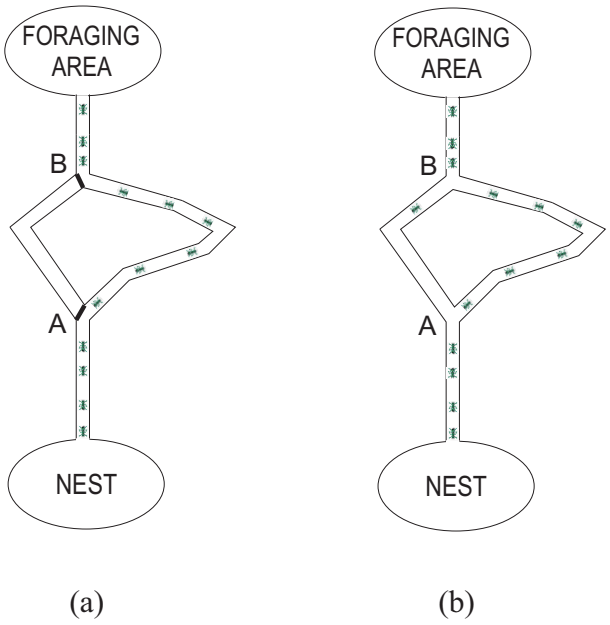

Figure 2.3 Broken-bridge experiment: (a) only longer path allowed, (b) after shorter path was added.

after 30 minutes the shorter path was added (see Figure 2.3). After that the ants chose the shorter path only sporadically and the colony was "trapped" on using the longer path. The reason for this is in the high pheromone concentration, and an almost exclusive continued reinforcement of the longer path. Pheromone evaporation, which usually favors the exploration of new paths, was in this case too slow: the lifetime of the pheromone is comparable to the duration of a trial (Goss et al., 1989). In short, this means that the pheromone evaporates too slowly to allow the ant colony to move from the path that was found to be "optimal" (high pheromone concentration) previously, to the newly created shorter path.

In the real world, ants wander around randomly and search for food. After finding food they return back to their nest. When returning, ants lay down pheromone trails. If other ants find such a pheromone trail, they are likely to follow it rather than to keep wandering around. If they find food, they reinforce the trail with pheromone even more.

Over time, the pheromone trail evaporates, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more pheromone evaporates. A short path, by comparison, gets marched over faster, and

thus the pheromone density remains high, as it is laid on the path faster than it can evaporate.

Thus, when an ant finds a good (in other words, short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leaves all the ants following a single path. Of course, as noted in the additional experiment, there should not be one already "established" path to the food source. The idea of the ant-colony algorithm is to mimic this behavior with "artificial ants" walking around the graph (their environment) representing the problem to be solved.

Stigmergy provides a new paradigm for developing decentralized, complex applications such as autonomous and collective robotics [52], communication in computer networks [101], multi-agent systems [47], optimization algorithms [27], etc.

2.2 Optimization with ants

An ant colony is an example of a distributed system that, in spite of the simplicity of its individuals, has a highly structured social organization. Because of this organization, an ant colony is capable of accomplishing complex natural tasks that far exceed the individual capacity of a single ant. It is not surprising, therefore, that computer scientists have taken inspiration from ants and their behavior in order to design algorithms for solving computationally demanding problems.

The foraging behavior of real ants, as described in the previous section, allows the ants to find the shortest paths between food sources and their nest [21]. This behavior was the inspiration for a new metaheuristic approach to solving hard optimization problems proposed by Dorigo et al. [26] and called the Ant-Colony Optimization (ACO).

ACO algorithms (see Algorithm 2.1) are based on a parameterized probabilistic model (pheromone model) that is used to model the chemical pheromone trails. Artificial ants incrementally construct solutions by adding solution components to a partial solution under consideration. In order to do this, artificial ants perform randomized walks on a completely connected graph $\mathcal{G}(C, L)$, called a construction graph, whose vertices are the solution components C, and the set L composed of the connections. When a constrained combinatorial optimization problem is considered, the problem

constraints, Ω, are built into the ants' constructive procedure in such a way that in every step of the construction process only feasible solution components can be added to the current partial solution.

Algorithm 2.1 Ant-Colony Optimization

```
1: while termination condition not satisfied do
2:    ScheduleActivities
3:        AntsActivity
4:        PheromoneEvaporation
5:        DaemonActions
6:    end ScheduleActivities
7: end while
```

The Algorithm 2.1 consists of three major activities:

AntActivity: In the construction phase an ant incrementally builds a solution by adding solution components to the partial solution constructed so far. The probabilistic choice of the next solution component to be added is done by means of transition probabilities. More specifically, ant n in step t moves from vertex $i \in C$ to vertex $j \in C$ with a probability given by:

$$\text{prob}_{ij,k}(t) = \begin{cases} \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}}{\sum_{l \in N_{i,k}} \tau_{il}^{\alpha}(t)\eta_{il}^{\beta}} & j \in N_{i,k} \\ 0 & j \notin N_{i,k} \end{cases}, \tag{2.1}$$

where η_{ij} is a priori available heuristic information, α and β are two parameters that determine the relative influence of the pheromone trail τ_{ij} and heuristic information, respectively, and $N_{i,n}$ is the feasible neighborhood of vertex i. If $\alpha = 0$, then only heuristic information is considered. Similarly, if $\beta = 0$, then only pheromone information is at work. Once an ant builds a solution, or while a solution is being built, the pheromone is being deposited (in vertices or on connections) according to the evaluation of a (partial) solution. This pheromone information will direct the search of the ants in the following iterations. The solution construction ends when an ant comes to the ending vertex (where the food is located).

PheromoneEvaporation: Pheromone-trail evaporation is a procedure that simulates the reduction of pheromone intensity. It is needed in order to avoid a too quick convergence of the algorithm to a sub-optimal solution. More specifically, pheromone evaporation is given by:

$$\tau_{ij}^{\text{NEW}} = (1 - \rho)\tau_{ij}^{\text{OLD}}, \tag{2.2}$$

where $\rho \in (0, 1]$ is an evaporation factor.

DaemonActions: Daemon actions can be used to implement centralized actions that cannot be performed by single ants. Examples are the use of a local search procedure applied to the solutions built by the ants, or the collection of global information that can be used to decide whether or not it is useful to deposit additional pheromone to bias the search process from a non-local perspective.

As we can see from the pseudo code in Algorithm 2.1, the **ScheduleActivities** construct does not specify how the three activities should be scheduled or synchronized, which also applies to the ants itself. This means it is up to the programmer to specify how these procedures will interact (in parallel or independently). This kind of approach leads to a lot of different organizing and design principles in ant colony optimization.

2.3 Organizing principles

A study of the swarm intelligence approach [7] revealed a useful set of organizing principles that can guide the design of efficient distributed applications for different kinds of problems. ACO inherits the following notable features:

Autonomy: The system does not require outside management or maintenance. Ants are autonomous, controlling their own behavior in a self-organized way.

Adaptability: Interactions between ants can arise through indirect communication via the local environment; two individuals interact indirectly when one of them modifies the environment and the other responds to the new environment at a later time. In this way ants are able to dynamically detect changes in the environment

and adopt to them. One could say that ant systems exhibit auto-configuration capabilities.

Scalability: ACO can be implemented in the form of groups consisting of a few or up to thousands of ants with the same control architecture.

Flexibility: No single ant in the colony is essential. Consequently, any ant can be dynamically added, removed, or replaced.

Robustness: ACO is a good example of a highly distributed architecture. This greatly enhances robustness; there is no need for central coordination, which means that despite the failure of one component the system as a whole will still function properly. If we combine scalability and flexibility, the ant system provides us with redundancy. This feature is essential for robustness.

Massive parallelism: The ant system is massively parallel and its functioning is truly distributed. Every ant inside its own group performs the same task. Basically, if we imagine each ant as a processor, the ACO architecture can be seen as a single-instruction-stream multiple-data-stream computer architecture [33].

Self-organization: Ant systems exhibit self-organization capabilities. The observed intelligence cannot be seen at a single ant itself, but emerges when the entire colony is at work.

Cost effectiveness: The ant-type system consists of a finite number of homogeneous ants, each of which has fairly limited capabilities on its own. Also, each ant has the same capabilities and control algorithm. It is clear that the autonomy and the highly distributed control afforded by the colony model greatly simplify the task of designing the implementation of parallel algorithms and hardware.

2.4 Design principles

Despite being a rather recent metaheuristic, ACO algorithms have already been applied to a large number of different combinatorial optimization problems. All ACO algorithms follow the basic layout described in Section 2.2. However, there are many factors that influence how these basic components are being implemented [111]. This is discussed in the following.

2.4.1 Definition of pheromone trails

A very important choice when applying ACO is the definition of the intended meaning of the pheromone trails. There are two major interpretations of a pheromone trail. First, the so-called standard interpretation refers to the desirability of visiting vertex j directly after vertex i. So, in this case it provides some information about the desirability of the relative positioning of the vertices i and j. Second, the pheromone trail can be interpreted as the desirability of visiting vertex i as the j-th vertex in the ants walk through the search graph. This is the so-called absolute positioning desirability.

The perfect example of the use of relative desirability is the well-known Traveling Salesman Problem (TSP), while absolute positioning is commonly used in Scheduling Problems (SPs). The main reason for the successful implementation of these different approaches into TSP and SP is the different role of the permutations in these two problems. When the permutations are cyclic, like in the TSP, where the relative and not the absolute order of the solution components is important, a relative-position-based pheromone trail is the appropriate choice. On the other hand, in most SPs two different permutations could represent two completely different solutions with very different costs. Therefore, the absolute-position-based pheromone trails are a better choice.

As we can see, the definition of the pheromone trail can be a very important decision when designing the algorithm for a particular problem. In the case of a bad choice in this step of the design the result will most probably be a poor performance.

2.4.2 Balancing exploration and exploitation

For a metaheuristic algorithm it is very important to have an appropriate balance between the exploitation of the experience gathered so far and the exploration of the search space. ACO offers several ways to achieve such a balance. This is usually done by pheromone-trails management. While ants deposit pheromone, they induce a probability distribution over the search space, which helps them to determine which part of the search space is being thoroughly searched. So, according to the pheromone distribution, ants can wander the search space almost randomly (uniform distribution), or on the other hand, they can concentrate only on a very small part (as small as only one

solution) of the search space (a type of degenerate distribution). If only one solution is possible the ants reach a so-called stagnation.

The ants' search experience is usually exploited by making the pheromone update a function of the solution quality achieved by each ant. Nevertheless, this is usually not enough to obtain good performance. One possible "upgrade" that is often used to improve an algorithm's performance is an elitist strategy. Here, the best solution found so far, in the course of the search process, contributes strongly to the procedure of trail updating.

Exploration of the search space is primarily achieved by the ants' randomized solution construction. In the case when heuristic information is not used, the exploration of the search space is higher in the initial iterations of the algorithm, and it decreases as the computation continues. Of course, one has to be very careful to ensure that the algorithm does not focus too strongly on what appear to be good regions in the search space. This can lead the algorithm to too early stagnation and, as a result, to local minima.

There are several ways to avoid stagnation occurring too early. Common to all approaches is maintaining a reasonable level of exploration through the whole search process. Some possible approaches are: use of a local pheromone update rule during the solution construction, use of an explicit lower limit on pheromone amount, use of re-initialization of the pheromone trails, and proper setting of the α and β parameters, which determine the relative influence of the pheromone trail and the heuristic information, and consequently balances the exploration and exploitation.

2.4.3 ACO and local search

It was shown that in many NP-hard combinatorial optimization problems, the ACO algorithm performed better when coupled with a local search algorithm. ACO is known as a coarse-grained search algorithm, while on the other hand, local search is a fine-grained algorithm. By coarse grained we mean that the search space is sampled at a lower resolution. In this way ACO is capable of finding good search regions quickly, but it has a hard time finding local optima. So, one can say that these two approaches are complementary. Local search can be treated as a type of daemon action. Usually,

local search algorithms improve the solutions generated by ants and the improved solutions are then used in the pheromone update. So when we "merge" these two approaches into one we have ants that probabilistically combine solution components that are part of the best locally optimal solutions found so far, and generate new, promising initial solutions for the local search. Even though it was shown that the use of local search contributes greatly to better performance in many ACO applications, this does not mean that ACO algorithms cannot successfully solve hard optimization problems on their own.

2.4.4 The importance of heuristic information

Problem-specific knowledge can be exploited with the use of heuristic information to direct the ants' probabilistic solution construction. Knowledge can be available a priori or at run-time. In static problems, where heuristic information is usually available a priori, the information is computed during the initialization and remains the same throughout the whole algorithm run. The main characteristics of static information are: it needs to be computed only once, and the computation is usually very trivial. However, in the case of dynamic problems, where the heuristic information is changing throughout the run, the heuristic information depends on the partial solution constructed so far and has to be computed for each step of the ant's walk. A dynamic calculation gives more accurate information about the heuristics, but at the expense of a higher computational cost. Another possibility for computing heuristic information is using lower bounds on the solution cost of the completion of the partial solution. With this method we can discard certain areas of the solution space because they lead to worse solutions than what is currently the best. A disadvantage of this approach is that the computation cost of finding lower bounds can be quite high. Especially because the lower bound has to be calculated for every step of each ant. Even though the use of heuristics is really important in ACO algorithms, this can be reduced with the use of local search. The reason is in a more direct use of the objective function to improve the solutions. Of course, this gives ACO algorithms more freedom when dealing with problems for which heuristic information is hard to a priori define or calculate.

2.4.5 The number of ants

A question often arises: why use a colony of ants instead of a single ant? The fact is, although a single ant is capable of generating a solution, the convergence and quality of solutions obtained with a colony is often much better. This is most obvious when ACO is used on geographically distributed problems, where the differential length effect exploited by ants in the solution can only arise in the presence of a colony of ants. For example, in routing problems ants solve many shortest-path problems in parallel, and this requires a colony of ants for each of these problems (finding a path between two vertices). But in combinatorial optimization problems the differential length effect is usually not exploited. This means that $m > 1$ ants building r solutions in l iterations is equivalent to one ant building r solutions in lm iterations. In this case the number of ants is not so important. But this holds only in theory. In practice there are a lot of ACO algorithms that use $m > 1$ to successfully solve hard combinatorial optimization problems. In general, the best value for m is different for every individual algorithm, and in most cases it has to be set experimentally. Fortunately, most ACO algorithms are quite robust in terms of the number of ants used.

2.4.6 Candidate lists

If an ACO algorithm is applied to a problem where ants have a large neighborhood to choose from, the solution construction is significantly slowed down and the probability of many ants visiting the same state is very small. This problem can be reduced by the use of a candidate list. A candidate list consists of a small set of promising neighbors of the current state. This promising neighbors are usually created considering a priori available information about the problem or some dynamically generated information. When it is applied the ACO can concentrate more on interesting parts of the search space. So far, the use of candidate lists or similar approaches in ACO algorithms is still rather unexplored. Inspiration from other techniques like TS [39] or Greedy Randomized Adaptive Search Procedure (GRASP) [30], where extensive use is made of candidate lists, could be useful for the development of effective candidate-list strategies for ACO.

2.5 ACO-based algorithms for combinatorial optimization

Here we present characteristic examples of a very large group of ACO-based algorithms that solve hard combinatorial optimization problems.

The Ant System (AS) [15, 22] was the first ant-based algorithm used to solve a hard combinatorial problem, i.e., the Traveling Salesman Problem (TSP). The main characteristics of this algorithm are positive feedback, distributed computation, and the use of a constructive greedy heuristic [26].

The Ant-Q (AQ) [24, 35] is a distributed approach to combinatorial optimization based on reinforcement learning. The AQ finds its basis in the AS and the Q-learning algorithm [127]. The fundamental difference between the AS and AQ is that in AQ only the ant that found the "best" path gets to deposit pheromone on its trail.

The Ant Colony System (ACS) [25, 110] algorithm is a successor of the AS and AQ. The main improvements are: pheromone trail updates are done offline, ants use a pseudo-random-proportional rule decision rule, and step by step updates are done online. For that reason it is simpler and more efficient than the AS and AQ.

The MAX-MIN Ant System (MMAS) [113] is also an extension of the AS with the following differences. Like in ACS, pheromone trail updates are done offline. To avoid stagnation, pheromone values are bounded by an upper and lower limit. Trails are initialized with the maximum possible amount of pheromone.

The rank-based Ant System (ASrank) [12] is an elitist variation of the AS. Here a rank of "best" ants is kept at all times and only they are allowed to deposit pheromone. At the end of each iteration the currently best solution is used to update pheromone.

The Hybrid Ant System (HAS) [36, 37] is the ACS updated with local search. Here local search is applied every time the ants build their solutions. Pheromone trails are updated according to these newly acquired locally optimal solutions.

The Approximate Nondeterministic Tree Search (ANTS) [86] algorithm is similar in structure to the tree-search algorithm [91]. The main difference is in its lack of a complete backtracking mechanism.

The Ant-Based Control (ABC) [102] algorithm was the first attempt to apply an ACO algorithm to a routing problem. A network can be represented by a directed graph. Here are ants used to build routing tables.

The AntNet [13] is another ant-based algorithm used for a routing problem.

The population-based Ant-Colony Optimization (PACO) [44] represents a new approach to problem solving compared to the standard ACO algorithms. Instead of the pheromone trails the PACO algorithm updates and maintains a population of solutions.

2.6 ACO-based algorithms for numerical optimization

So far there were only few ant-based approaches proposed for numerical optimization problems. The first one was the Continuous ACO (CACO) [8]. It comprises two levels: global and local. The CACO uses the ant-colony framework to perform local searches, whereas global search is handled by a GA. It was later followed by the Continuous Interacting Ant Colony (CIAC) [28], the Aggregation Pheromone System [119], the Improved Ant-Colony Algorithm [14], the Extended ACO for continuous and mixed-variable [104], etc.

Although these algorithms draw inspiration from the ACO metaheuristic, they do not follow it closely. One of the few algorithms that follow the ACO metaheuristic was proposed by Socha [104] and is called the Extended Ant-Colony Optimization (eACO). As it is an extension of a generic ACO, it can solve mixed discrete-continuous optimization problems. In the case of numerical optimization problems, the domain can change from discrete to continuous. The only adaptation needed is a move from using the discrete probability distribution to a continuous one. Instead of choosing a new component at step i, like ant-based algorithms usually do, the ants generate a random number according to a certain probability density function. As mentioned before, the probability distribution can be either discrete or continuous. In this way the eACO is capable of solving continuous and mixed-variable optimization problems.

2.7 Applications of ant-colony optimization algorithms

In Table 2.1 different applications of ant-colony optimization algorithms are shown (partially adapted from [23]). Of course, Table 2.1 cannot present a complete overview,

Table 2.1 Applications of ant-colony optimization algorithms.

Problem name / Authors	Algorithm name	Year
Traveling salesman		
Dorigo, Maniezzo and Colorni	AS	1991
Gambardella and Dorigo	Ant-Q	1995
Dorigo and Gambardella	ACS and ACS-3-opt	1996
Stützle and Hoos	MMAS	1997
Bullnheimer, Hartl and Strauss	ASrank	1997
Guntsch and Middendorf	PACO	2002
Quadratic assignment		
Maniezzo, Colorni and Dorigo	AS-QAP	1994
Gambardela, Taillard and Dorigo	HAS-QAP	1997
Stützle and Hoos	MMAS-QAP	1997
Maniezzo	ANTS-QAP	1998
Maniezzo and Colorni	AS-QAP	1999
Scheduling problems		
Colorni, Dorigo and Maniezzo	AS-JSP	1994
Stützle	AS-FSP	1997
Bauer et al.	ACS-SMTTP	1999
den Besten, Stützle and Dorigo	AS-VRP	1997
Vehicle routing		
Bullnheimer, Hartl and Strauss	AS-VRP	1997
Gambardella, Taillard and Agazzi	HAS-VRP	1999
Connection-oriented network routing		
Schoonderwoerd et al.	ABC	1996
Di Caro and Dorigo	AntNet-FS	1998
Bonabeau et al.	ABC-smart ants	1998
Connection-less network routing		
Di Caro and Dorigo	AntNet and AntNet-FA	1997
van der Put and Rothkrantz	ABC-backward	1998
Sequential ordering		
Gambardella and Dorigo	HAS-SOP	1997
Shortest common supersequence		
Michel and Middendorf	AS-SCS	1998
Frequency assignment		
Maniezzo and Carbonaro	ANTS-FAP	1998
Generalized assignment		
Ramalhinho, Lorenço and Serra	MMAS-GAP	1998
Multiple knapsack		
Leguizamón and Michalewicz	AS-MKP	1999
Optical network routing		
Navarro Varela and Sinclair	ACO-VWP	1999
Redundancy allocation		
Liang and Smith	ACO-RAP	1999
Mesh partitioning		
Korošec and Šilc	ACO	2002
Multi-criteria optimization		
Guntsch and Middendorf	PACO	2003
Multi-parameter optimization		
Bilchev and Parmee	CACO	1995
Monmarché, Venturini, and Slimane	API	2000
Dréo and Siary	CIAC	2002
Socha	eACO	2004
Korošec and Šilc	MASA	2005
Korošec and Šilc	DASA	2006

and the interested reader is referred to [11, 18, 111] for additional surveys.

2.8 A comparison with other nature-inspired algorithms

Nature inspired a number of modern optimization techniques [7]. The SA is modeled from the thermodynamic behavior of solids. The GA starts with a randomly generated population. Individuals of the population are updated with the use of crossover and mutation operators. Each individual is evaluated using a fitness function. The Neural Network (NN) is a computing paradigm that is loosely modeled after cortical structures of the brain. In most cases the NN is a distributed learning technique that changes its structure based on external or internal information that flows through the network.

All these nature-inspired algorithms share many common features with ACO. Random techniques are present in the update mechanism of ACO, SA, and GA. Interaction and self-organization are present in ACO and GA. Emergence capabilities are present in ACO and NN [20].

As a final note, in the recent past it has been shown that under certain conditions, some versions of ACO (e.g., Graph-based Ant System [45]) can find the optimal solution with a probability arbitrarily close to one [46, 112]. But in general, the problem of convergence to the optimal solution of a generic ACO algorithm will most probably remain open due to generality of the ACO metaheuristic. Nevertheless, this results put ACO to the same level as the SA or GAs in terms of a solution-finding capability.

As seen in Section 2.5, there are many ant-based algorithms that use only one colony of ants. In the next chapter we present an algorithm that, in contrast to these algorithms, applies multiple colonies.

3 The multiple ant-colonies approach: the mesh-partitioning problem

Combinatorial optimization problems, which appears in data-mining and text-mining, belong to the wider class of so-called clustering problems, which are concerned with the grouping of objects into homogeneous subgroups. For these problems, ant-based algorithms have also been proposed [48]. However, ant-based clustering differs from ACO in several fundamental respects:

- It draws its inspiration from the clustering behavior observed in real ants (not the foraging behavior, as in the case of ACO).
- In contrast to ACO, it is not a metaheuristic; it tackles only the specific task of clustering.
- Unlike ACO, it does not make use of artificial pheromone.
- It shows no synergetic effect, i.e., its performance is mostly independent of population size.

Rather than the ant-based clustering approach to combinatorial optimization, we discuss three approaches that are based on ACO metaheuristics. Informally, each of these approaches uses multiple ant colonies instead of only one (as is the case in ACO). We decided to evaluate these algorithms on a well-known *NP*-hard clustering problem called the mesh-partitioning problem [71, 99].

3.1 The mesh-partitioning problem

Many of the problems that arise in mechanical, civil, automobile, and aerospace engineering can be expressed in terms of partial differential equations and solved with the finite-element method. If a partial differential equation involves a function, f, then the purpose of the finite-element method is to determine an approximation to f. To do this the domain is put into the discrete form of a set of geometrical elements consisting of

nodes: a process known as meshing. The value of f is then computed for each of these nodes, and the solutions for the other points are interpolated from these values [16].

In real-world engineering problems, meshing is a demanding task because meshes usually have large numbers (e.g., hundreds of thousands) of elements. For this reason, the finite-element method is usually parallelized, which means the mesh is partitioned and distributed among several processors: a process known as *mesh partitioning*. To achieve high computational efficiency it is important that the mesh partitioning produces workloads that are well balanced and that the inter-processor communication is minimized. This is a combinatorial optimization problem and is a special case of the well-known graph-partitioning problem (see Figure 3.1).

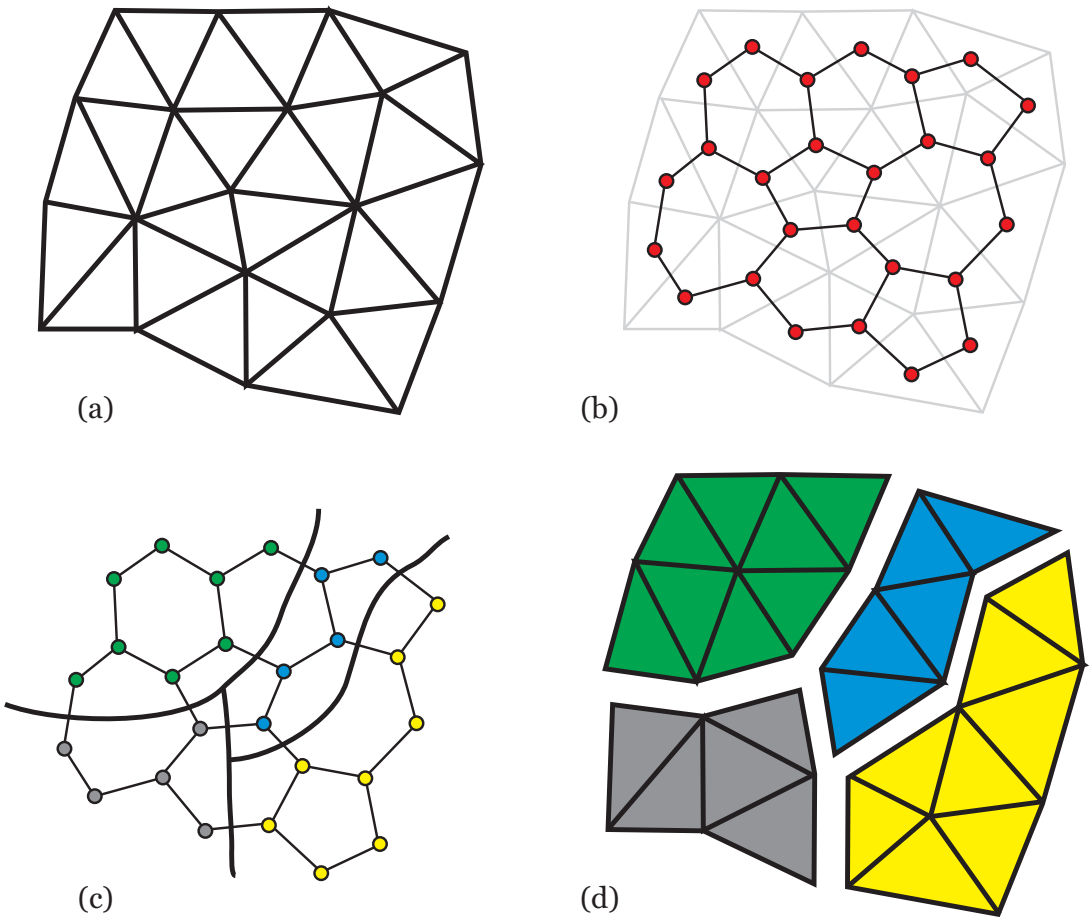

Figure 3.1 Mesh partitioning: (a) sample mesh, (b) mesh with induced graph, (c) after graph partitioning, and (d) the resulting partitioned mesh.

The graph-partitioning problem is defined as follows. Let $\mathcal{G}(V, E)$ be an undirected graph consisting of a non-empty set V of vertices and a set $E \subseteq V \times V$ of edges. A k-partition D of $\mathcal{G}$ comprises k mutually disjointed subsets $D_1, D_2, \ldots, D_k$ (called domains) of V, whose union is V. The set of edges that connect the different domains of a partition D is called an edge-cut, and is denoted by $\xi(D)$. A partition D is considered to be balanced if the sizes of the domains are roughly the same, i.e., if

$$\beta(D) = \max_{1 \leq i \leq k} |D_i| - \min_{1 \leq i \leq k} |D_i| \approx 0. \tag{3.1}$$

The graph-partitioning problem is to find a balanced partition with a minimum edge-cut.

Unfortunately, the graph-partitioning problem is an *NP*-hard optimization problem. This means that it is impossible to find optimum solutions in polynomial time (unless $P = NP$). Consequently, heuristic approaches are normally used for mesh partitioning. Many mesh-partitioning algorithms are described in the literature [29]. Some of these just use geometric information about the mesh and employ a fast, parallel, sorting algorithm in their implementation [103]. Spectral partitioning methods also make use of the mesh-connectivity information and solve the eigenvalue problem to compute the partition [95].

Recently, a number of studies have shown that stochastic heuristics have great potential for solving a wide range of problems [131]. Examples include TS [53], SA [118], NNs [4], and GAs [132]. These methods are widely applicable and have proven to be very powerful in practice [10]. But with ACO metaheuristics only a few attempts have been made [83]. In the following, an ACO-based mesh-partitioning method that originates from our previous work [59, 60, 62, 63, 70, 71, 72, 73, 74, 75, 76, 77, 99, 117] is presented.

3.2 The basic algorithm

There are k colonies of ants competing for food, which in this case represents the vertices of the graph (mesh). Eventually, the ants gather the food to their nests, i.e., they partition the mesh into k sub-meshes. More precisely, the Basic Multiple Ant-Colony Algorithm (B-MACA) proceeds as follows (see Algorithm 3.1).

Algorithm 3.1 Basic Multiple Ant-Colony Algorithm

```
1: Initialize
2: while not ending condition met do
3:   for all ants of colony do
4:     for all colonies do
5:       if carrying food then
6:         if in nest then
7:           DropFood
8:         else
9:           MoveToNest
10:        end if
11:      else if food here then
12:        PickUpFood
13:      else if food ahead then
14:        MoveForward
15:      else if in nest then
16:        MoveToAwayPheromone
17:      else if help signal then
18:        MoveToHelp
19:      end if
20:    end for
21:  end for
22:  for all grid cells do
23:    EvaporatePheromone
24:  end for
25: end while
```

Initially, the graph is mapped onto the grid that represents the ants' habitat (where the ants can move). There are many possibilities for this mapping, one of which is random mapping, as used in our case. Then the ants are evenly distributed to k nest loci on the grid, which are chosen according to a certain strategy, from where they start

their foraging and gathering of food. An ant can move in one of three directions (forward, left and right). The direction is chosen by a probability rule based on pheromone intensity. When an ant attempts to move off the grid it is forced to move left or right with equal probability. When an ant finds food it tries to pick it up. To do this, it checks whether the quantity of the temporarily gathered food in its nest is at the limit (the capacity of storage is limited by the problem's constraints). If the limit has not been reached, then the weight of the food is calculated from the number of cut edges created by assigning the selected vertex to the partition associated with the nest of the current ant; otherwise the ant moves in a randomly selected direction. If the food is too heavy for one ant to pick it up (and not too heavy for a few ants to pick it up) then the ant sends a help signal within a radius of a few cells. So, if other ants are in the neighborhood, they will help this ant to carry the food to the nest. On the way back to the nest an ant deposits pheromone on the trail that it is making, so the other ants can follow its trail and gather more food from that, or a nearby, cell. When an ant reaches the nest it drops the food in the first possible place around the nest (e.g., in a clockwise direction). After an ant has dropped its food, it starts a new round of foraging. Of course, ants can also gather food from other nest loci. When an ant tries to pick up food from other nest loci it performs the same procedure as when foraging for food, with the exception that when the food is too heavy to be picked up, the ant moves on instead of sending a help signal. In this way the temporary solution is significantly improved.

As mentioned above, there are some constraints imposed on the B-MACA algorithm. The first is the colony's storage-capacity constraint, which is implemented so that no single colony can gather all the food into its nest, and to maintain the appropriate balance between domains. The second constraint ensures that when the pheromone intensity of a certain cell drops below a fixed value, that cell's pheromone intensity is restored to the initial value. In this way we maintain a high exploration level. Other constraints are as follows: there can only be a limited number of vertices put in a single cell; each ant can carry only a limited number of pieces of food; the food that is being brought back to the nest is a kind of a tabu, i.e., it is not available to other ants. A short

tabu list consisting of the last m pieces of food that were moved helps the algorithm to escape from local minima.

3.3 The multilevel algorithm

One already-established way to speed up and globally improve the partitioning method is the use of multilevel techniques. Here, the basic idea is to group vertices together to form clusters that define a new graph. The next step is to recursively iterate this procedure until the graph size falls below a certain threshold. This is followed by a successive refinement of these coarser graphs. This procedure is known as the multilevel paradigm. The multilevel idea (see Figure 3.2) was first proposed by Barnard and Simon [6] as a method of speeding-up spectral bisection; it was improved by Hendrickson and Leland [50], who generalized it to encompass the local refinement algorithms.

The implementation consists of two parts: graph contraction and partition expansion. In the first part a coarser graph $\mathcal{G}_{\ell+1}(V_{\ell+1}, E_{\ell+1})$ is created from $\mathcal{G}_\ell(V_\ell, E_\ell)$ by finding the largest independent subset of graph edges and then collapsing them. Each selected edge is collapsed and the vertices $u_1, u_2 \in V_\ell$ that are at either end of it are merged into the new vertex $v \in V_{\ell+1}$ with weight $|v| = |u_1| + |u_2|$. The edges that have not been collapsed are inherited by the new graph $\mathcal{G}_{\ell+1}$ and the edges that have become duplicated are merged and their weight is summed. Because of inheritance, the total weight of the graph remains the same and the total edge weight is reduced by an amount equal to the weight of the collapsed edges, which have no impact on the graph balance or the edge cut.

In the second part the already-optimized partition (with the B-MACA) of graph $\mathcal{G}_\ell$ is expanded. The optimized partition must be interpolated onto its parent graph $\mathcal{G}_{\ell-1}$. Because of the simplicity of the coarsening in the first part, the interpolation itself is trivial. If vertex $v \in V_\ell$ belongs to domain D_i, then after refinement the matched pair $u_1, u_2 \in V_{\ell-1}$ that represents v, will also be in D_i. In this way the graph is expanded to its original size, and on every level ℓ of the expansion we run our basic ant-colony algorithm. This is referred to as the Multilevel Multiple Ant-Colony Algorithm (M-MACA) approach.

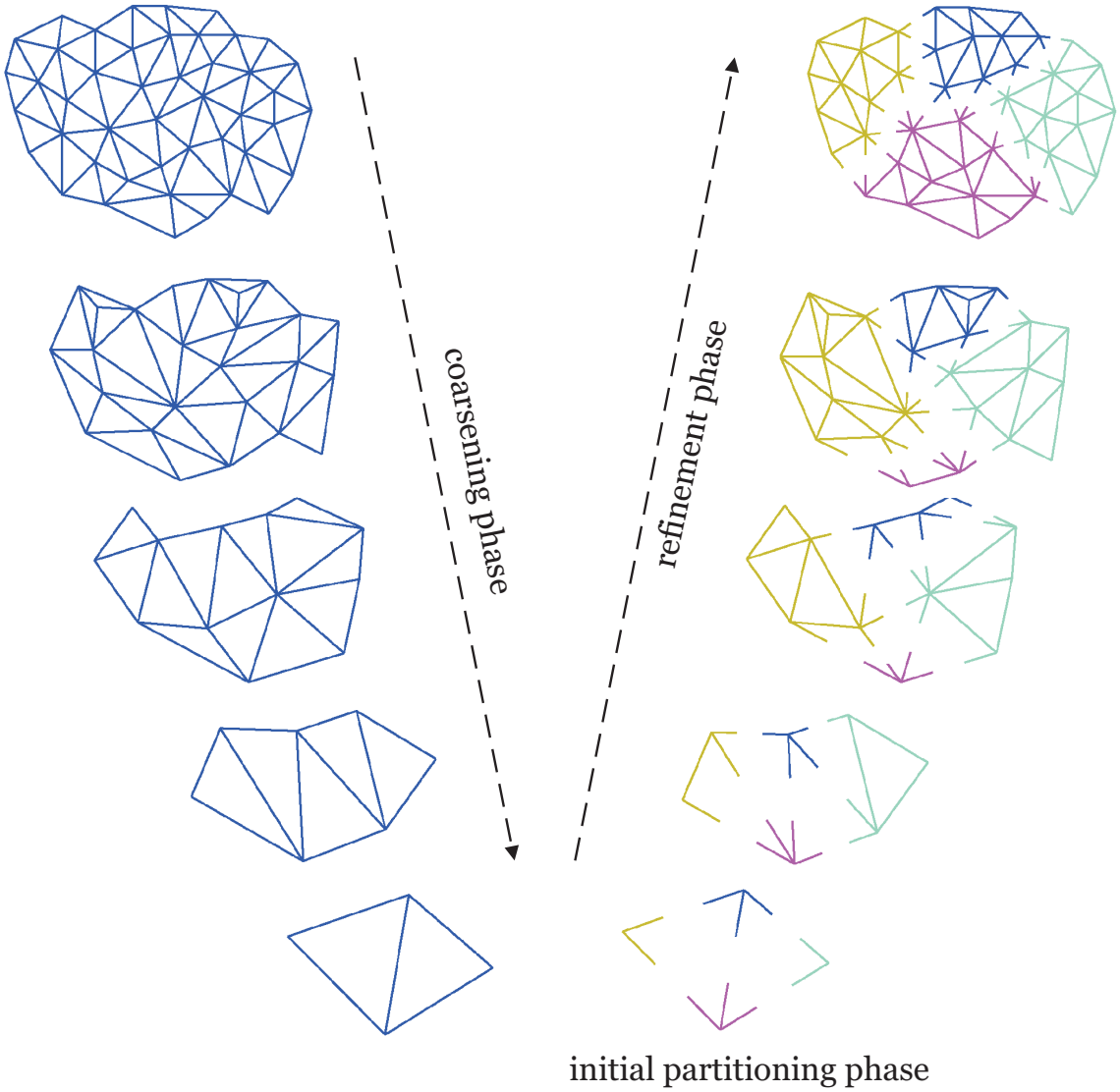

Figure 3.2 The three phases of multilevel k-way partitioning.

Due to the large graphs and the increased number of levels, the number of vertices in a single cell increases rapidly. To overcome this problem we introduced a method called bucket sort that accelerates and improves the algorithm's convergence by choosing the most "promising" vertex from the cell. The bucket sort, which was first introduced by Fiduccia and Mattheyses [31], has become an essential tool for the efficient and rapid sorting and adjustment of vertices in terms of their gain. The basic idea is that all the vertices with a particular gain g are put together in a "bucket" ranked g. In this way the problem of finding a vertex with maximum gain is converted into finding the non-empty bucket with the highest rank, and then picking a vertex from it. If

a chosen vertex migrates from one domain to another, only its gain and the gains of all its neighbors have to be recalculated and put back into appropriate buckets. In our implementation each bucket is represented by a double-linked list of vertices. Because of the multilevel process, it often happens that the potential gain values are dispersed over a wide range. For this reason we have introduced the 2-3 tree, and so avoided large and sparse arrays of pointers. We store the non-empty buckets in the 2-3 tree, so each leaf in the tree represents a bucket. For an even faster search we have created one 2-3 tree for each colony on every cell that has vertices on it (see Figure 3.3). In this way we have increased the speed of the search, as well as the add and delete operations.

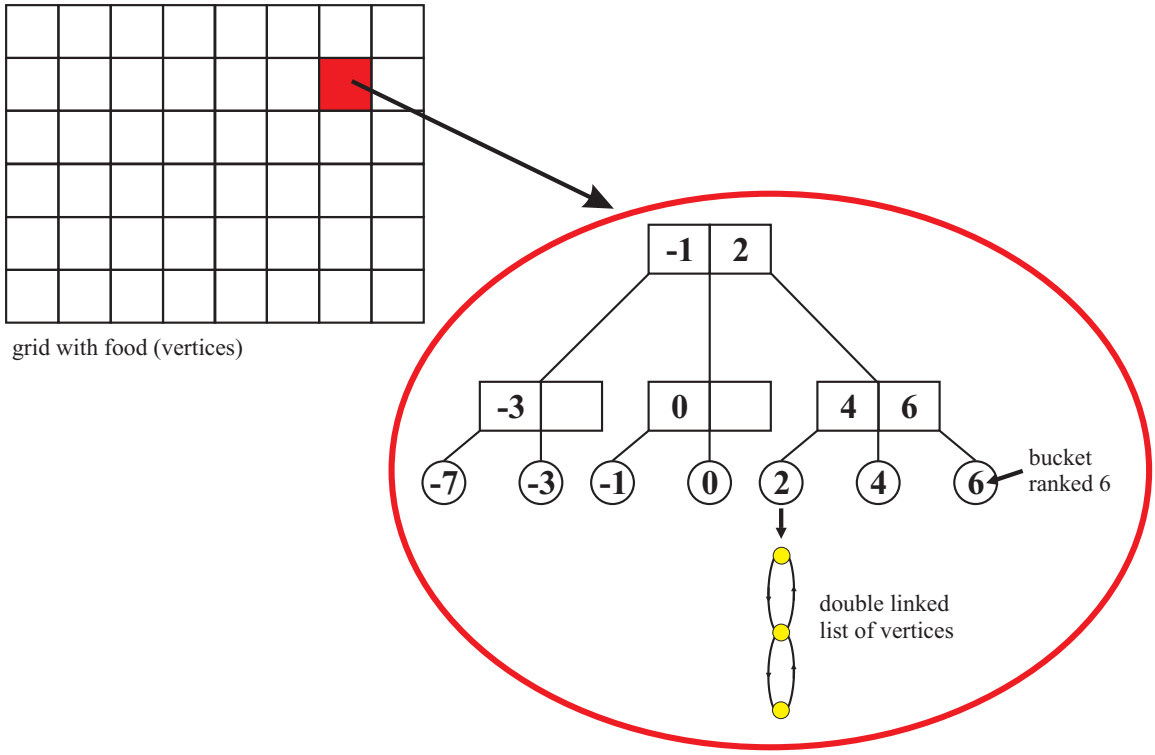

Figure 3.3 Representation of 2-3 tree used to speed up bucket sorting.

3.4 The hybrid algorithm

We have merged the Vector Quantization (VQ) and the basic ant-colony algorithm into a single algorithm called the Hybrid Multiple Ant-Colony Algorithm (H-MACA) [117]. With the VQ we compute a partition, which is then used as a starting partition for the

B-MACA. With the B-MACA we refine this partition so that the best possible result is obtained.

The VQ method [84] is a stochastic approximation method that uses the basic structure of the input vectors to solve a specific problem (for example, data compression). In other words, the input space is divided into a finite number of regions (domains) and for each region there is a representative vector. When a mapping function (device) receives a new input vector it maps it into a region that best represents this vector. This is a simple example of some kind of compression. Of course this is only one possibility of using this method. We used it as a mapping device for the mesh-partitioning problem. The mesh vertices are usually locally connected to their neighbors. Now we can treat the position of each mesh vertex as an input vector and each domain in our partition as a region in input space. We try to divide the "mesh" space into domains, so that the size (the number of vertices) of each domain is approximately the same, with as few as possible connections between the domains. A vector quantizer maps ℓ-dimensional vectors in the vector space $\mathbb{R}^\ell$ into a finite set of vectors

$$Y = \{\mathbf{y}_i : i = 1, 2, \ldots, k\}. \tag{3.2}$$

Each vector $\mathbf{y}_i$ is called a codeword and the set of all the codewords is called a codebook. Associated with each codeword $\mathbf{y}_i$ is a nearest-neighbor region called the Voronoi region [123], which is defined by:

$$V_i = \left\{\mathbf{x} \in \mathbb{R}^\ell : \|\mathbf{x} - \mathbf{y}_i\| \leq \|\mathbf{x} - \mathbf{y}_j\|, \forall i \neq j\right\}. \tag{3.3}$$

The set of Voronoi regions partition the entire space $\mathbb{R}^\ell$ such that

$$\left(\bigcup_{i=1}^{k} V_i = \mathbb{R}^\ell\right) \wedge \left(\bigcap_{i=1}^{k} V_i = \emptyset\right). \tag{3.4}$$

An example of the VQ is shown in Figure 3.4. Here we used a two-dimensional graph ($\ell = 2$), but it can easily be expanded to any other number of dimensions. We can see 45 input vectors that are divided into $k = 13$ domains (Voronoi regions $V_1, V_2, \ldots, V_{13}$) and are represented by the codewords $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{13}$.

The VQ consists of the following six steps:

Step 1: Set the number of domains k according to a given problem.

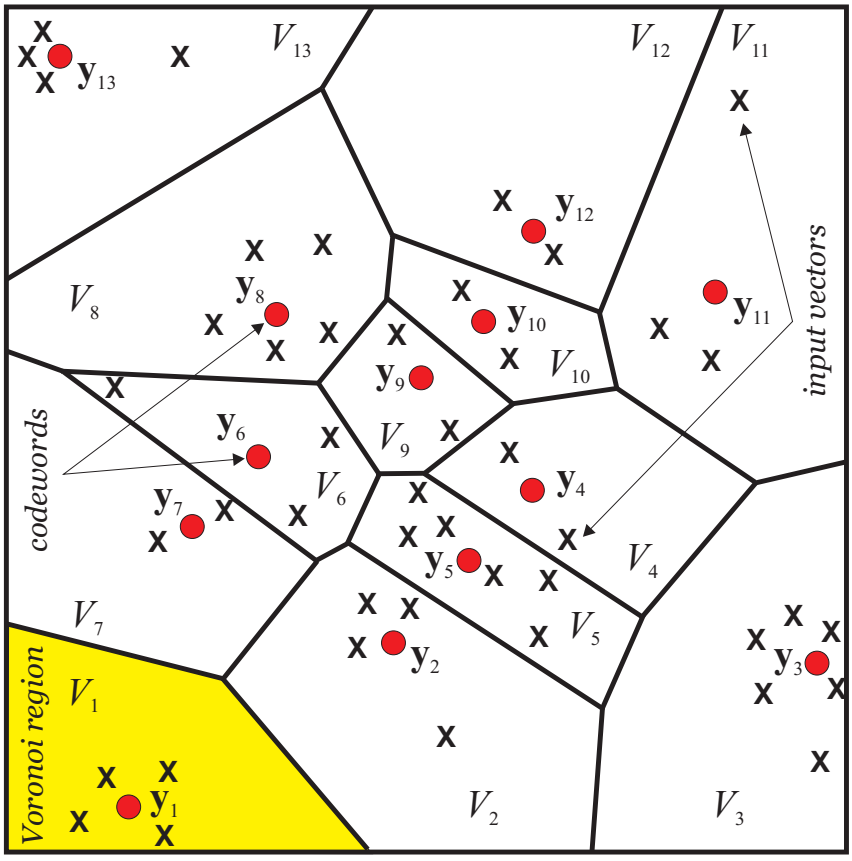

Figure 3.4 An example of vector quantization problem.

Step 2: Read the input graph and its coordinates. The input vector consists of three elements. The first two are the vertex coordinates and the third is the vertex density. By density we mean how close together its connected neighbors are: the closer the neighbors, the higher the vertex density.

Step 3: Select k codewords. The initial codewords can be selected randomly from the input vectors or as random points in the input space.

Step 4: Calculate the Euclidian distance between the input vector and the codewords. The input vector is assigned to the domain of the codeword that returns the minimum value according to the function:

$$f(\mathbf{x}, \mathbf{y}_i) = \varepsilon(\mathbf{x}, \mathbf{y}_i) - \xi_i - \beta_i, \tag{3.5}$$

where $\varepsilon(\mathbf{x}, \mathbf{y}_i)$ is a function that calculates the Euclidian distance between $\mathbf{x}$ and $\mathbf{y}_i$, ξ_i represents the change in the edge-cut if $\mathbf{x}$ belonged to the i-th domain, and β_i

denotes the difference between the number of vertices in the largest and the i-th domains.

Step 5: Compute the new set of codewords. We add up all the $\mathbf{x}_i$ vectors in the i-th domain and divide the summation by the number of input vectors in the domain:

$$\mathbf{y}_i = \frac{1}{m} \sum_{j=1}^{m} x_{ij}, \tag{3.6}$$

where m represents the number of input vectors in the i-th domain.

Step 6: Repeat steps 4 and 5 until the values of the codewords converge, usually to a suboptimal solution.

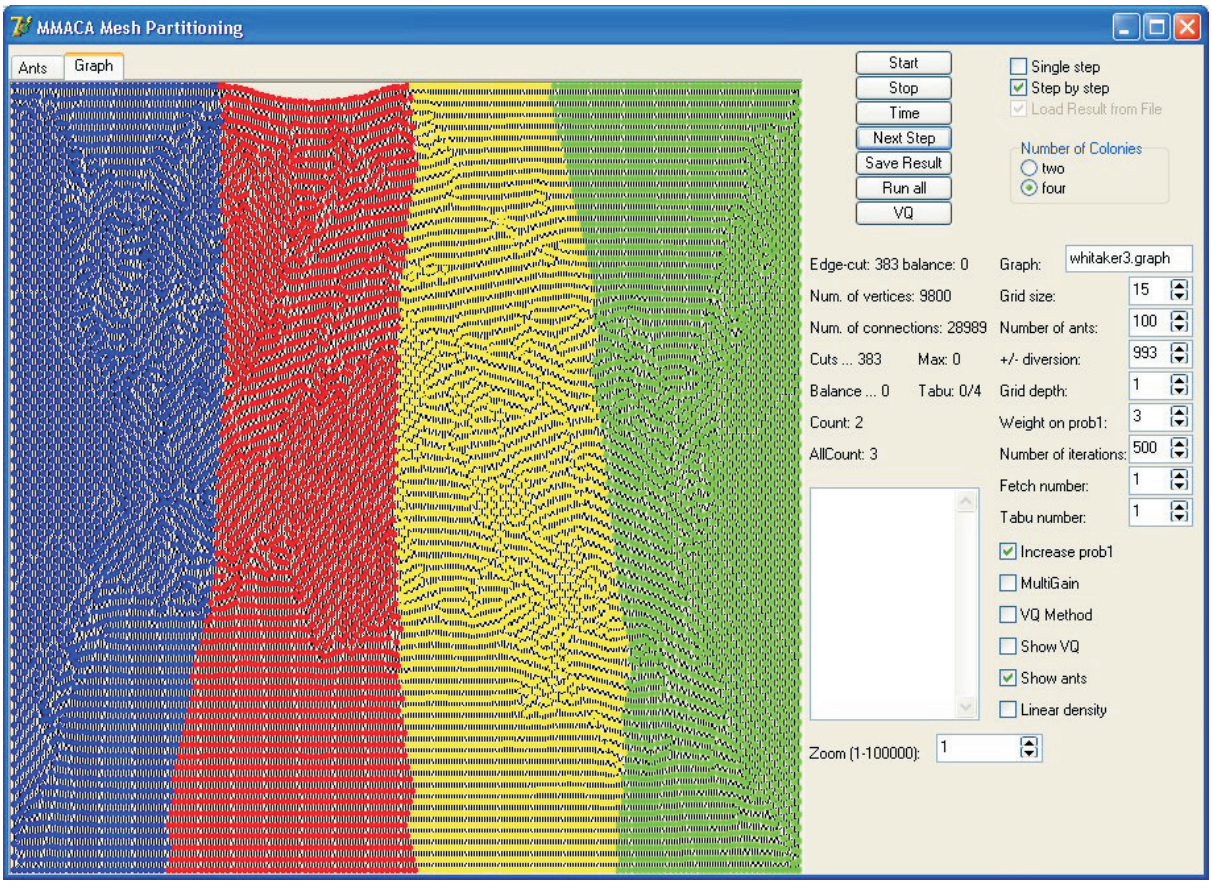

Figure 3.5 Output of the MACA algorithm: a graph partitioned into four domains.

3.5 Performance evaluation

3.5.1 The experimental environment

We implemented the B-MACA, M-MACA, and H-MACA in Borland® Delphi™. The experiments were made on a computer with an AMD Athlon™ XP 1800+ processor running the Microsoft® Windows® XP operating system. The implementation also includes a visualization tool to assist the user in selecting the appropriate parameters of the algorithm (see Figures 3.5–3.7).

Figure 3.5 shows what the output of the MACA-type algorithm looks like. Here we can see how the algorithm partitioned the graph into four domains and which vertices belong to which partition.

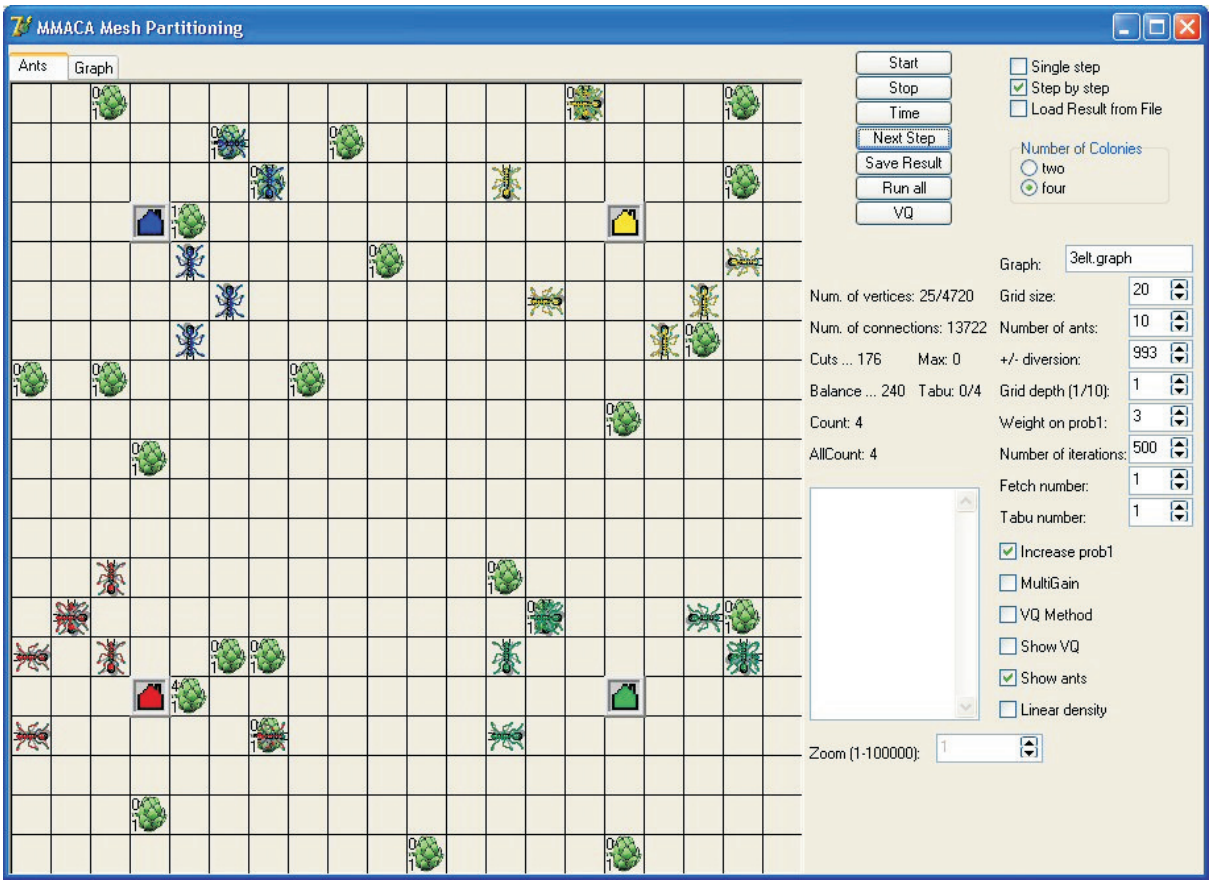

Figure 3.6 MACA interface: food on the grid at the beginning of the algorithm run.

Figure 3.6 represents what the grid looks like after four iterations of the algorithm. As we can see, ants are foraging for food, which is scattered all over the grid. One unit

of food represents a graph vertex which in the case of the M-MACA can be composed of many vertices of the initial graph. In Figure 3.7 we can see that food is laid around the nests and ants are "stealing" food from other nests.

On the right-hand side of the grid we can see the instruments that the algorithm is controlled with. We are able to set the grid size (Grid size), the number of ants per colony (Number of ants), the desired balance between domains (± diversion), the maximum number of iterations without improvement per level (Number of iterations), the weight of heuristic information (Weight on prob1), the maximum number of gathered food pieces (Fetch number), and the maximum length of the tabu list (Tabu number). All the other data represent the current state of the algorithm.

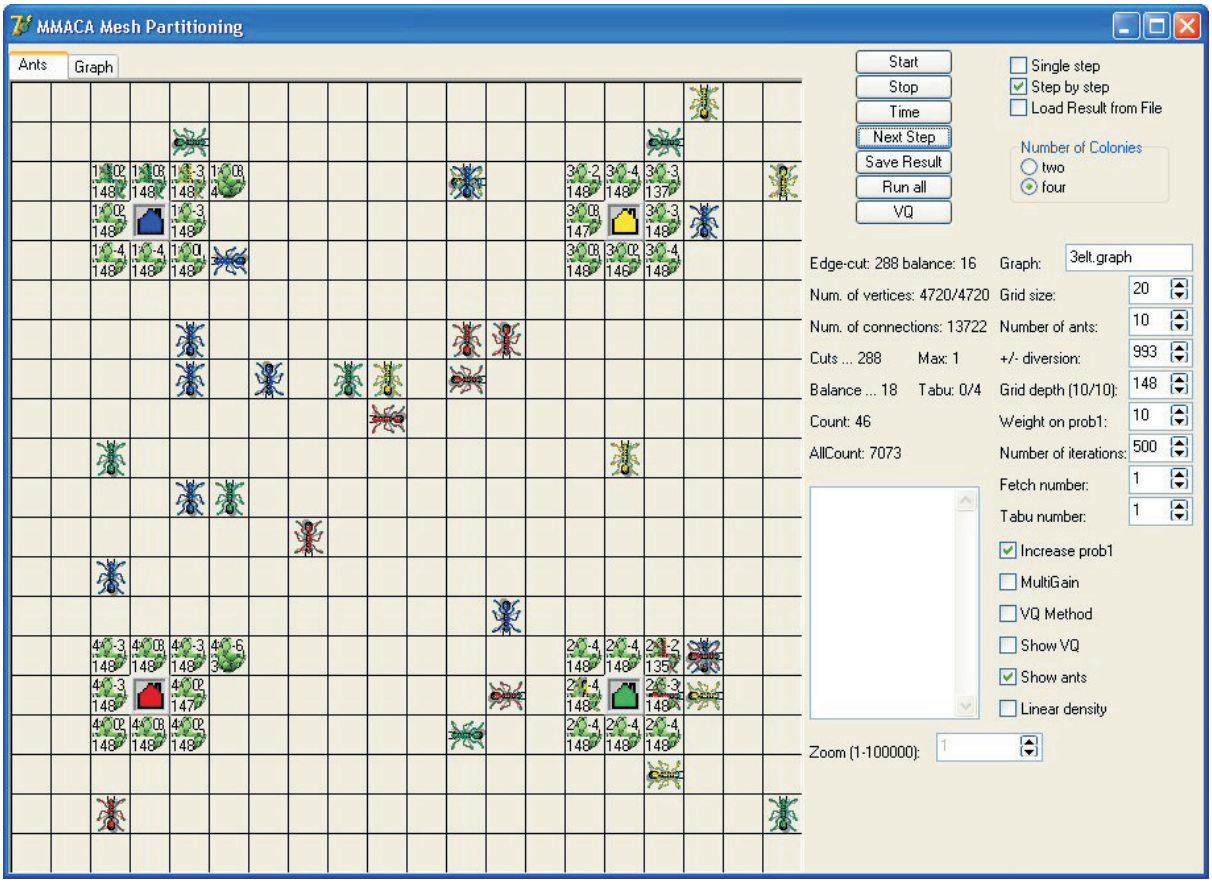

Figure 3.7 MACA interface: food on the grid at the end of the algorithm run.

Table 3.1 Benchmark graphs.

Graph	Number of vertices $\|V\|$	Number of edges $\|E\|$
graph1*	50	86
graph2*	50	143
grid1***	252	476
grid1_dual***	224	420
graph3*	400	546
graph4*	400	1,006
netz4504_dual***	615	1,171
U 1000.5***	1,000	2,394
U1000.10***	1,000	4,696
U1000.20***	1,000	9,339
ukerbe1_dual***	1,866	3,538
netz4504***	1,961	2,578
add20**	2,395	7,462
data**	2,851	15,093
grid2_dual***	3,136	6,112
grid2***	3,296	6,432
airfoil***	4,253	12,289
3elt**	4,720	13,722
uk**	4,824	6,837
add32**	4,960	9,462
ukerbe1***	5,981	7,852
airfoil_dual***	8,034	11,813
bcsstk33**	8,738	291,583
3elt_dual***	9,000	13,287
whitaker3**	9,800	28,989
crack**	10,240	30,380
wing_nodal**	10,937	75,488
fe_4elt2**	11,143	32,818
4elt**	15,606	45,878
big***	15,606	45,878
fe_sphere**	16,386	49,152
cti**	16,840	48,232
whitaker3_dual***	19,190	28,581
crack_dual***	20,141	30,043
cs4**	22,499	43,858
big_dual***	30,269	44,929

* Randomly generated

** Graph Partitioning Archive: staffweb.cms.gre.ac.uk/∼c.walshaw/partition/

*** Graph Collection: www.uni-paderborn.de/cs/ag-monien/RESEARCH/PART/graphs.html

3.5.2 The benchmark suite

For the purpose of testing, we have randomly generated four graphs. Meanwhile, all the other benchmark graphs were taken from the Graph Partitioning Archive at the University of Greenwich and from the Graph Collection Web page at the University of Paderborn. All the graphs are listed in Table 3.1.

3.5.3 The basic algorithm

Several experiments were performed to evaluate the B-MACA [70, 71]. We compared the B-MACA with the well-known partitioning programs k-Metis 4.0 [56], Randomized Tabu Search (RTS) [116], and Hybrid Genetic Algorithm (HGA) [116]. Each test graph was partitioned into two and four domains ($k = 2$ and $k = 4$). The results, which are given in terms of edge cut $\xi(D)$ and balance $\beta(D)$, are shown in Table 3.2.

Table 3.2 Experimental results (B-MACA, $k = 2, 4$).

Algorithm		k-Metis 4.0		RTS		HGA		B-MACA	
Graph	k	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$
graph1	2	15	0	14	0	14	0	13	2
	4	32	12	25	1	25	1	25	1
graph2	2	35	0	33	0	33	0	33	0
	4	77	9	57	1	57	1	57	1
graph3	2	45	8	41	0	40	0	41	0
	4	85	4	82	0	76	0	79	0
graph4	2	208	10	181	0	187	0	182	0
	4	331	2	309	0	310	0	316	2

Table 3.2 shows that the B-MACA performs well on small graphs ($|V| < 500$). We ran additional experiments [70, 71] on larger graphs ($|V| > 1{,}000$) and found that the B-MACA performs much worse than the RTS or HGA. To improve its performance, we enhanced the B-MACA with a multilevel technique.

3.5.4 The multilevel algorithm

We again partitioned each of the test graphs into two and four domains ($k = 2$ and $k = 4$). The results are again pairs, $\xi(D)$ and $\beta(D)$. We allowed a 0.2% unbalance, i.e., $\beta(D) \leq 0.002|V|$. The M-MACA was run 30 times on each graph; and from the

solutions obtained we chose the one with minimum $\xi(D)$ as the final result for that graph.

The results of our experiments are shown in Table 3.3, where the M-MACA is compared with the k-METIS 4.0 [56], the Chaco 2.0 [50] with multilevel Kernighan-Lin global partitioning method, and the new mixed simulated annealing and tabu search algorithm MLSATS [5].

Table 3.3 Experimental results (M-MACA, $k = 2, 4$).

Algorithm		k-Metis 4.0		Chaco 2.0		MLSATS		M-MACA	
Graph	k	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$
add20	2	773	23	630	1	696	119	601	5
	4	1,214	45	1,242	1	1,193	57	1,196	3
data	2	253	67	210	1	196	131	199	1
	4	486	42	444	1	395	67	424	1
3elt	2	148	72	124	0	87	108	90	0
	4	250	63	258	0	199	45	225	2
uk	2	33	126	23	0	54	186	20	2
	4	50	47	60	0	261	74	50	2
add32	2	12	62	11	0	10	2	10	2
	4	38	59	53	0	35	66	40	5
bcsstk33	2	13,393	139	10,199	0	10,064	100	10,222	12
	4	22,909	129	25,529	1	22,442	219	22,632	14
whitaker3	2	135	96	135	0	130	0	126	16
	4	407	97	439	0	448	198	386	6
crack	2	221	248	209	0	184	62	184	0
	4	478	106	457	0	401	216	381	8
wing_nodal	2	1,892	315	1,747	1	1,670	543	1,711	23
	4	3,898	163	3,817	1	3,596	273	3,619	11
fe_4elt2	2	132	297	130	1	130	1	130	1
	4	400	87	378	1	350	113	350	2
4elt	2	149	104	242	0	130	1	130	1
	4	385	197	416	1	367	369	389	13
fe_sphere	2	444	418	422	0	384	576	402	4
	4	828	46	835	1	786	329	810	15
cti	2	401	300	369	0	318	120	332	24
	4	1,104	156	1,000	0	966	360	1,045	14
cs4	2	397	653	418	1	418	907	397	1
	4	1,102	312	1,135	1	1,103	403	1,038	21

Table 3.4 Results from the Graph Partitioning Archive.

Graph	k	Algorithm(s)	$\xi(D)$ with 1% unbalance
add20	2	GCSVD	599
	4	M-MACA	1,186
data	2	AMG	188
	4	M-MACA	410
3elt	2	GrPart, M-MACA	89
	4	JE, MLSATS	199
uk	2	AMG, M-MACA	19
	4	JE	44
add32	2	J2.2, MLSATS, M-MACA	10
	4	JE	33
bcssth33	2	GrPart	10,109
	4	iJ	21,685
whitaker3	2	JE, M-MACA	126
	4	JE	380
crack	2	AMG, M-MACA	183
	4	JE	368
wing_nodal	2	MQI	1,696
	4	JE	3,572
fe_4elt2	2	MRSB, MLSATS, M-MACA	130
	4	JE	349
4elt	2	GrPart	138
	4	JE	327
fe_sphere	2	JE	386
	4	N/A	768
cti	2	JE, MLSATS, M-MACA	318
	4	M-MACA	963
cs4	2	JE	367
	4	JE	940

AMG : A bisection algorithm based on classical Algebraic MultiGrid from S. Wishko, A. Brandt and D. Ron

GCSVD : Gene Correlation with Singular Value Decomposition [88]

GrPart : A. Kozhushkin's implementation of iterative multilevel Kernighan-Lin

iJ : Iterated JOSTLE - iterated multilevel Kernighan-Lin (k-way) [126]

J2.2 : JOSTLE - multilevel Kernighan-Lin (k-way); version 2.2 [126]

J2.2 JE : JOSTLE Evolutionary - combined evolutionary/multilevel scheme [105]

MLSATS : MultiLevel refinated mixed Simulated Annealing and Tabu Search [5]

MRSB : Multilevel Recursive Spectral Bisection [6]

MQO : Max-flow Quotient-cut Improvement, a bisection algorithm, which uses many multiple tries and improves an initial partition provided by METIS [82]

M-MACA : Multilevel Multiple Ant-Colony Algorithm [70, 71]

Clearly, the M-MACA performed very well, since it is superior to the classical k-METIS and Chaco algorithms. Even though the MLSATS produced some results that have a better $\xi(D)$, they had a much higher $\beta(D)$ than that produced by the M-MACA.

The M-MACA also returned some solutions that are better than currently available (Autumn 2006) solutions in the Graph Partitioning Archive (Table 3.4). Furthermore, the M-MACA is even comparable with the combined evolutionary/multilevel scheme used in the JOSTLE Evolutionary algorithm [105], which is currently the most promising mesh-partitioning algorithm.

3.5.5 The hybrid algorithm

Now we present and discuss the results of the experimental evaluation of the H-MACA in comparison with the well-known partitioning programs p-Metis 4.0 [56], Chaco 2.0 [50] with the multilevel Kernighan-Lin global partitioning method, and the M-MACA [74].

We partitioned each of the graphs into two and four domains ($k = 2$ and $k = 4$). Each score is described with the best obtained edge-cut, $\xi(D)$. It is important to mention that the balance $\beta(D)$ was kept inside 0.2% of the $|V|$. The results of our experiments are shown in Table 3.5. It can be seen that in most cases the best partition was obtained with the H-MACA algorithm.

Table 3.5 Experimental results (H-MACA, $k = 2, 4$).

Algorithm		Chaco 2.0		p-Metis 4.0		M-MACA		H-MACA	
Graph	k	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$	$\xi(D)$	$\beta(D)$
3elt	2	124	0	98	0	90	0	90	0
	4	258	0	252	0	225	2	212	4
3elt_dual	2	70	0	70	0	44	6	45	8
	4	130	0	120	0	112	4	154	7
airfoil	2	82	1	85	1	74	1	81	1
	4	182	1	179	1	176	2	190	3
airfoil_dual	2	60	0	40	0	37	0	40	14
	4	111	1	84	1	80	7	110	7
big	2	242	0	165	0	141	0	139	0
	4	416	1	405	1	354	11	382	14
big_dual	2	92	1	92	1	78	1	77	11
	4	219	1	196	1	215	18	222	25
crack	2	209	0	206	0	184	0	196	2
	4	457	0	458	0	377	6	371	9
crack_dual	2	130	1	101	1	80	25	87	1
	4	228	1	201	2	169	3	164	9
grid1	2	26	0	20	0	18	0	18	0
	4	48	0	40	0	38	0	38	0
grid1_dual	2	16	0	16	0	16	0	16	0
	4	37	0	35	0	35	0	35	0
grid2	2	38	0	37	0	34	0	34	2
	4	106	0	121	0	94	2	92	2
grid2_dual	2	35	0	32	0	32	0	32	0
	4	99	0	91	0	90	2	96	4
netz4504	2	25	1	26	1	22	1	24	1
	4	66	1	62	1	50	1	49	3
netz4504_dual	2	21	1	21	1	19	1	19	1
	4	54	1	49	1	44	2	44	2
U1000.5	2	10	0	1	0	1	0	2	0
	4	20	0	6	0	7	2	12	2
U1000.10	2	115	0	56	0	39	0	39	0
	4	200	0	108	2	99	2	107	2
U1000.20	2	294	0	253	0	220	4	221	2
	4	554	0	515	2	546	2	497	2
ukerbe1	2	30	1	28	1	27	1	28	1
	4	82	1	64	1	63	2	61	1
ukerbe1_dual	2	25	0	25	0	22	0	22	0
	4	56	1	51	1	52	3	48	1
whitaker3	2	135	0	128	0	126	16	127	0
	4	439	0	424	0	383	0	383	2
whitaker3_dual	2	82	0	74	0	64	18	65	0
	4	251	1	210	1	200	6	195	12

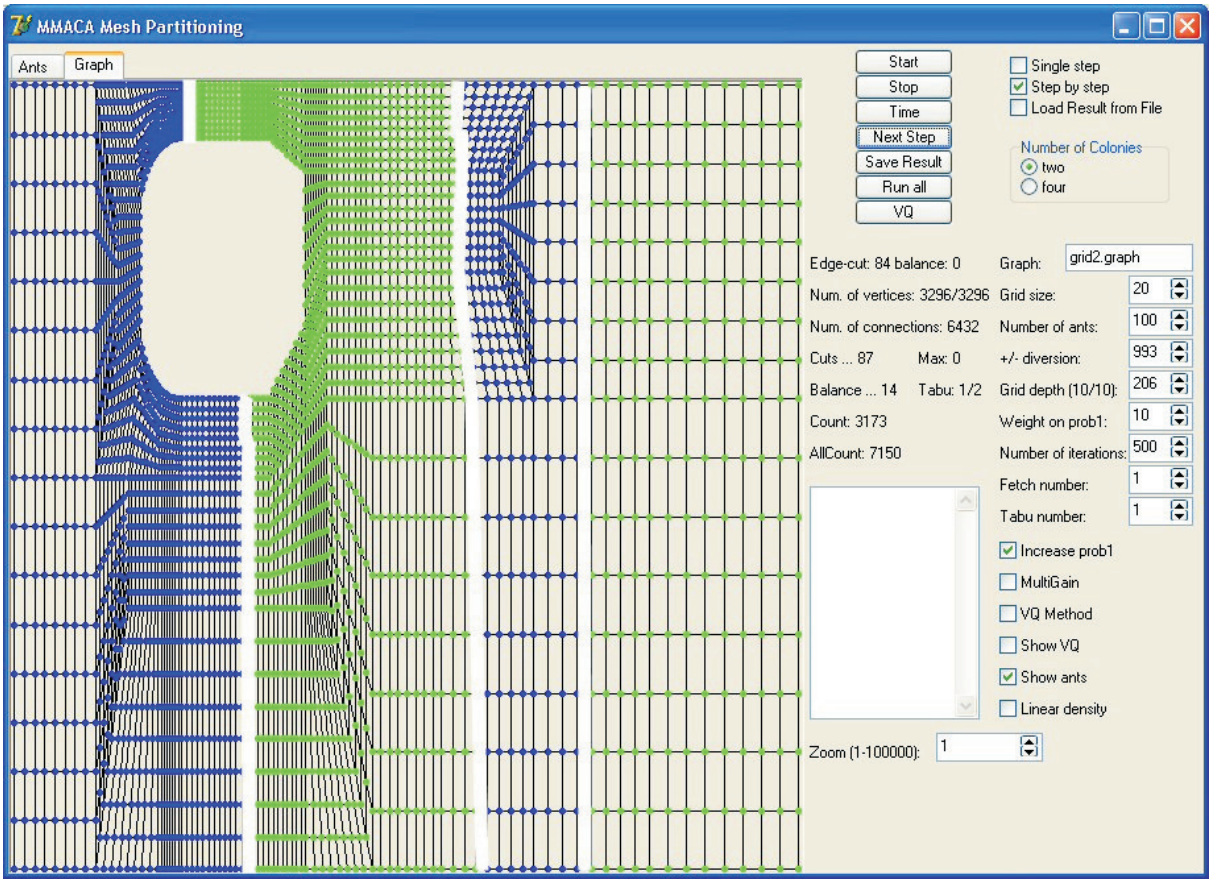

Figure 3.8 Mesh-partitioning visualization: M-MACA.

Figures 3.8 and 3.9 show the main drawback of the M-MACA. We can see vertices represented as blue and green dots, where blue dots belong to the one domain, and green dots to the other. With the solid white line we have emphasized the border between these two domains. The final solution obtained with the M-MACA includes "islands" (i.e., sets of connected vertices of the same color) that belong to different domains. The islands are due to a bad initial partition and the inability of the M-MACA to merge the islands into homogeneous regions (each having only one border). This drawback is eliminated by using the VQ to obtain the initial partitions. In Figure 3.8 (M-MACA) one can see four such islands, whereas there are only two in Figure 3.9 (H-MACA), which is a desired situation.

The multilevel approach has been shown to be a very useful and powerful addition to ant-colony metaheuristics. With this experience in mind, we decided to test

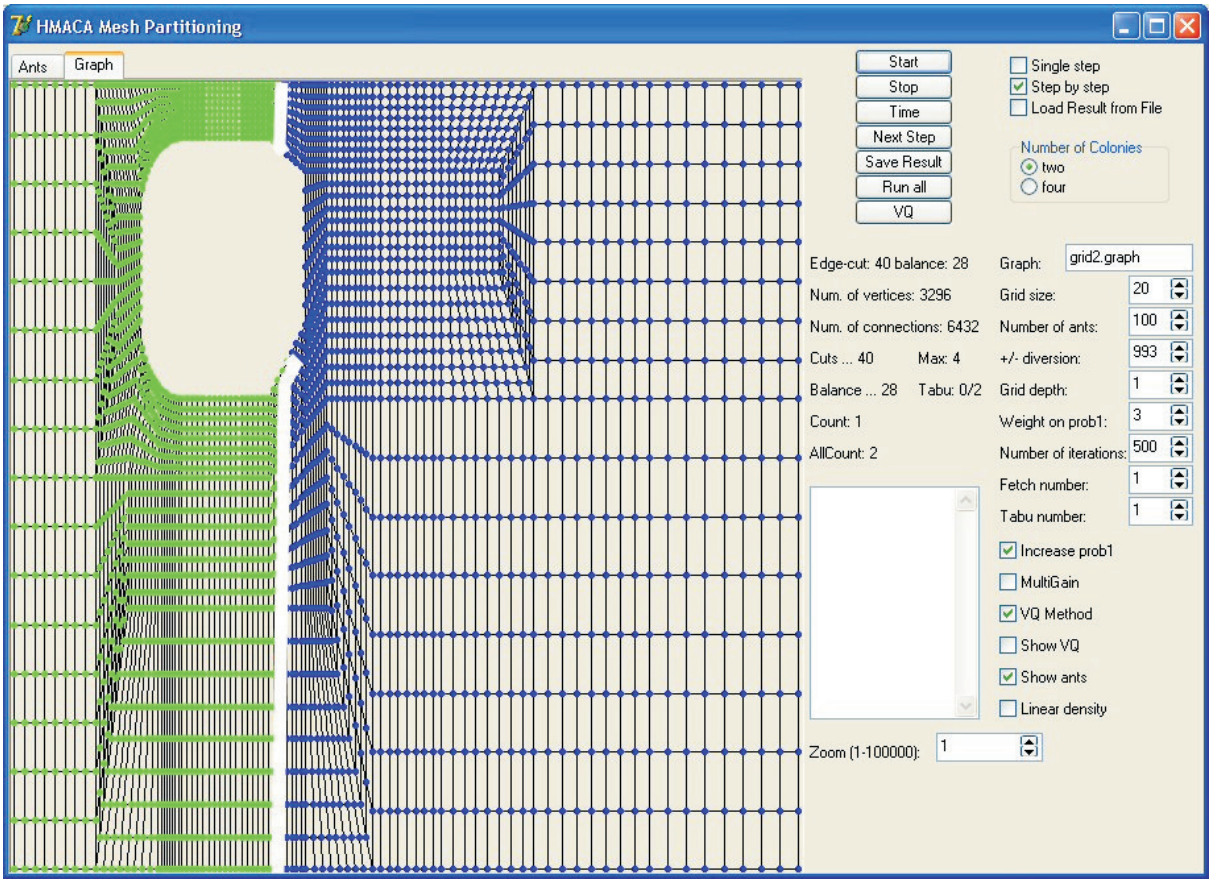

Figure 3.9 Mesh-partitioning visualization: H-MACA.

the combination of a multilevel approach and ant-colony metaheuristics on numerical optimization problems in the next chapter.

4 The multilevel ant-stigmergy approach

In the past decades, different kinds of optimization algorithms have been designed and applied to solve multi-parameter function-optimization problems. Some of the popular approaches are multi-parameter GAs [129], Evolution Strategies (ES) [19], Differential Evolution (DE) [107], Particle Swarm Optimization (PSO) [57], classical methods such as the quasi-Newton method [98], other non-evolutionary methods such as SA [58], TS [39], and most recently, ant-colony-based algorithms [65, 66, 67].

Algorithms inspired by ant-colony behavior are becoming increasingly popular in computer-science and operational research. However, a direct application of the ACO to a multi-parameter optimization problem is difficult. In this chapter we will show a new implementation of an ant-colony metaheuristic on numerical, multi-parameter optimization problems that are often solved by algorithms for continuous optimization.

Generally, ant-based algorithms solve different problems by means of a graph representation. We decided to use a search graph as shown in Figure 4.1.

Because of the nature of ant-based algorithms we first had to put the continuous multi-parameter problem into discrete form. More precisely, if a parameter p_i has a range from L_i to U_i and the discrete step is Δ_i, then a discrete parameter p_i has

$$n_i = \left\lceil \frac{U_i - L_i}{\Delta_i} \right\rceil + 1 \tag{4.1}$$

discrete values.

Now that we have a discrete form of the problem we are able to apply the new optimization approach—the so-called *multilevel ant-stigmergy approach*—to solve the discrete multi-parameter optimization problems.

4.1 Problem representation

A search graph is defined as a connected, directed, non-weighted, acyclic graph. It is also rooted and ordered. We translate all the discrete parameter values into a search graph. For this purpose we define a search graph

$$\mathcal{G} = (V, E) \tag{4.2}$$

with a set of vertices

$$\begin{aligned} &V = \bigcup_{d=1}^{D} V_d, \\ &V_d = \left\{ v_{\langle d,1 \rangle}, \ldots, v_{\langle d,n_d \rangle} \right\} \end{aligned} \tag{4.3}$$

and set of edges between the vertices

$$\begin{aligned} &E = \bigcup_{d=1}^{D} E_d, \\ &E_d = \left\{ e_{\langle d-1,i \rangle, \langle d,j \rangle} = \left(v_{\langle d-1,i \rangle}, v_{\langle d,j \rangle} \right) \ | v_{\langle d-1,i \rangle} \in V_{d-1} \wedge v_{\langle d,j \rangle} \in V_d \right\}, \end{aligned} \tag{4.4}$$

where D represents the length of the longest path in the search graph, which equals the number of parameters, and n_d represents the number of discrete values of a parameter p_d.

In Figure 4.1 we see that $v_{\langle 1,1 \rangle}$ represents the first discrete value of the first (can be randomly chosen) parameter and $v_{\langle 1,2 \rangle}$ represents the second discrete value, and so on. Every vertex at distance $d-1$ is connected to all the vertices at distance d. With this search graph we have covered the whole solution space of the discrete multi-parameter problem. For example, if we start in the $start$ vertex ($d = 0$) and follow the search graph to the ending vertex ($d = D$) we always get a path of length D. This path consists of D vertices and each vertex belongs to one of the parameters. So what we have here is one possible solution of the multi-parameter function. In this way we can create any solution from the solution space of a discrete problem. The efficiency of the path depends on how good is the result obtained by the parameter values found on the path. We call this translated problem the problem of finding the cheapest path. This type of solution creation is very suited to the ant-based approach. One more thing that we have to do is to define attributes for each vertex. In our case each vertex has two different types of attributes: one is a constant and represents the discrete parameter value, while the other is a variable and represents the amount of pheromone, τ.

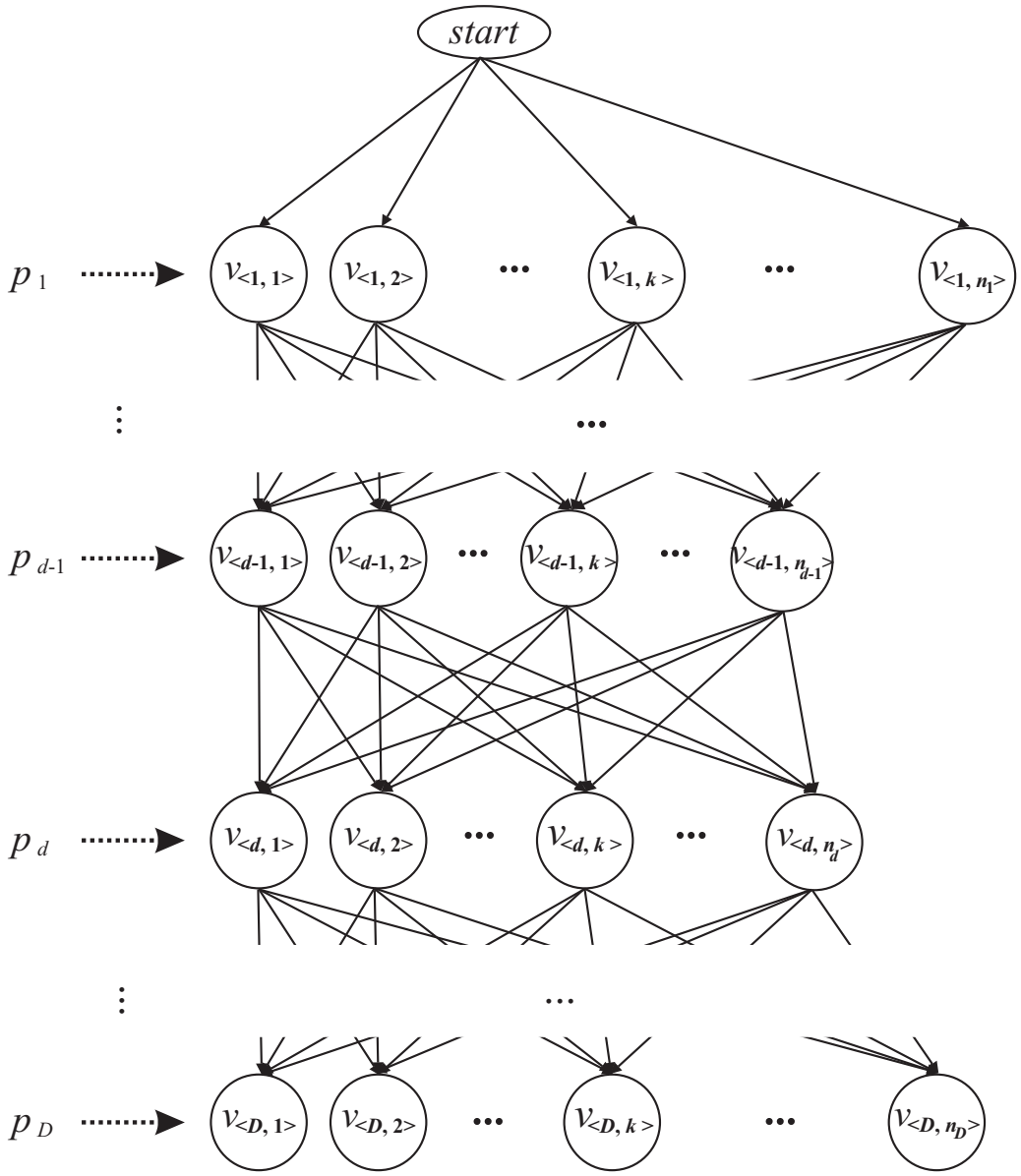

Figure 4.1 Search graph representation of a discrete multi-parameter optimization problem.

4.2 A multilevel paradigm

We consider the multilevel approach and its potential to aid the solution of optimization problems. As already described in Section 3.3, the multilevel approach is a simple one, which in its most basic form involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found (sometimes for the original problem, sometimes at the coarsest level) and then iteratively refined at each

level. As a general solution strategy the multilevel procedure has been in use for many years and has been applied to many problem areas.

However, with the exception of the graph-partitioning problem [55, 76], multilevel techniques in conjunction with ACO have not been widely applied to combinatorial optimization problems [125].

The multilevel approach consists of two main phases: coarsening and refinement. In our case we will concentrate on graph coarsening and refinement, but it can be used on any other structure.

4.2.1 Coarsening

Coarsening is done by merging two or more neighboring vertices into a single vertex (Figure 4.2); this is done in L iterations (we call them levels $\ell = 0, 1, \ldots, L$). Let us consider coarsening from level ℓ to level $\ell + 1$ at a distance d. Here

$$V_d^\ell = \left\{ v_{\langle d,1\rangle}^\ell, \ldots, v_{\langle d,n_d^\ell\rangle}^\ell \right\} \tag{4.5}$$

is a set of vertices at level ℓ and distance d of the search graph $\mathcal{G}$, where $1 \leq d \leq D$. If n_d^1 is the number of vertices at a starting level of coarsening and a distance d, then for every level ℓ the equation

$$n_d^{\ell+1} = \left\lceil \frac{n_d^\ell}{s_d^\ell} \right\rceil \tag{4.6}$$

is true, where s_d^ℓ is the number of vertices at level ℓ, which are merged into one vertex at level $\ell + 1$.

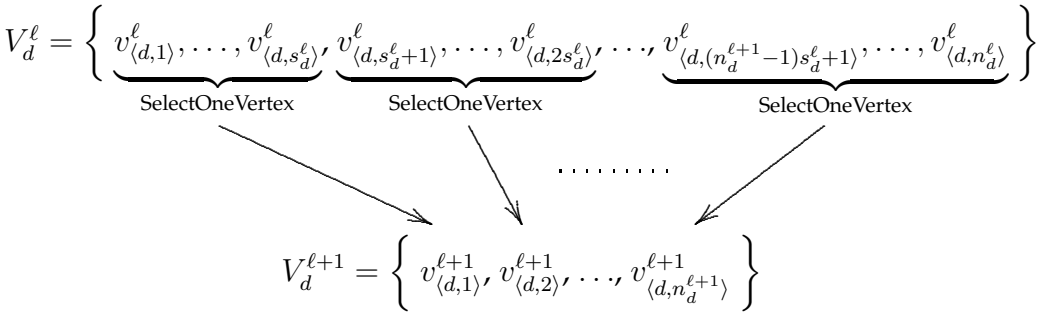

Figure 4.2 Coarsening of the search graph $\mathcal{G}$ from level ℓ to $\ell + 1$ at distance d (corresponding to values of the parameter p_d).

So what we do is we divide V_d^ℓ into $n_d^{\ell+1}$ subsets, where

$$V_d^\ell = \bigcup_{k=1}^{n_d^{\ell+1}} V_{\langle d,k\rangle}^\ell, \forall i,j \in \left\{1,\ldots,n_d^{\ell+1}\right\} \wedge i \neq j : V_{\langle d,i\rangle}^\ell \cap V_{\langle d,j\rangle}^\ell = \emptyset. \tag{4.7}$$

Each subset is defined as follows:

$$\begin{aligned} V_{\langle d,1\rangle}^\ell &= \left\{v_{\langle d,1\rangle}^\ell,\ldots,v_{\langle d,s_d^\ell\rangle}^\ell\right\}, \\ V_{\langle d,2\rangle}^\ell &= \left\{v_{\langle d,s_d^\ell+1\rangle}^\ell,\ldots,v_{\langle d,2s_d^\ell\rangle}^\ell\right\}, \\ &\vdots \\ V_{\langle d,n_d^{\ell+1}\rangle}^\ell &= \left\{v_{\langle d,(n_d^{\ell+1}-1)s_d^\ell+1\rangle}^\ell,\ldots,v_{\langle d,n_d^\ell\rangle}^\ell\right\}. \end{aligned} \tag{4.8}$$

Set $V_d^{\ell+1} = \left\{v_{\langle d,1\rangle}^{\ell+1},\ldots,v_{\langle d,n_d^{\ell+1}\rangle}^{\ell+1}\right\}$ is the set of vertices at distance d at level $\ell+1$, where $v_{\langle d,k\rangle}^{\ell+1} \in V_{\langle d,k\rangle}^\ell$ is selected on some predetermined principle. For example, random pick, the most left/right/centered vertex in the subset, etc.

The outline of the coarsening pseudo code from V_d^ℓ to $V_d^{\ell+1}$ is as follows:

Algorithm 4.1 Coarsening

1: **for** $k = 1$ to $n_d^{\ell+1}$ **do**
2: $v_{\langle d,k\rangle}^{\ell+1} =$ SelectOneVertex($V_{\langle d,k\rangle}^\ell$)
3: **end for**

4.2.2 Refinement

Because of the simplicity of the coarsening, the refinement itself is very trivial. Let us consider a refinement from level ℓ to level $\ell-1$ at distance d (Figure 4.3).

The outline of the refinement pseudo code is as follows:

Algorithm 4.2 Refinement

1: **for** $k = 1$ to n_d^ℓ **do**
2: **for** each $v_{\langle d,i\rangle}^{\ell-1} \in V_{\langle d,k\rangle}^{\ell-1}$ **do**
3: $v_{\langle d,i\rangle}^{\ell-1} =$ CopyVariables($v_{\langle d,k\rangle}^\ell$) // in our case $\tau_{\langle d,i\rangle}^{\ell-1} = \tau_{\langle d,k\rangle}^\ell$
4: **end for**
5: **end for**

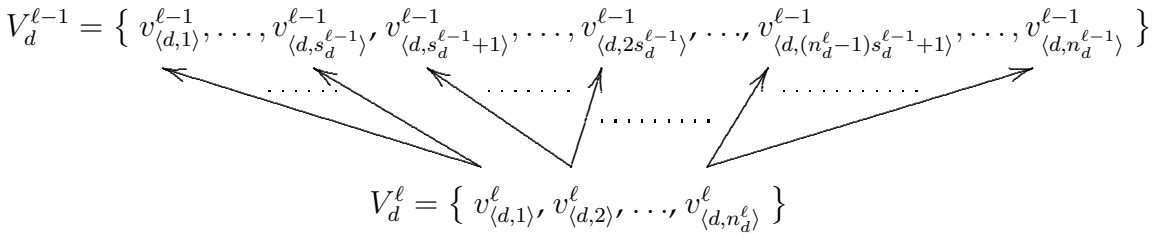

Figure 4.3 Refinement of the search graph $\mathcal{G}$ from level ℓ to $\ell - 1$ at distance d (corresponding to the values of parameter p_d).

Here the variable vertex attributes (in our case the amount of pheromone), as a result of the optimization at level ℓ, are transferred to level $\ell - 1$ with the use of the Copy-Variables function. Therefore, each vertex of subset $V^{\ell-1}_{\langle d,k\rangle}$ is assigned with the same value of variable attributes, which corresponds to the vertex $v^{\ell}_{\langle d,k\rangle}$ that was chosen in the coarsening from level $\ell - 1$ to level ℓ, while the constant vertex attributes stay the same.

4.2.3 The multilevel algorithm

Finally, the outline of the Multilevel Algorithm pseudo code could look like this:

Algorithm 4.3 Multilevel Algorithm

```
1: structure[0] = Initialization
2: for ℓ = 0 to L − 1 do
3:    structure[ℓ + 1] = Coarsening(structure[ℓ])
4: end for
5: for ℓ = L down to 0 do
6:    Solver(structure[ℓ])      // e.g., optimization algorithm
7:    if ℓ > 0 then
8:       structure[ℓ − 1] = Refinement(structure[ℓ])
9:    end if
10: end for
```

4.3 Ant-stigmergy optimization

The basic concept, as introduced at the beginning of this chapter, is as follows: first, we translate the multi-parameter problem into a search graph and then use an optimization technique to find the cheapest path in the constructed graph; this path consists of the values of the optimized parameters. In our case we use the Ant-Stigmergy Algorithm (ASA), the routes of which can be found in the ACO. The ASA consists of three main phases: initialization, optimization and local search.

Let us start with initialization. Here we translate the parameters of the problem into a search graph. This way we translate the multi-parameter problem into a problem of finding the cheapest path. Figure 4.1 shows how this is done.

Prior to the actual optimization an initial amount of pheromone, τ^0, is deposited uniformly in all the vertices in the search graph. There are a number of ants in a colony, all of which begin simultaneously from the $start$ vertex. The probability with which they choose the next vertex depends on the amount of pheromone in the vertices. Ants use a probability rule to determine which vertex will be chosen next. More specifically, ant α in step d moves from vertex $v_{\langle d-1,i\rangle} \in \left\{v_{\langle d-1,1\rangle}, \ldots, v_{\langle d-1,n_{d-1}\rangle}\right\}$ to vertex $v_{\langle d,j\rangle} \in \left\{v_{\langle d,1\rangle}, \ldots, v_{\langle d,n_d\rangle}\right\}$ with the probability given by:

$$\text{prob}_{ij,\alpha}(d) = \frac{\tau_{\langle d,j\rangle}}{\sum_{1\leq k\leq n_d} \tau_{\langle d,k\rangle}}, \tag{4.9}$$

where $\tau_{\langle d,k\rangle}$ is the amount of pheromone in vertex $v_{\langle d,k\rangle}$. The ants repeat this action until they reach the ending vertex. Then, the gathered parameter values of each ant (which can be found on its path) are evaluated. Next, each ant returns to the $start$ vertex and on its way deposits pheromone in the vertices according to the evaluation result: the better the result, the more pheromone is deposited in the vertices. After all the ants have returned to the start vertex, a so-called daemon action is made, which in this case consists of depositing some additional pheromone on what is currently the best path and also a smaller amount in neighboring vertices. Afterwards, pheromone evaporation from all the vertices occurs, i.e., the amount of pheromone is decreased by some predetermined percentage, ρ, in each vertex $v_{\langle d,k\rangle}$ in the search graph $\mathcal{G}$:

$$\tau^{\text{NEW}}_{\langle d,k\rangle} = (1-\rho)\tau^{\text{OLD}}_{\langle d,k\rangle}. \tag{4.10}$$

The whole procedure is then repeated until some ending condition is met (e.g., some predetermined number of iterations).

A local search has become a mandatory addition to any ant-based algorithm [37]. By using a local search it is usually possible to improve the convergence or improve the best solution, $\mathbf{p}^*$, found so far. We use it because our basic search technique is oriented more toward finding the best area of the solution space. We can say that the search is of a broader type, so a local search is used to improve the best solution. In our case a type of steepest-descent algorithm was used. The pseudo code of the Local Search (LS) can be seen in Algorithm 4.4.

Algorithm 4.4 Local Search

1: $\mathbf{p}^* = \{p_1, \ldots, p_i, \ldots, p_D\}$
2: change = True
3: **while** change **do**
4: change = False
5: **for** $i = 1$ to D **do**
6: **if** Evaluate($\{p_1, \ldots, p_i + p_i^{\text{step}}, \ldots, p_D\}$) < Evaluate($\mathbf{p}^*$) **then**
7: change = True
8: **while** Evaluate($\{p_1, \ldots, p_i + p_i^{\text{step}}, \ldots, p_D\}$) < Evaluate($\mathbf{p}^*$) **do**
9: $\mathbf{p}^* = \{p_1, \ldots, p_i + p_i^{\text{step}}, \ldots, p_D\}$
10: $p_i = p_i + p_i^{\text{step}}$
11: **end while**
12: **else if** Evaluate($\{p_1, \ldots, p_i - p_i^{\text{step}}, \ldots, p_D\}$) < Evaluate($\mathbf{p}^*$) **then**
13: change = True
14: **while** Evaluate($\{p_1, \ldots, p_i - p_i^{\text{step}}, \ldots, p_D\}$) < Evaluate($\mathbf{p}^*$) **do**
15: $\mathbf{p}^* = \{p_1, \ldots, p_i - p_i^{\text{step}}, \ldots, p_D\}$
16: $p_i = p_i - p_i^{\text{step}}$
17: **end while**
18: **end if**
19: **end for**
20: **end while**

Finally, the outline of the ASA pseudo code can be seen in Algorithm 4.5.

Algorithm 4.5 Ant-Stigmergy Algorithm

```
1:  searchGraph = Initialization(parameters)
2:  SearchGraphInitialization(initial pheromone amount)
3:  while not current level ending condition met do
4:     for all ants do
5:        path = FindPath(searchGraph)
6:        Evaluate(path)
7:     end for
8:     UpdatePheromone(all ants paths vertices)
9:     DaemonAction(best path)
10:    EvaporatePheromone(all vertices)
11: end while
12: LocalSearch(best solution)
```

When we ran the ASA on small search graphs the results were encouraging. But when we tried it on real problems or functions, which generate much larger graphs, it turned out that the convergence was slow and the results were very poor [60]. Therefore, we decided to apply a *multilevel approach* (see Figure 4.4).

4.4 The multilevel ant-stigmergy algorithm (MASA)

It is now time to merge the previously mentioned methods (Multilevel and the ASA) into a single algorithm. This approach is called the *Multilevel Ant-Stigmergy Algorithm* (MASA). The MASA consists of five main phases: initialization, coarsening, optimization, refinement, and local search. Each phase is exactly the same as described in Sections 4.2 and 4.3.

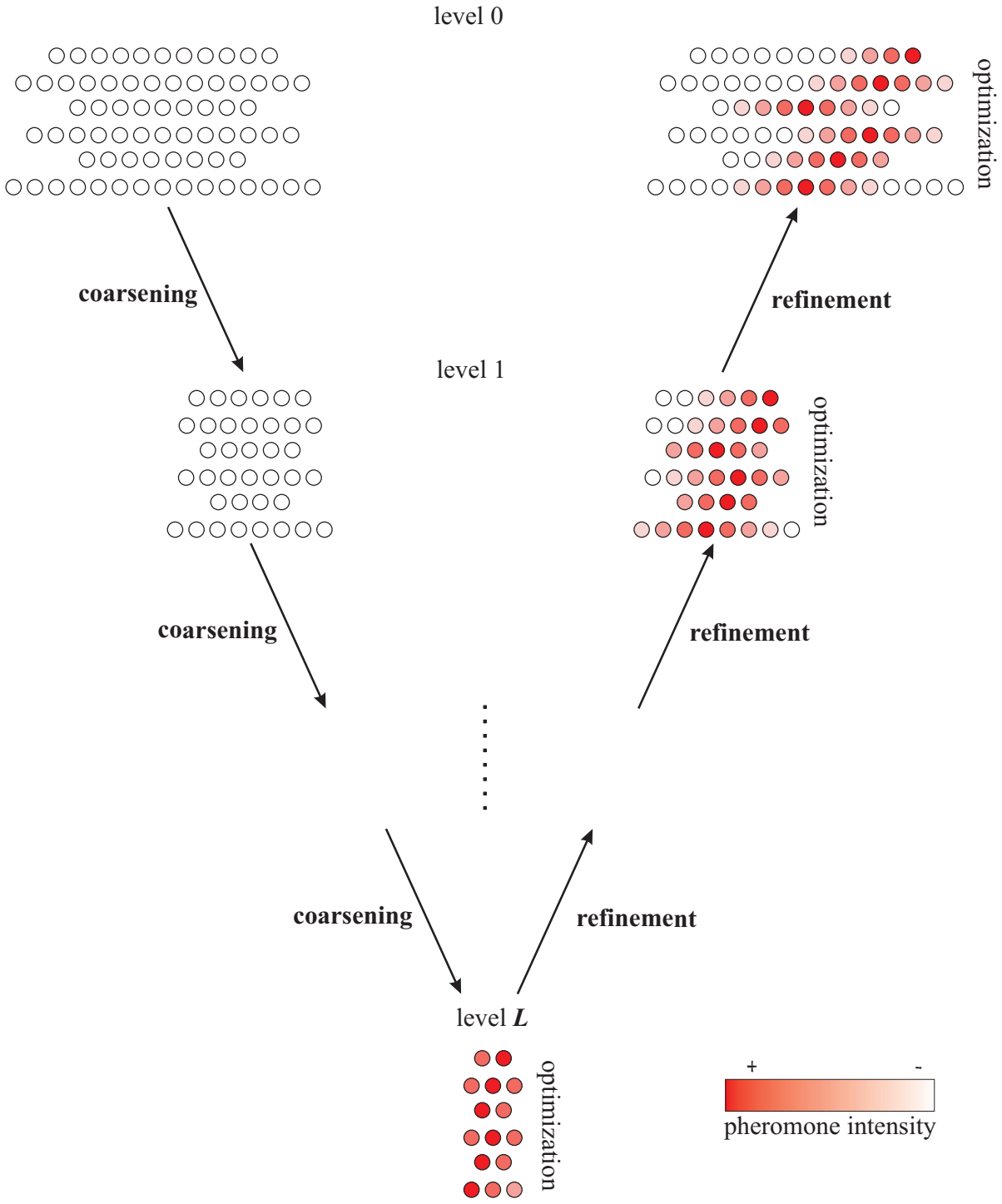

Figure 4.4 Multilevel approach on graph structure (the edges are omitted to make the figure clear).

The outline of the MASA pseudo code can be seen in Algorithm 4.6.

Algorithm 4.6 Multilevel Ant-Stigmergy Algorithm

```
1: searchGraph[0] = Initialization(parameters)
2: for ℓ = 0 to L − 1 do
3:    searchGraph[ℓ + 1] = Coarsening(searchGraph[ℓ])
4: end for
5: SearchGraphInitialization(initial pheromone amount)
6: for ℓ = L down to 0 do
7:    while not current level ending condition met do
8:       for all ants do
9:          path = FindPath(searchGraph[ℓ])
10:         Evaluate(path)
11:      end for
12:      UpdatePheromone(all ants paths vertices)
13:      DaemonAction(best path)
14:      EvaporatePheromone(all vertices)
15:   end while
16:   if ℓ > 0 then
17:      searchGraph[ℓ − 1] = Refinement(searchGraph[ℓ])
18:   end if
19: end for
20: LocalSearch(best solution)
```

4.4.1 Distributed implementation

Like many other metaheuristic approaches, the MASA allows direct parallelization schemes, and parallelism can be exploited on one or more levels [109]. In our implementation it was decided to choose the largest scale where entire search can be performed concurrently. Such an implementation, called parallel interacting ant colonies [96], is based on the well-known master/slave approach .

The non-distributed MASA approach presented in Section 4.4 was based on a single ant colony. But in the distributed approach this colony is split into N sub-colonies, where N is the number of processors. Every sub-colony is controlled by the server (see Algorithm 4.7).

Algorithm 4.7 Distributed Multilevel Ant-Stigmergy Algorithm - Server

```
1: StartAllClients
2: while not ending condition met do
3:    if received evaluated paths from client then
4:       BroadcastPaths(all other clients)
5:    end if
6: end while
7: solutions = ReceiveSolutions&StopAllClients
8: bestSolution = Best(solutions)
9: LocalSearch(bestSolution)
```

Each sub-colony searches for a solution according to the Algorithm 4.8. This algorithm is similar to the MASA (see Algorithm 4.6), except that at certain iterations an exchange of information between the sub-colonies occurs. With the use of the SendEvaluatedPathsToServer function the paths with an updated amount of pheromone are sent to the server, which then broadcasts this information to all the other clients (sub-colonies) with the use of the BroadcastPaths function. The updated amount of pheromone is determined by the Evaluate function. On the other hand, the information (paths with an updated amount of pheromone) is gathered from other sub-colonies using the ReceivePathsFromServer function. The last difference is in the UpdatePheromone function. Here, the pheromone is not just deposited on the found paths but also on the received ones.

4.4.2 Grid implementation

The distributed MASA was designed to work on any kind of Grid that uses the TCP/IP protocol for communication. One can see that the distributed MASA consists of a parallel part (runs on clients) and a sequential part (local search – runs on server). This

Algorithm 4.8 Distributed Multilevel Ant-Stigmergy Algorithm - Client

```
1: searchGraph[0] = GraphConstruction(parameters)
2: for ℓ = 0 to L − 1 do
3:    searchGraph[ℓ + 1] = Coarsening(searchGraph[ℓ])
4: end for
5: SearchGraphInitialization(initial pheromone amount)
6: for ℓ = L down to 0 do
7:    while not current level ending condition met do
8:       for all ants in sub-colony do
9:          path = FindPath(searchGraph)
10:         Evaluate(path)
11:      end for
12:      SendEvaluatedPathsToServer(all ants)
13:      ReceivePathsFromServer(from all other clients)
14:      UpdatePheromone(all found and received paths vertices)
15:      DaemonAction(best path)
16:      EvaporatePheromone(all vertices)
17:   end while
18:   if ℓ > 0 then
19:      searchGraph[ℓ − 1] = Refinement(searchGraph[ℓ])
20:   end if
21: end for
```

would mean that we can speed up the search process only in the parallel part, but the sequential part stays constant. But we can go further than that by using the so-called software pipeline. Our pipeline consists of two stages. The first stage is a parallelized part of the algorithm, and the second stage is local search. So while the server is running the local search the clients can already search for new solutions. Therefore, we can acquire new solutions as fast as the local search is performed. Of course we can also have multiple servers, which would further decrease the time needed to acquire new solutions.

4.5 Performance evaluation

4.5.1 The experimental environment

The computer platform used to perform the experiments was an AMD Opteron™ 2.6-GHz processor, 2 GB of RAM, and the Microsoft® Windows® XP operating system.

The experiments were done on non-noisy and noisy benchmark functions (see Subsection 4.5.2). We ran the MASA 30 times in each experiment. The number of maximum function evaluations per experiment N was set to 500,000. The number of ants was 10, $\rho = 0.1$, and the coarsening was implemented by merging two vertices into one (which defines the number of levels L). With regard to the ending condition for each level, we have two different policies. In the case of non-noisy functions, the ending condition was set to "no best solution found for the last 50 iterations", while in the case of noisy functions we limited the number of evaluations per level to $\frac{N}{L}$. In this way we ensure that the algorithm does not stay too long on coarse-grained graphs, i.e., levels with high ℓ. We must note that during the experimentation we did not fine-tune the algorithms parameters, but only make a limited number of experiments to find satisfying settings.

4.5.2 The benchmark suite

For the benchmark functions we have decided to use some widely used functions. For evaluation purposes we used three different function dimensions $D = |\mathbf{p}| = 5, 25,$ and 50. The function definitions are as follows:

Sphere f_{S}:

$$f_S(\mathbf{p}) = \sum_{i=1}^{D} p_i^2. \tag{4.11}$$

Griewangk f_{G}:

$$f_G(\mathbf{p}) = \frac{1}{4{,}000} \sum_{i=1}^{D} (p_i - 100)^2 - \prod_{i=1}^{D} \cos\left(\frac{p_i - 100}{\sqrt{i}}\right) + 1. \tag{4.12}$$

Rastrigin F1 $f_{\mathrm{R\_F1}}$:

$$f_{R\_F1}(\mathbf{p}) = \sum_{i=1}^{D} \left(10 + p_i^2 - 10\cos(2\pi p_i)\right). \tag{4.13}$$

Rosenbrock f_{R}:

$$f_R(\mathbf{p}) = \sum_{i=1}^{D-1} \left(100\left(p_{i+1} - p_i^2\right)^2 + \left(p_i - 1\right)^2\right). \tag{4.14}$$

Krink F2 $f_{\mathrm{K\_F2}}$:

$$f_{K\_F2}(\mathbf{p}) = \sum_{i=1}^{D} \left(37.816415 + |p_i - 50| - 40\sin\left(\frac{5\pi p_i}{18}\right)\right). \tag{4.15}$$

Krink F3 $f_{\mathrm{K\_F3}}$:

$$f_{nK\_F3}(\mathbf{p}) = \sum_{i=1}^{D} \left(89.016293 - |p_i - 50| + 40\sin\left(\frac{5\pi p_i}{18}\right)\right). \tag{4.16}$$

Krink functions are derivatives of the *Krink F1* function [80]:

$$f_{K\_F1}(\mathbf{p}) = \sum_{i=1}^{D} \left(50 + |p_i - 50| - 40\sin\left(\frac{5\pi p_i}{18}\right)\right) \tag{4.17}$$

where the maximum is sought. The $f_{\mathrm{K\_F2}}$ is negated and shifted $f_{\mathrm{K\_F1}}$, while $f_{\mathrm{K\_F3}}$ is only shifted $f_{\mathrm{K\_F1}}$. With negating the optimum stays in the same point, while with shifting the global minimum was set at approximately zero.

The optimization of noisy functions is a common task occurring in various applications. In some applications the function to be minimized is only known to a low precision. For the purpose of simulating this problem we introduce noisy versions of the benchmark functions that are defined as

$$\tilde{f}(\mathbf{p}, s) = \frac{1}{s}\sum_{i=1}^{s} \left(f(\mathbf{p}) + \mathrm{Gauss}(0, 1)\right), \tag{4.18}$$

where s is the number of samples (evaluations with added noise) needed to compute a noisy function, and $\mathrm{Gauss}(0, 1)$ is a Gaussian distribution with a mean of 0 and a standard deviation of 1. For evaluation purposes we used three different degrees of sampling, $s = 10, 50,$ and 100.

The problem of dealing with noisy functions has been addressed by various researchers, mainly for ES [97], Evolution Programming (EP) [34], GA [51], PSO [57], and DE [79].

4.5.3 Compared algorithms

The GA, PSO, and DE are very popular numerical optimization procedures. The results reported in [79, 121] show that the DE generally outperforms the other algorithms. Therefore, we decided to compare the MASA with the DE.

The DE is a stochastic, population-based optimization algorithm. It was introduced by Storn and Price [106] and was developed to optimize the real (float) parameters of a real-valued function. DE resembles the structure of an EA, but differs from traditional EAs in its generation of new candidate solutions and by its use of a "greedy" selection scheme. The basic idea of DE is outlined in Algorithm 4.9.

Algorithm 4.9 Differential Evolution

```
1: population = RandomCreate(parameters)
2: Evaluate(population)
3: while not ending condition met do
4:    for each parent from population do
5:       candidates[1..3] = RandomSelect(population)
6:       newCandidate = Calculate(candidates[1..3])
7:       BinomialCrossover(parent, newCandidate)
8:       Evaluate(newCandidate)
9:       if newCandidate better than parent then
10:         parent = newCandidate
11:      end if
12:   end for
13:   RandomEnumerate(population)
14: end while
```

The newCandidate is calculated as a weighted sum of three randomly chosen candidates that are different from the parent. Only then does the parent participate in the creation of the candidate—the candidate is modified by a crossover with its parent. Finally, the candidate is evaluated and compared to the parent. The candidate replaces the parent in the population, only if it is better than the parent. The described proce-

dure (body of the for loop in the Algorithm 4.9) is repeated for all the parent individuals from the population. When it is finished the individuals from the population are randomly enumerated and the procedure is repeated.

Recall that the DE is used on continuous problems, while the MASA works on discrete problems.

The DE has three parameters, which were set to the following values, as proposed by Krink et al. [79]: the population size was 50, the crossover constant was 0.8, and the scaling factor was 0.5. We ran the DE on each test function 30 times. The maximum number of function evaluations per experiment was set to $500{,}000$.

4.5.4 The complexity of the algorithm

The algorithm's complexity is estimated as suggested in [114] by calculating the following formula $\frac{\widehat{T}_2 - T_1}{T_0}$, where computing time T_0 is independent of the function dimension and is calculated by running the benchmark algorithm described in Algorithm 4.10, T_1 is the computing time for $200{,}000$ evaluations just for function *Rosenbrock* f_{R} (for function definition see Eq. 4.14 in Subsection 4.5.2) and $\widehat{T}_2$ is the mean time of five executions, but now considering the complete computing time of the algorithm for function f_{R}.

Algorithm 4.10 Benchmark

1: **for** $i = 1$ to $1{,}000{,}000$ **do**
2: x = 5.55
3: x = x + x
4: x = x * x
5: x = sqrt(x)
6: x = ln(x)
7: x = exp(x)
8: y = x/x
9: **end for**

The results presented in Table 4.1 show that the MASA has a higher complexity than the DE (for the algorithm definition see Subsection 4.5.3), but when dealing with real-

world problems this deficiency becomes insignificant compared to the time needed to compute a single evaluation of an objective function [64].

Table 4.1 Algorithm complexity (function f_{R}, $D = 50$).

Algorithm	T_0 [s]	T_1 [s]	$\widehat{T}_2$ [s]	$\frac{\widehat{T}_2 - T_1}{T_0}$
MASA	0.2	3.0	44.0	205
DE	0.2	3.0	6.0	15

4.5.5 An evaluation

The global minimum of the test functions f_{S}, f_{G}, $f_{\mathrm{R\_F1}}$, and f_{R} is exactly zero, while for $f_{\mathrm{K\_F2}}$ and $f_{\mathrm{K\_F3}}$ we set the constants so that the global minimum is as close to zero possible, see Table 4.2.

Table 4.2 Function constraints and optimal values.

Function	Lower bound L_i	Upper bound U_i	Step Δ_i	Minimum value
$f_{\mathrm{S}}(\mathbf{p})$	-100	100	10^{-3}	$f_{\mathrm{S}}(\mathbf{0}) = 0$
$f_{\mathrm{G}}(\mathbf{p})$	-600	600	10^{-2}	$f_{\mathrm{G}}(\mathbf{100}) = 0$
$f_{\mathrm{R\_F1}}(\mathbf{p})$	-5.12	5.12	10^{-4}	$f_{\mathrm{R\_F1}}(\mathbf{0}) = 0$
$f_{\mathrm{R}}(\mathbf{p})$	-50	50	10^{-3}	$f_{\mathrm{R}}(\mathbf{1}) = 0$
$f_{\mathrm{K\_F2}}(\mathbf{p})$	0	100	10^{-3}	$f_{\mathrm{K\_F2}}(\approx \mathbf{52.167}) \approx 0$
$f_{\mathrm{K\_F3}}(\mathbf{p})$	0	100	10^{-3}	$f_{\mathrm{K\_F3}}(\approx \mathbf{99.031}) \approx 0$

The evaluation results of the MASA with and without LS on non-noisy functions are presented in Table 4.3, where the best and the average solutions obtained from 30 runs are shown for each experiment. The standard deviation of the solutions and the average number of function evaluations per experiment are also included.

Table 4.3 Experimental results of the MASA on non-noisy functions.

	Function	Best	Mean	Std	Avg iter
w	$D = 5$				
i	f_{S}	0	0	0	9,693
t	f_{G}	0	$0.616 \cdot 10^{-1}$	$0.598 \cdot 10^{-1}$	11,337
h	$f_{\mathrm{R\_F1}}$	0	0	0	8,875
o	f_{R}	$0.133 \cdot 10^{-1}$	$0.282 \cdot 10^{-1}$	$0.104 \cdot 10^{-1}$	80,227
u	$f_{\mathrm{K\_F2}}$	$0.136 \cdot 10^{-5}$	4.733	3.514	15,741
t	$f_{\mathrm{K\_F3}}$	$-0.406 \cdot 10^{-3}$	5.613	5.334	21,610
	$D = 25$				
l	f_{S}	$0.500 \cdot 10^{-5}$	$0.920 \cdot 10^{-5}$	$0.202 \cdot 10^{-5}$	22,747
o	f_{G}	$0.296 \cdot 10^{-4}$	$0.148 \cdot 10^{-1}$	$0.140 \cdot 10^{-1}$	30,653
c	$f_{\mathrm{R\_F1}}$	$0.119 \cdot 10^{-4}$	0.696	0.911	31,979
a	f_{R}	9.712	31.340	36.310	466,687
l	$f_{\mathrm{K\_F2}}$	$0.980 \cdot 10^{-4}$	3.547	3.955	58,965
	$f_{\mathrm{K\_F3}}$	$-0.159 \cdot 10^{-2}$	4.691	5.996	56,518
	$D = 50$				
s	f_{S}	$0.340 \cdot 10^{-2}$	$0.460 \cdot 10^{-2}$	$0.839 \cdot 10^{-3}$	27,343
e	f_{G}	$0.142 \cdot 10^{-3}$	$0.356 \cdot 10^{-2}$	$0.610 \cdot 10^{-2}$	46,228
a	$f_{\mathrm{R\_F1}}$	$0.714 \cdot 10^{-4}$	0.663	1.149	55,601
r	f_{R}	38.844	80.789	46.993	348,816
c	$f_{\mathrm{K\_F2}}$	$0.480 \cdot 10^{-3}$	3.828	4.502	87,853
h	$f_{\mathrm{K\_F3}}$	$-0.155 \cdot 10^{-2}$	3.224	4.464	86,534
	$D = 5$				
	f_{S}	0	0	0	9,703
w	f_{G}	0	$0.616 \cdot 10^{-1}$	$0.598 \cdot 10^{-1}$	11,347
i	$f_{\mathrm{R\_F1}}$	0	0	0	8,885
t	f_{R}	$0.133 \cdot 10^{-1}$	$0.280 \cdot 10^{-1}$	$0.102 \cdot 10^{-1}$	80,246
h	$f_{\mathrm{K\_F2}}$	$0.136 \cdot 10^{-5}$	4.733	3.514	15,751
	$f_{\mathrm{K\_F3}}$	$-0.609 \cdot 10^{-3}$	5.613	5.334	21,626
	$D = 25$				
l	f_{S}	0	0	0	22,852
o	f_{G}	0	$0.148 \cdot 10^{-1}$	$0.140 \cdot 10^{-1}$	30,761
c	$f_{\mathrm{R\_F1}}$	0	0.696	0.911	32,084
a	f_{R}	$0.174 \cdot 10^{-1}$	0.949	2.636	500,000
l	$f_{\mathrm{K\_F2}}$	$0.681 \cdot 10^{-5}$	3.547	3.955	59,069
	$f_{\mathrm{K\_F3}}$	$-0.304 \cdot 10^{-2}$	4.690	5.997	56,639
	$D = 50$				
s	f_{S}	0	0	0	27,562
e	f_{G}	0	$0.328 \cdot 10^{-2}$	$0.608 \cdot 10^{-2}$	46,472
a	$f_{\mathrm{R\_F1}}$	0	0.663	1.149	55,824
r	f_{R}	$0.744 \cdot 10^{-1}$	5.126	18.595	500,000
c	$f_{\mathrm{K\_F2}}$	$0.136 \cdot 10^{-4}$	3.827	4.502	88,073
h	$f_{\mathrm{K\_F3}}$	$-0.609 \cdot 10^{-2}$	3.221	4.465	86,784

In Table 4.3 we can see that on almost all functions and dimensions the MASA found an optimal or near-optimal solution. The only function where it performed worse was f_{R}. The main reason for this is that the first expression $100\,(p_{i+1} - p_i^2)^2$ has two global minima at $p_i = 0$ and $p_i = 1$, $i = 1, 2, \ldots, D$, while the second expression $(p_i - 1)^2$ has only one global minimum at $p_i = 1$. The p_i^2 in the first expression prefers $p_i = 0$ over $p_i = 1$. Since the first expression dominates over the second, the MASA is at first misled into solution $p_i = 0$, from where it can slowly move toward the global minimum at $p_i = 1$, $i = 1, 2, \ldots, D$.

Table 4.4 Experimental results of the MASA without local search on noisy functions with $D = 50$.

Function	Best	Mean	Std	Avg iter
$s = 10$				
$\tilde{f}_S$	-0.433	0.261	0.314	500,000
$\tilde{f}_G$	0.791	1.646	0.527	500,000
$\tilde{f}_{R\_F1}$	6.113	10.589	2.511	500,000
$\tilde{f}_R$	46.428	527.665	825.313	500,000
$\tilde{f}_{K\_F2}$	78.918	119.566	21.214	500,000
$\tilde{f}_{K\_F3}$	132.751	164.080	22.010	500,000
$s = 50$				
$\tilde{f}_S$	76.441	212.538	91.326	500,000
$\tilde{f}_G$	2.946	5.131	1.235	500,000
$\tilde{f}_{R\_F1}$	22.213	74.578	19.105	500,000
$\tilde{f}_R$	3,051.539	57,664.280	43,795.415	500,000
$\tilde{f}_{K\_F2}$	352.753	425.406	26.116	500,000
$\tilde{f}_{K\_F3}$	667.893	764.757	54.007	500,000
$s = 100$				
$\tilde{f}_S$	1,062.354	2,337.157	761.205	500,000
$\tilde{f}_G$	17.040	32.121	8.081	500,000
$\tilde{f}_{R\_F1}$	114.638	167.781	20.200	500,000
$\tilde{f}_R$	1,270,827	3,740,176	2,279,957	500,000
$\tilde{f}_{K\_F2}$	502.813	671.239	68.903	500,000
$\tilde{f}_{K\_F3}$	866.699	1,070.813	80.812	500,000

To evaluate the performance of the MASA on noisy functions we decided to test it on functions with dimension $D = 50$. Table 4.4 shows the results of the MASA without

LS on the noisy functions. One can see that with the increase of degree of sampling, s, the results deteriorate noticeably.

Table 4.5 Experimental results of the DE on non-noisy functions.

Function	Best	Mean	Std	Avg iter
$D = 5$				
f_{S}	0	0	0	8,785
f_{G}	0	0	0	39,697
$f_{\mathrm{R\_F1}}$	0	0	0	18,222
f_{R}	0	$0.315 \cdot 10^{-7}$	$0.131 \cdot 10^{-6}$	84,646
$f_{\mathrm{K\_F2}}$	$0.742 \cdot 10^{-4}$	$0.742 \cdot 10^{-4}$	0	500,000
$f_{\mathrm{K\_F3}}$	0.418	8.100	8.466	500,000
$D = 25$				
f_{S}	0	0	0	52,230
f_{G}	0	$0.986 \cdot 10^{-3}$	$0.308 \cdot 10^{-2}$	100,140
$f_{\mathrm{R\_F1}}$	0.995	14.307	14.083	500,000
f_{R}	0	$0.139 \cdot 10^{-1}$	$0.745 \cdot 10^{-1}$	476,097
$f_{\mathrm{K\_F2}}$	14.000	209.900	86.522	500,000
$f_{\mathrm{K\_F3}}$	19.795	87.365	39.688	500,000
$D = 50$				
f_{S}	0	0	0	105,560
f_{G}	0	$0.493 \cdot 10^{-3}$	$0.188 \cdot 10^{-2}$	132,061
$f_{\mathrm{R\_F1}}$	11.940	98.290	43.517	500,000
f_{R}	15.188	37.273	14.522	500,000
$f_{\mathrm{K\_F2}}$	600.382	921.951	156.130	500,000
$f_{\mathrm{K\_F3}}$	97.104	228.734	65.940	500,000

In Table 4.5 we can see that for f_{S} and f_{G} the DE returns near optimal results for all dimensions; for $f_{\mathrm{R\_F1}}$ and $f_{\mathrm{K\_F2}}$ it returns near optimal results only for $D = 5$; for f_{R} it returns near optimal results for $D < 50$; for $f_{\mathrm{K\_F3}}$ the DE does not produce any good results in $500{,}000$ evaluations.

Table 4.6 shows the results of the DE on noisy functions. Like in the case of the MASA, we tested the DE on functions with the dimension $D = 50$. Here we also notice a deterioration of the results. The impact of the higher degree of sampling, $s = 50$ and $s = 100$, on the performance of the DE is greater than for the MASA.

A convergence comparison of the DE and the MASA on non-noisy functions can be seen in Figures 4.5–4.7, where the graphs show the mean performance of 30 runs for

each function.

Table 4.6 Experimental results of the DE on noisy functions with $D = 50$.

Function	Best	Mean	Std	Avg iter
$s = 10$				
$\tilde{f}_S$	-0.656	-0.393	0.117	$500{,}000$
$\tilde{f}_G$	$0.387 \cdot 10^{-1}$	0.436	0.130	$500{,}000$
$\tilde{f}_{R\_F1}$	338.419	372.889	16.961	$500{,}000$
$\tilde{f}_R$	40.450	66.248	32.114	$500{,}000$
$\tilde{f}_{K\_F2}$	$1{,}549.370$	$1{,}700.032$	72.681	$500{,}000$
$\tilde{f}_{K\_F3}$	106.612	252.194	92.002	$500{,}000$
$s = 50$				
$\tilde{f}_S$	807.386	$1{,}358.840$	463.151	$500{,}000$
$\tilde{f}_G$	9.055	13.071	3.079	$500{,}000$
$\tilde{f}_{R\_F1}$	409.340	453.605	21.390	$500{,}000$
$\tilde{f}_R$	$1{,}708{,}481$	$4{,}523{,}049$	$2{,}358{,}303$	$500{,}000$
$\tilde{f}_{K\_F2}$	$1{,}715.753$	$1{,}963.043$	81.910	$500{,}000$
$\tilde{f}_{K\_F3}$	634.912	910.486	121.962	$500{,}000$
$s = 100$				
$\tilde{f}_S$	$7{,}057.189$	$12{,}035.640$	$2{,}571.304$	$500{,}000$
$\tilde{f}_G$	62.169	108.015	21.525	$500{,}000$
$\tilde{f}_{R\_F1}$	439.637	500.555	24.734	$500{,}000$
$\tilde{f}_R$	$39{,}842{,}415$	$122{,}003{,}535$	$57{,}879{,}863$	$500{,}000$
$\tilde{f}_{K\_F2}$	$1{,}977.308$	$2{,}108.498$	73.489	$500{,}000$
$\tilde{f}_{K\_F3}$	954.527	$1{,}165.232$	95.198	$500{,}000$

Figure 4.5 shows the functions with $D = 5$. Here we do not notice any large performance differences between the algorithms. The algorithms' average returned results and convergence are approximately the same. For higher dimensions, $D = 25$ (Figure 4.6) and $D = 50$ (Figure 4.7), we observe, with the exception of f_{R}, that the MASA outperforms the DE.

The convergence comparison of the DE and the MASA on noisy functions with $D = 50$ can be seen in Figures 4.8–4.10. For a higher degree of sampling, $s = 50$ (Figure 4.9) and $s = 100$ (Figure 4.10), we observe that the MASA outperforms the DE. With the MASA one can see a cascading approach toward the optimal solution. The reason for this is the constant number of function evaluations at each level.

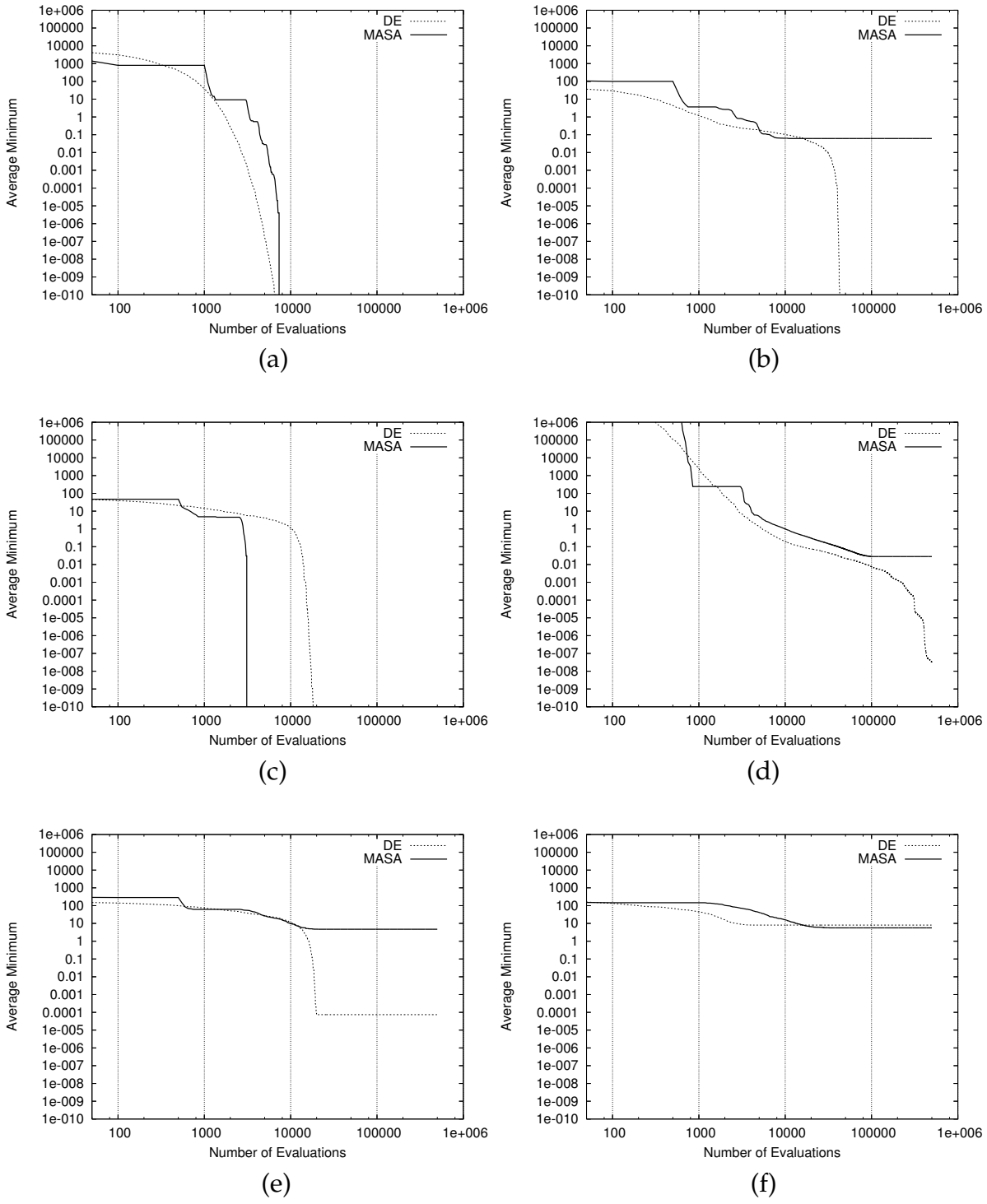

Figure 4.5 Convergence graphs for (a) *Sphere*, (b) *Griewangk*, (c) *Rastrigin F1*, (d) *Rosenbrock*, (e) *Krink F2*, and (f) *Krink F3* functions with $D = 5$.

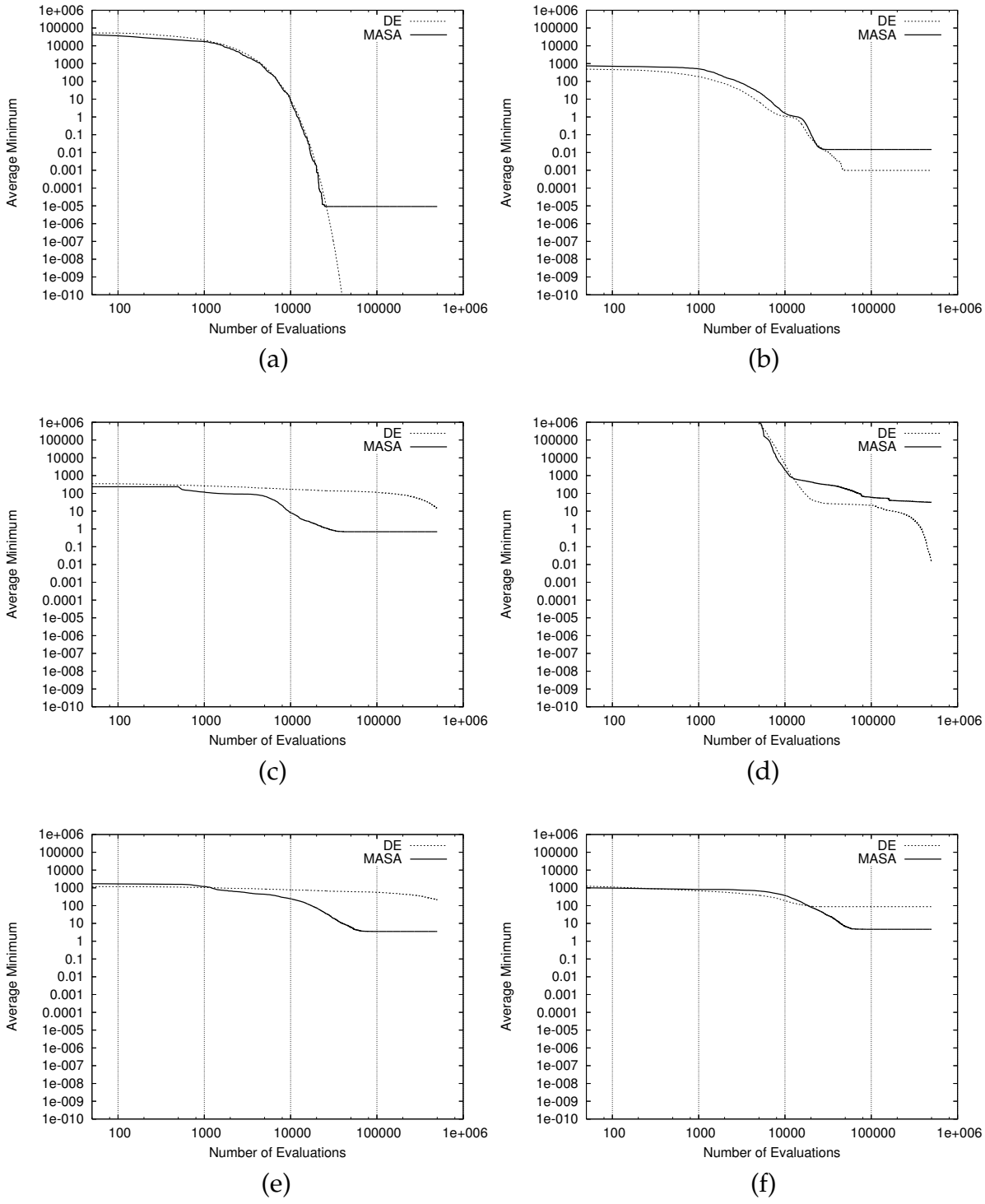

Figure 4.6 Convergence graphs for (a) *Sphere*, (b) *Griewangk*, (c) *Rastrigin F1*, (d) *Rosenbrock*, (e) *Krink F2*, and (f) *Krink F3* functions with $D = 25$.

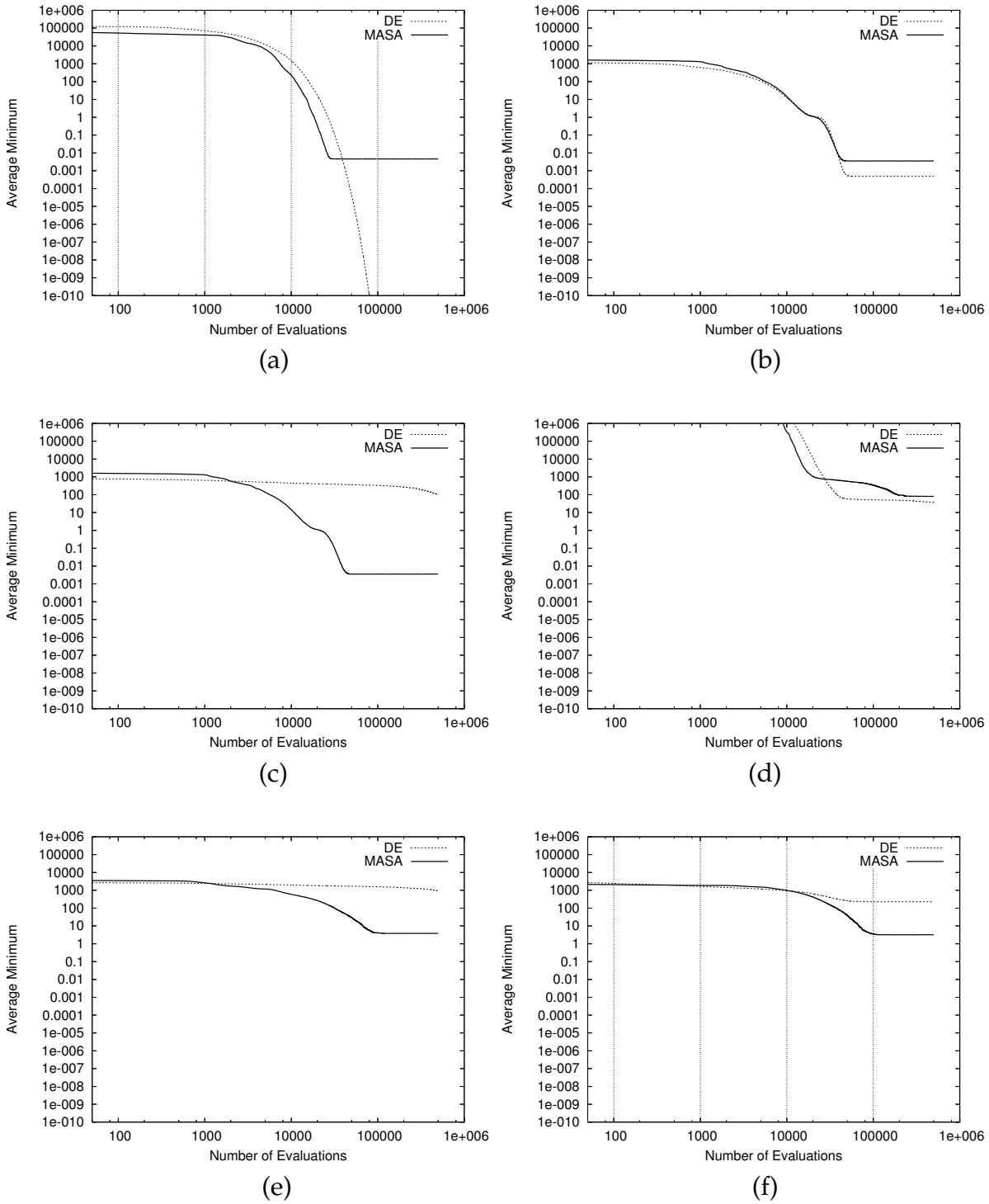

Figure 4.7 Convergence graphs for (a) *Sphere*, (b) *Griewangk*, (c) *Rastrigin F1*, (d) *Rosenbrock*, (e) *Krink F2*, and (f) *Krink F3* functions with $D = 50$.

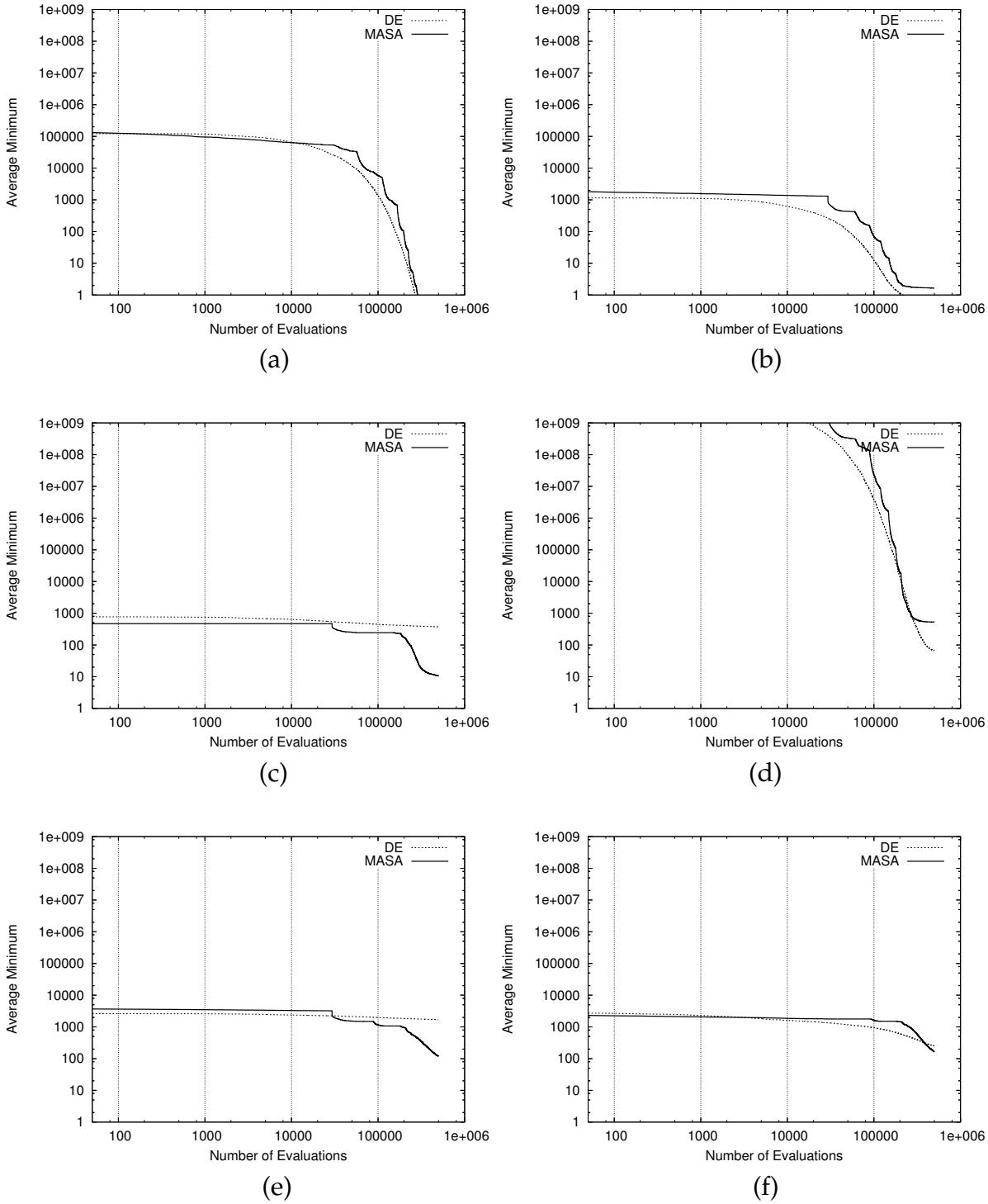

Figure 4.8 Convergence graphs for (a) *Sphere*, (b) *Griewangk*, (c) *Rastrigin F1*, (d) *Rosenbrock*, (e) *Krink F2*, and (f) *Krink F3* noisy functions with $s = 10$ and $D = 50$.

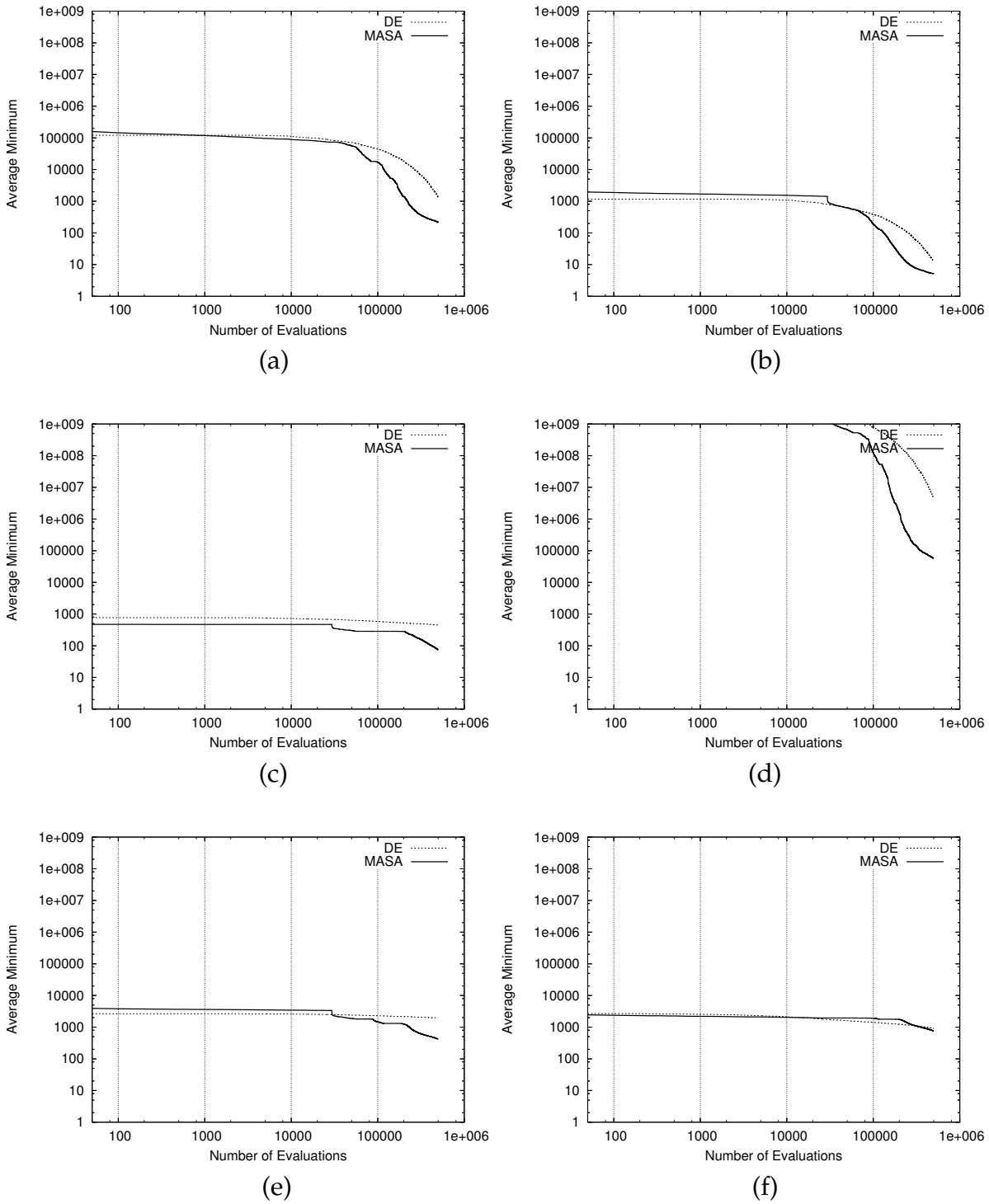

Figure 4.9 Convergence graphs for (a) *Sphere*, (b) *Griewangk*, (c) *Rastrigin F1*, (d) *Rosenbrock*, (e) *Krink F2*, and (f) *Krink F3* noisy functions with $s = 50$ and $D = 50$.

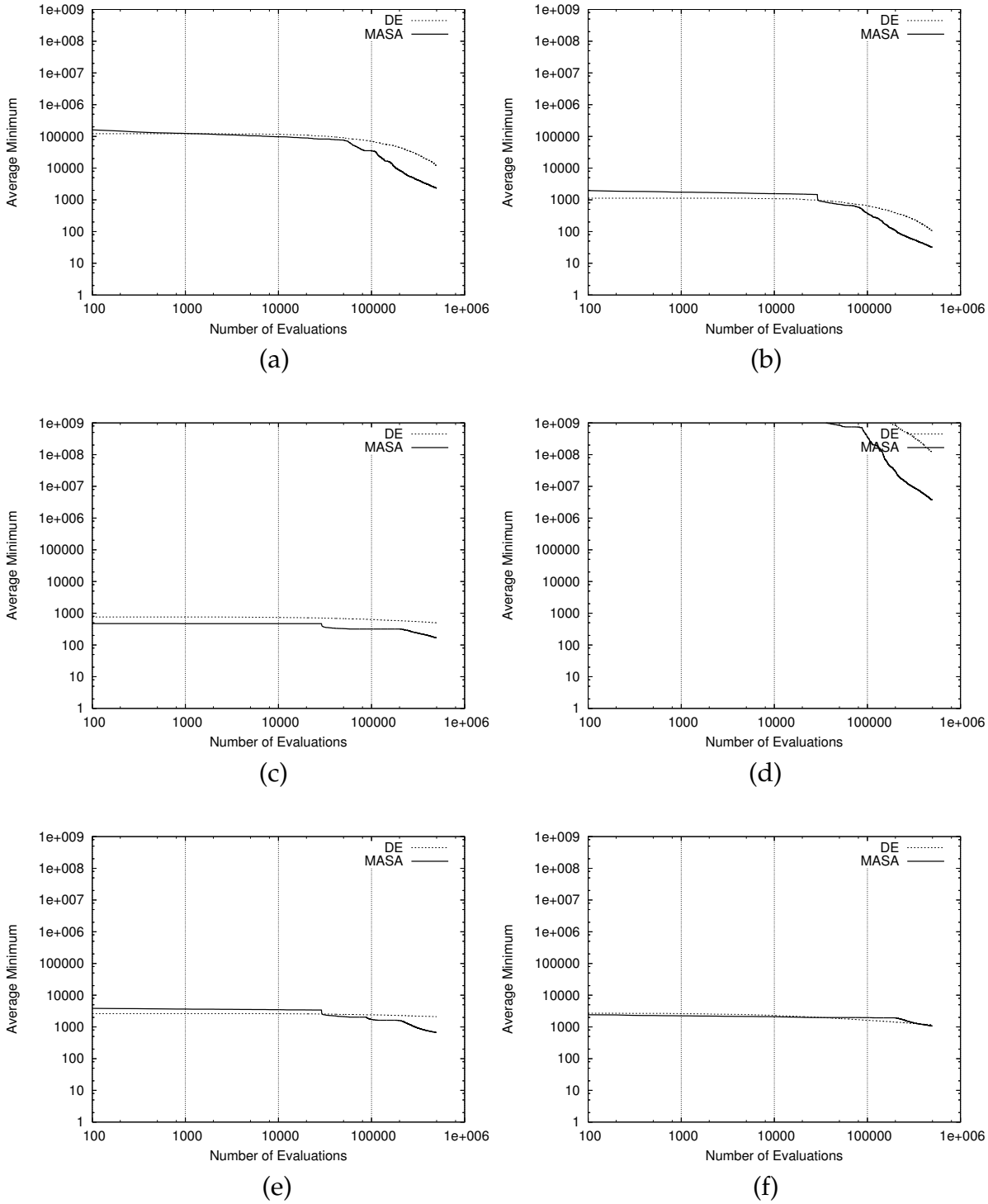

Figure 4.10 Convergence graphs for (a) *Sphere*, (b) *Griewangk*, (c) *Rastrigin F1*, (d) *Rosenbrock*, (e) *Krink F2*, and (f) *Krink F3* noisy functions with $s = 100$ and $D = 50$.

If we look at all these figures carefully we can see that for low-dimensional functions the performance is comparable, while for higher dimensions the MASA outperformed the DE for all but one of the functions.

Regarding the noisy functions we can observe that the impact of the higher degree of sampling on the performance of the DE is greater than for the MASA.

Even though we have seen that a discrete version of numerical optimization problems can be successfully solved with the MASA, there might be some cases where putting real parameters into discrete form is not possible. For these cases we developed a new algorithm, which is described in the next chapter. It will be adjusted to the numerical optimization problem and not vice versa, as was the case with the MASA.

5 The differential ant-stigmergy approach

In contrast to Chapter 4 we will here introduce a new approach to the continuous multi-parameter optimization problem using a newly developed ACO-based algorithm [68].

5.1 Problem representation

5.1.1 The fine-grained discrete form of continuous domain

In the following, a process of transformation from a continuous domain into a fine-grained discrete form is presented.

Let p_i' be the current value of the i-th parameter. During the search for the optimal parameter value, the new value is assigned to the i-th parameter as follows:

$$p_i = p_i' + \delta_i. \tag{5.1}$$

Here, δ_i is the so-called *parameter difference* and is chosen from the set

$$\Delta_i = \Delta_i^- \cup \{0\} \cup \Delta_i^+, \tag{5.2}$$

where

$$\Delta_i^- = \left\{\delta_{i,k}^-|\ \delta_{i,k}^- = -b^{k+L_i-1}, k = 1, 2, \ldots, d_i\right\} \tag{5.3}$$

and

$$\Delta_i^+ = \left\{\delta_{i,k}^+|\ \delta_{i,k}^+ = b^{k+L_i-1}, k = 1, 2, \ldots, d_i\right\}. \tag{5.4}$$

Here

$$d_i = U_i - L_i + 1. \tag{5.5}$$

Therefore, for each parameter p_i, the parameter difference, δ_i, has a range from b^{L_i} to b^{U_i}, where b is the so-called *discrete base*,

$$L_i = \lfloor \log_b(\varepsilon_i) \rfloor, \tag{5.6}$$

and

$$U_i = \lfloor \log_b \left(\max(p_i) - \min(p_i) \right) \rfloor. \tag{5.7}$$

With the parameter ε_i, the maximum precision of the parameter p_i is set. The precision is limited by the computer's floating-point arithmetics.

Let us consider a simple example for the parameter p_i with $\max(p_i) = 400$, $\min(p_i) = -350$, $b = 10$, and $\varepsilon_i = 10^{-3}$. Then $L_i = -3$, $U_i = 2$, and $d_i = 6$. Finally, Δ_i is constructed as follows:

$$\left\{ -10^2, -10^1, -10^0, -10^{-1}, -10^{-2}, -10^{-3}, 0, 10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2 \right\}.$$

5.1.2 Graph representation

From all the sets Δ_i, $1 \leq i \leq D$, where D represents the number of parameters, a so-called *differential graph* $\mathcal{G} = (V, E)$ with a set of vertices, V, and a set of edges, E, between the vertices is constructed. Each set Δ_i is represented by the set of vertices, $V_i = \{v_{i,1}, \ldots, v_{i,2d_i+1}\}$, and $V = \bigcup_{i=1}^{D} V_i$. Then we have that

$$\Delta_i = \left\{ \underbrace{\delta^-_{i,d_i}, \ldots, \delta^-_{i,d_i-(j-1)}, \ldots, \delta^-_{i,1}}_{\Delta_i^-}, 0, \underbrace{\delta^+_{i,1}, \ldots, \delta^+_{i,j}, \ldots, \delta^+_{i,d_i}}_{\Delta_i^+} \right\} \tag{5.8}$$

is equal to

$$V_i = \left\{ v_{i,1}, \ldots, v_{i,j}, \ldots, v_{i,d_i}, \underbrace{v_{i,d_i+1}}_{0}, v_{i,d_i+2}, \ldots, v_{i,d_i+1+j}, \ldots, v_{i,2d_i+1} \right\}, \tag{5.9}$$

where

$$\begin{aligned} v_{i,j} &\xrightarrow{\delta} \delta^-_{i,d_i-(j-1)}, \\ v_{i,d_i+1} &\xrightarrow{\delta} 0, \\ v_{i,d_i+1+j} &\xrightarrow{\delta} \delta^+_{i,j} \end{aligned} \tag{5.10}$$

and $j = 1, 2, \ldots, d_i$.

To enable a more flexible movement over the search space, the weight ω is added to Eq. 5.1:

$$p_i = p'_i + \omega\delta(v_i) \tag{5.11}$$

where $\omega = \text{RandomInteger}(1, b-1)$.

Each vertex of the set V_i is connected to all the vertices that belong to the set V_{i+1} (see Figure 5.1). Therefore, this is a directed graph, where each path v from the $start$ vertex to any of the ending vertices is of equal length and can be defined with v_i as:

$$v = v_1 v_2 \cdots v_D, \tag{5.12}$$

where $v_i \in V_i$, $i = 1, 2, \ldots, D$.

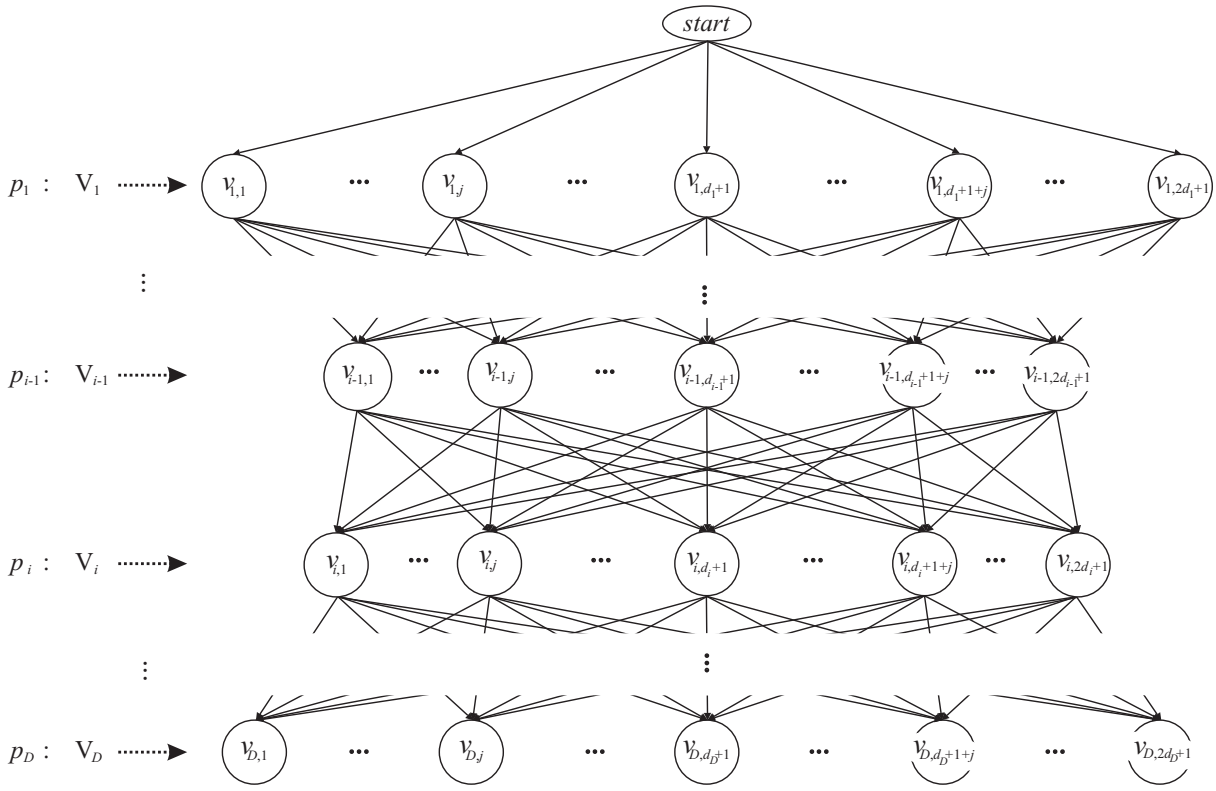

Figure 5.1 Differential graph representation.

The optimization task is to find a path v, such that $f(\mathbf{p}) < f(\mathbf{p}')$, where $\mathbf{p}'$ is currently the best solution, and $\mathbf{p} = \mathbf{p}' + \Delta(v)$ (using Eq. 5.11 for all elements of $\mathbf{p}$). Additionally, if the objective function $f(\mathbf{p})$ is smaller than $f(\mathbf{p}')$, then the $\mathbf{p}'$ values are replaced with $\mathbf{p}$ values.

5.2 The differential ant-stigmergy algorithm (DASA)

In this section we present a new optimization algorithm called the *Differential Ant-Stigmergy Algorithm* (DASA).

The optimization consists of an iterative improvement of the currently best solution, $\mathbf{p}'$, by constructing an appropriate path v, that uses Eq. 5.11 and returns a new best solution. The DASA pseudo code can be seen in Algorithm 5.1.

First a solution $\mathbf{p}'$ is manually set or randomly chosen. Then a search graph is created and an initial amount of pheromone (i.e., the minimum pheromone intensity under which the amount never drops), $\tau^0_{V_i}$, is deposited on all the vertices from the set $V_i \subset V, 1 \leq i \leq D$, according to a Gaussian probability density function

$$\text{Gauss}(p, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(p-\mu)^2}{2\sigma^2}}, \tag{5.13}$$

where μ is the mean, σ is the standard deviation, and $\mu = 0$, $\sigma = \sigma_{\max} = 1$ (see Figure 5.2).

There are m ants in a colony, all of which begin simultaneously from the $start$ vertex. Ants use a probability rule to determine which vertex will be chosen next. More specifically, ant α in step i moves from a vertex in set V_{i-1} to vertex $v_{i,j} \in \{v_{i,1}, \ldots, v_{i,2d_i+1}\}$ with a probability given by:

$$\text{prob}_j(\alpha, i) = \frac{\tau(v_{i,j})}{\sum_{1 \leq k \leq 2d_i+1} \tau(v_{i,k})}, \tag{5.14}$$

where $\tau(v_{i,k})$ is the amount of pheromone in vertex $v_{i,k}$. The ants repeat this action until they reach the ending vertex. For each ant, solution $\mathbf{p}$ is constructed (see Eq. 5.11) and evaluated with a calculation of $f(\mathbf{p})$. The best solution, $\mathbf{p}^b$, out of m solutions is compared to the currently best solution $\mathbf{p}'$. If $f(\mathbf{p}^b)$ is better than $f(\mathbf{p}')$, then $\mathbf{p}'$ values are replaced with $\mathbf{p}^b$ values. Furthermore, in this case the pheromone amount is redistributed according to the associated path $v^b = v^b_1 \cdots v^b_{i-1} v^b_i \cdots v^b_D$. New probability density functions have maxima in the vertices v^b_i and the standard deviations are inversely proportional to the improvements of the solutions (see Figure 5.3).

Pheromone evaporation is defined by some predetermined percentage ρ on each probability density function as follows:

Algorithm 5.1 Differential Ant-Stigmergy Algorithm

```
1: p' = RandomSolution(parameters)
2: best = Evaluate( p')
3: searchGraph = Initialization(parameters)
4: SearchGraphInitialization(initial pheromone amount)
5: while not ending condition met do
6:    currentBest = inf
7:    for all m ants do
8:       path = FindPath(searchGraph)
9:       p = p' + ωδ(path)
10:      result = Evaluate(p)
11:      if result < currentBest then
12:         currentBest = result
13:         bestPath = path
14:         p^b = p
15:      end if
16:   end for
17:   if currentBest < best then
18:      best = currentBest
19:      p' = p^b
20:      PheromoneRedistribution(bestPath)
21:   end if
22:   PheromoneEvaporation(searchGraph)
23:   DaemonAction                    //optional
24: end while
```

$$\mu^{\text{NEW}} = (1-\rho)\mu^{\text{OLD}} \tag{5.15}$$

and

$$\sigma^{\text{NEW}} = \begin{cases} (1+\rho)\sigma^{\text{OLD}} & (1+\rho)\sigma^{\text{OLD}} < \sigma_{\max} \\ \sigma_{\max} & \text{otherwise} \end{cases}. \tag{5.16}$$

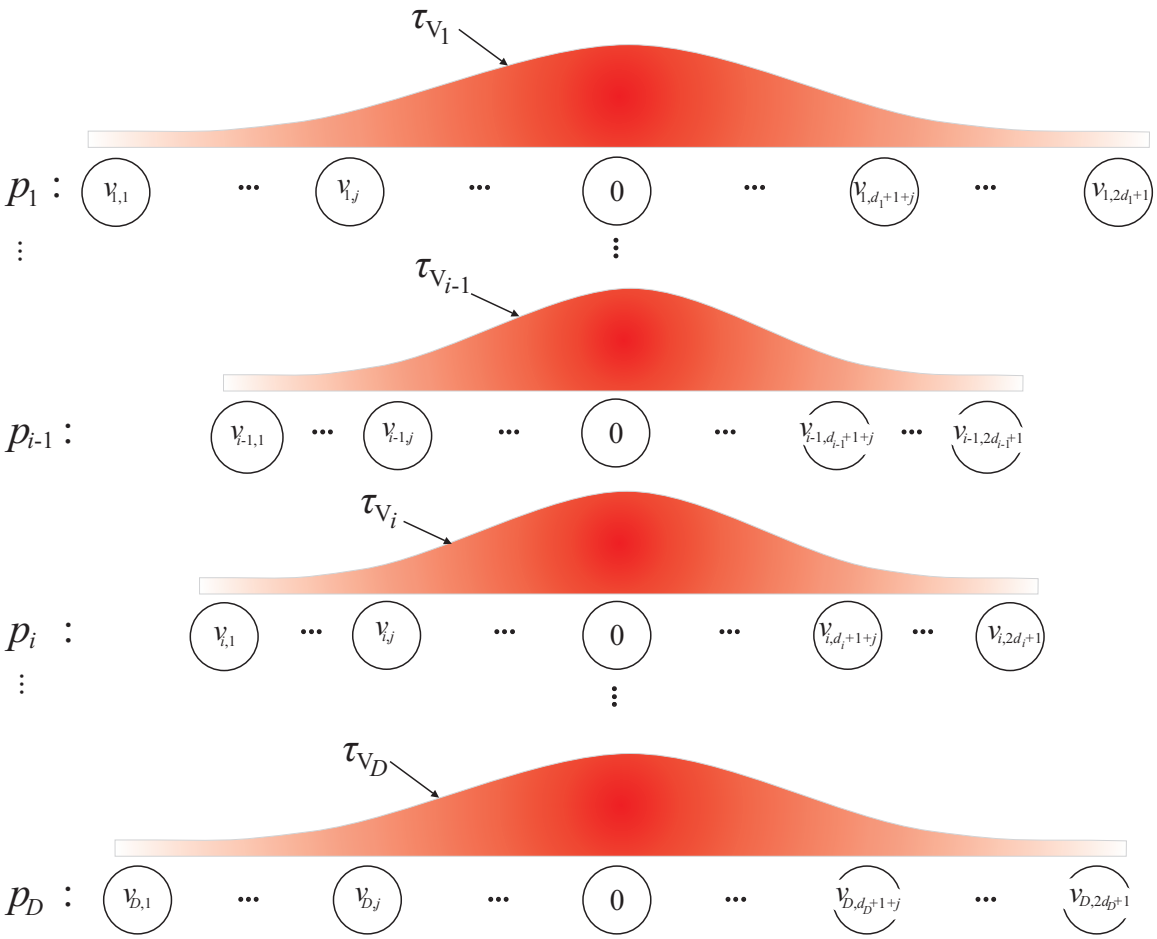

Figure 5.2 Initial pheromone distribution.

The whole procedure is then repeated until some ending condition is met. Through the iterations of the algorithm we slowly decrease the maximum standard deviation, $\sigma_{\max}$, and with it improve the convergence (an example of daemon action).

5.3 Performance evaluation

5.3.1 The experimental environment

The computer platform used to perform the experiments was based on AMD Opteron™ 2.6-GHz processor, 2 GB of RAM, and the Microsoft® Windows® XP operating system.

The DASA has three parameters: the number of ants, m, the pheromone evaporation factor, ρ, and the maximum parameter precision, ε. Their settings are: $m = 10$, $\rho = 0.1$, and $\varepsilon = 10^{-12}$. We must note that during the experimentation we did not fine-tune

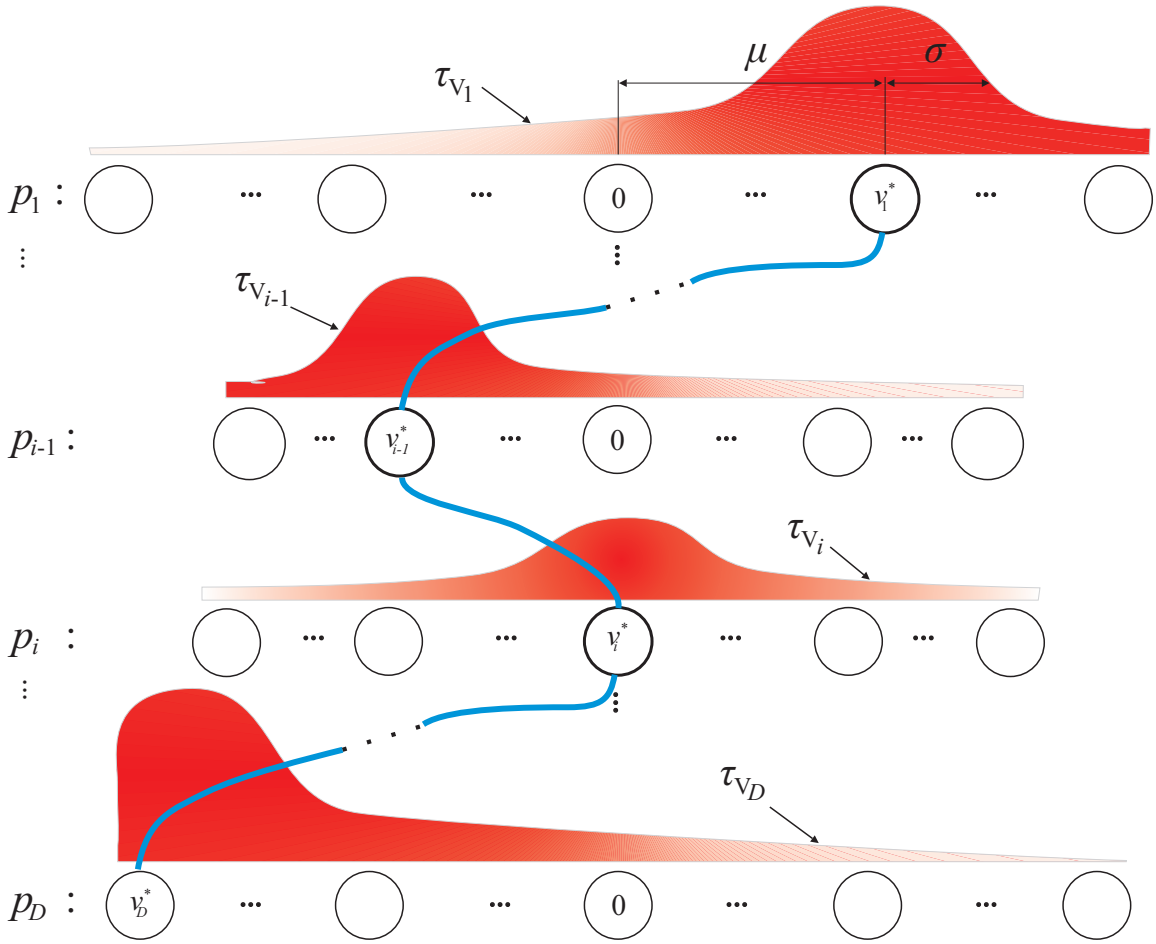

Figure 5.3 Pheromone distribution after a new best solution is found.

the algorithms parameters, but only make a limited number of experiments to find satisfying settings.

5.3.2 The benchmark suite

The DASA was tested on four benchmark functions of dimension 30. The complete definition of the test-suit is available in [114]. Function f_3 (*Shifted Rotated High Conditional Elliptic Function*) is unimodal and function f_9 (*Shifted Rastrigin's Function*) is multi-modal. Functions f_{13} (*Expanded Extended Griewangk's plus Rosenbrock's Function*) and f_{15} (*Hybrid Composition Function*) result from the composition of several functions. To prevent exploitation of the symmetry of the search space and of the typical zero value associated with the global optimum, the local optimum is shifted to a value different from zero, and the function value of the global optimum is non-zero.

5.3.3 Compared algorithms

The DASA was compared to the DE [100] (see Subsection 4.5.3) and results of three well-known algorithms:

- a restart *Covariance Matrix Adaptation Evolution Strategy* with increasing population size (CMA-ES) [3],
- a real-coded *Memetic Algorithm* (MA) [90],
- a continuous *Estimation of Distribution Algorithm* (EDA) [130].

The CMA-ES introduced by Hansen and Ostermeier [49] is an ES that adapts the full covariance matrix of a normal search (mutation) distribution. By increasing the population size for each restart—as suggested in [3]—the search characteristics become more global after each restart.

The MA is a GA that applies a separate local search process to refine new individuals. The GA applied to make the exploration (i.e., to maintain diversity in the population), the local search applied to improve new solutions (i.e., to exploit the most promising regions in the search space). In [90] a steady-state GA is used.

The EDA is based on probabilistic modeling instead of classical genetic operators such as crossover or mutation. The EDA used in [130] employs a multivariate Gaussian distribution to model selected individuals and generate new individuals and is therefore able to represent a correlation between variables in the selected individuals via the full covariance matrix of the system.

5.3.4 The complexity of the algorithm

To estimate the algorithm's complexity we have calculated $\frac{\widehat{T}_2 - T_1}{T_0}$, where computing time T_0 is independent of the function dimension and is calculated by running the benchmark algorithm described in Algorithm 4.10. T_1 is the computing time for $200{,}000$ evaluations only for function f_3 and $\widehat{T}_2$ is the mean time of five executions, but now considering the complete computing time of the algorithm for the function f_3. The results are included in Table 5.1.

Table 5.1 Algorithm complexity (function f_3, $D = 30$).

Algorithm	The system	T_0 [s]	T_1 [s]	$\widehat{T}_2$ [s]	$\frac{\widehat{T}_2 - T_1}{T_0}$
CMA-ES	Pentium 4 3 GHz / 1 GB Red Hat Linux 2.4 MATLAB 7.0.1	0.40	41.00	*24.00	—
DE	AMD Sempron 2800+ / 1 GB Mandrake Linux 10.1 C	0.29	7.64	8.49	2.94
MA	Pentium 4 2.8 GHz / 512 MB Linux kernel v. 2.6 C++ with GCC 3.3.2	0.42	8.63	13.45	11.48
EDA	Xeon 2.4 GHz / 1 GB Windows XP (SP2) MATLAB 6	**6.93	1.45	5.22	0.54
DASA	AMD Opteron 2.6 GHz / 2 GB Windows XP (SP2) Delphi 2006	0.19	58.94	59.20	1.37

* The large value of T_1 reflects the large number of objective function calls, while for T_2 a complete, eventually large, population is evaluated (serially) within a single function call.

** Due to poor loop implementation in MATLAB 6.

5.3.5 An evaluation

The function error, $f(\mathbf{p}) - f(\mathbf{p}^*)$ being with $\mathbf{p}^*$ the optimum, is recorded at four checkpoints (after 1,000, 10,000, 100,000, and 300,000 function evaluations). The error data is collected for 25 runs after which the trials are ordered from best to worst. The trial mean and the standard deviation as well as the results of the best, median, and worst trials are presented for each of the four checkpoints. The error values are presented in Tables 5.2–5.5.

Table 5.2 Error values for the thirty-dimensional *Shifted Rotated High Conditional Elliptic Function* (f_3), measured after $1{,}000$, $10{,}000$, $100{,}000$, and $300{,}000$ function evaluations.

Function evaluations		Algorithm				
		CMA-ES	DE	MA	EDA	DASA
10^3	*Best*	$3.84 \cdot 10^{8}$	$2.18 \cdot 10^{8}$	$9.63 \cdot 10^{7}$	$8.95 \cdot 10^{8}$	$6.11 \cdot 10^{7}$
	Median	$1.00 \cdot 10^{9}$	$5.66 \cdot 10^{8}$	$2.69 \cdot 10^{8}$	$1.23 \cdot 10^{9}$	$2.80 \cdot 10^{8}$
	Worst	$2.07 \cdot 10^{9}$	$9.53 \cdot 10^{8}$	$5.82 \cdot 10^{8}$	$1.92 \cdot 10^{9}$	$5.57 \cdot 10^{8}$
	Mean	$1.07 \cdot 10^{9}$	$5.53 \cdot 10^{8}$	$2.94 \cdot 10^{8}$	$1.25 \cdot 10^{9}$	$3.10 \cdot 10^{8}$
	Std	$4.43 \cdot 10^{8}$	$1.78 \cdot 10^{8}$	$3.04 \cdot 10^{7}$	$2.67 \cdot 10^{8}$	$1.31 \cdot 10^{8}$
10^4	*Best*	$1.24 \cdot 10^{6}$	$3.58 \cdot 10^{7}$	$1.81 \cdot 10^{7}$	$1.79 \cdot 10^{8}$	$4.55 \cdot 10^{6}$
	Median	$4.90 \cdot 10^{6}$	$6.90 \cdot 10^{7}$	$4.17 \cdot 10^{7}$	$2.71 \cdot 10^{8}$	$1.15 \cdot 10^{7}$
	Worst	$1.42 \cdot 10^{7}$	$1.66 \cdot 10^{8}$	$8.51 \cdot 10^{7}$	$3.84 \cdot 10^{8}$	$1.95 \cdot 10^{7}$
	Mean	$6.11 \cdot 10^{6}$	$8.15 \cdot 10^{7}$	$4.14 \cdot 10^{7}$	$2.76 \cdot 10^{8}$	$1.16 \cdot 10^{7}$
	Std	$3.79 \cdot 10^{6}$	$3.25 \cdot 10^{7}$	$2.95 \cdot 10^{6}$	$5.03 \cdot 10^{7}$	$4.44 \cdot 10^{6}$
10^5	*Best*	$4.07 \cdot 10^{-9}$	$3.89 \cdot 10^{5}$	$1.76 \cdot 10^{6}$	$2.41 \cdot 10^{7}$	$5.77 \cdot 10^{5}$
	Median	$5.44 \cdot 10^{-9}$	$1.33 \cdot 10^{6}$	$4.91 \cdot 10^{6}$	$3.55 \cdot 10^{7}$	$1.07 \cdot 10^{6}$
	Worst	$8.66 \cdot 10^{-9}$	$3.38 \cdot 10^{6}$	$6.80 \cdot 10^{6}$	$4.55 \cdot 10^{7}$	$1.94 \cdot 10^{6}$
	Mean	$5.55 \cdot 10^{-9}$	$1.52 \cdot 10^{6}$	$5.51 \cdot 10^{6}$	$3.49 \cdot 10^{7}$	$1.23 \cdot 10^{6}$
	Std	$1.09 \cdot 10^{-9}$	$8.92 \cdot 10^{-5}$	$6.05 \cdot 10^{5}$	$4.94 \cdot 10^{6}$	$3.97 \cdot 10^{5}$
$3 \cdot 10^5$	*Best*	$4.07 \cdot 10^{-9}$	$5.46 \cdot 10^{4}$	$5.55 \cdot 10^{5}$	$2.27 \cdot 10^{6}$	$1.27 \cdot 10^{5}$
	Median	$5.44 \cdot 10^{-9}$	$2.43 \cdot 10^{5}$	$7.64 \cdot 10^{5}$	$3.66 \cdot 10^{6}$	$4.32 \cdot 10^{5}$
	Worst	$8.66 \cdot 10^{-9}$	$9.00 \cdot 10^{5}$	$1.56 \cdot 10^{6}$	$5.88 \cdot 10^{6}$	$8.15 \cdot 10^{5}$
	Mean	$5.55 \cdot 10^{-9}$	$2.89 \cdot 10^{5}$	$8.77 \cdot 10^{5}$	$3.75 \cdot 10^{6}$	$4.59 \cdot 10^{5}$
	Std	$1.09 \cdot 10^{-9}$	$1.93 \cdot 10^{5}$	$5.81 \cdot 10^{4}$	$9.09 \cdot 10^{5}$	$2.02 \cdot 10^{5}$

Table 5.3 Error values for the thirty-dimensional *Shifted Rastrigin's Function* (f_9), measured after $1{,}000$, $10{,}000$, $100{,}000$, and $300{,}000$ function evaluations.

Function evaluations		Algorithm				
		CMA-ES	DE	MA	EDA	DASA
10^3	*Best*	$2.19 \cdot 10^2$	$2.99 \cdot 10^2$	$1.82 \cdot 10^2$	$4.07 \cdot 10^2$	$4.60 \cdot 10^1$
	Median	$2.50 \cdot 10^2$	$3.72 \cdot 10^2$	$3.00 \cdot 10^2$	$4.76 \cdot 10^2$	$9.13 \cdot 10^1$
	Worst	$2.87 \cdot 10^2$	$4.25 \cdot 10^2$	$4.00 \cdot 10^2$	$5.44 \cdot 10^2$	$1.52 \cdot 10^2$
	Mean	$2.53 \cdot 10^2$	$3.77 \cdot 10^2$	$2.99 \cdot 10^2$	$4.80 \cdot 10^2$	$9.29 \cdot 10^1$
	Std	$1.65 \cdot 10^1$	$3.00 \cdot 10^1$	$1.00 \cdot 10^1$	$3.51 \cdot 10^1$	$2.75 \cdot 10^1$
10^4	*Best*	$2.39 \cdot 10^1$	$8.17 \cdot 10^1$	$6.28 \cdot 10^1$	$3.23 \cdot 10^2$	$9.95 \cdot 10^{-1}$
	Median	$4.88 \cdot 10^1$	$9.74 \cdot 10^1$	$1.04 \cdot 10^2$	$3.66 \cdot 10^2$	2.99
	Worst	$7.96 \cdot 10^1$	$1.13 \cdot 10^2$	$1.50 \cdot 10^2$	$3.87 \cdot 10^2$	4.98
	Mean	$4.78 \cdot 10^1$	$9.85 \cdot 10^1$	$1.05 \cdot 10^2$	$3.62 \cdot 10^2$	2.95
	Std	$1.15 \cdot 10^1$	8.42	3.17	$1.62 \cdot 10^1$	1.17
10^5	*Best*	2.98	$1.90 \cdot 10^{-8}$	3.98	$2.18 \cdot 10^2$	0
	Median	6.96	$5.93 \cdot 10^{-8}$	7.96	$2.50 \cdot 10^2$	0
	Worst	$1.19 \cdot 10^1$	$1.39 \cdot 10^{-7}$	$1.19 \cdot 10^1$	$2.78 \cdot 10^2$	0
	Mean	6.89	$6.68 \cdot 10^{-8}$	7.55	$2.50 \cdot 10^2$	0
	Std	2.22	$3.39 \cdot 10^{-8}$	$5.36 \cdot 10^{-1}$	$1.34 \cdot 10^1$	0
$3 \cdot 10^5$	*Best*	$4.35 \cdot 10^{-6}$	0	$7.78 \cdot 10^{-9}$	$2.10 \cdot 10^2$	0
	Median	$9.95 \cdot 10^{-1}$	0	$9.95 \cdot 10^{-1}$	$2.30 \cdot 10^2$	0
	Worst	4.97	0	1.99	$2.48 \cdot 10^2$	0
	Mean	$9.38 \cdot 10^{-1}$	0	$6.81 \cdot 10^{-1}$	$2.30 \cdot 10^2$	0
	Std	1.18	0	$1.21 \cdot 10^{-1}$	9.44	0

Table 5.4 Error values for the thirty-dimensional *Expanded Extended Griewangk's plus Rosenbrock's Function* (f_{13}), measured after $1{,}000$, $10{,}000$, $100{,}000$, and $300{,}000$ function evaluations.

Function evaluations		Algorithm				
		CMA-ES	DE	MA	EDA	DASA
10^3	*Best*	$3.05 \cdot 10^{1}$	$3.12 \cdot 10^{4}$	$4.09 \cdot 10^{2}$	$4.66 \cdot 10^{5}$	$1.33 \cdot 10^{4}$
	Median	$7.36 \cdot 10^{1}$	$1.29 \cdot 10^{5}$	$3.86 \cdot 10^{3}$	$7.39 \cdot 10^{5}$	$1.38 \cdot 10^{5}$
	Worst	$4.98 \cdot 10^{2}$	$4.33 \cdot 10^{5}$	$1.06 \cdot 10^{4}$	$1.13 \cdot 10^{6}$	$6.47 \cdot 10^{5}$
	Mean	$1.14 \cdot 10^{2}$	$1.62 \cdot 10^{5}$	$3.95 \cdot 10^{3}$	$7.50 \cdot 10^{5}$	$2.12 \cdot 10^{5}$
	Std	$1.07 \cdot 10^{2}$	$8.66 \cdot 10^{4}$	$4.62 \cdot 10^{2}$	$1.93 \cdot 10^{5}$	$1.81 \cdot 10^{5}$
10^4	*Best*	2.46	$3.20 \cdot 10^{1}$	9.97	$1.35 \cdot 10^{5}$	2.57
	Median	3.87	$8.02 \cdot 10^{1}$	$1.49 \cdot 10^{1}$	$2.97 \cdot 10^{5}$	6.34
	Worst	5.62	$2.47 \cdot 10^{2}$	$1.96 \cdot 10^{1}$	$5.24 \cdot 10^{5}$	$1.38 \cdot 10^{1}$
	Mean	3.80	$1.02 \cdot 10^{2}$	$1.51 \cdot 10^{1}$	$3.08 \cdot 10^{5}$	7.02
	Std	$7.27 \cdot 10^{-1}$	$6.33 \cdot 10^{1}$	$4.49 \cdot 10^{-1}$	$1.14 \cdot 10^{5}$	3.33
10^5	*Best*	2.43	2.31	2.76	$1.84 \cdot 10^{3}$	1.20
	Median	2.83	3.90	9.07	$4.30 \cdot 10^{3}$	2.02
	Worst	3.67	$1.39 \cdot 10^{1}$	$1.28 \cdot 10^{1}$	$9.93 \cdot 10^{3}$	2.73
	Mean	2.89	4.55	8.66	$4.52 \cdot 10^{3}$	2.04
	Std	$3.59 \cdot 10^{-1}$	2.25	$4.42 \cdot 10^{-1}$	$1.91 \cdot 10^{3}$	$4.17 \cdot 10^{-1}$
$3 \cdot 10^5$	*Best*	1.10	2.31	1.33	$3.82 \cdot 10^{1}$	$9.62 \cdot 10^{-1}$
	Median	2.61	3.89	2.54	$6.86 \cdot 10^{1}$	1.93
	Worst	3.20	$1.39 \cdot 10^{1}$	$1.03 \cdot 10^{1}$	$1.29 \cdot 10^{2}$	2.56
	Mean	2.49	4.51	3.96	$7.36 \cdot 10^{1}$	1.88
	Std	$5.13 \cdot 10^{-1}$	2.26	$5.38 \cdot 10^{-1}$	$2.36 \cdot 10^{1}$	$3.99 \cdot 10^{-1}$

Table 5.5 Error values for the thirty-dimensional *Hybrid Composition Function* (f_{15}), measured after $1{,}000$, $10{,}000$, $100{,}000$, and $300{,}000$ function evaluations.

Function evaluations		Algorithm				
		CMA-ES	DE	MA	EDA	DASA
10^3	*Best*	$4.93 \cdot 10^2$	$8.82 \cdot 10^2$	$5.46 \cdot 10^2$	$1.03 \cdot 10^3$	$2.32 \cdot 10^2$
	Median	$6.93 \cdot 10^2$	$1.08 \cdot 10^3$	$7.49 \cdot 10^2$	$1.14 \cdot 10^3$	$6.28 \cdot 10^2$
	Worst	$8.51 \cdot 10^2$	$1.19 \cdot 10^3$	$1.05 \cdot 10^3$	$1.21 \cdot 10^3$	$7.84 \cdot 10^2$
	Mean	$6.69 \cdot 10^2$	$1.08 \cdot 10^3$	$7.62 \cdot 10^2$	$1.13 \cdot 10^3$	$5.89 \cdot 10^2$
	Std	$1.15 \cdot 10^2$	$7.13 \cdot 10^1$	$2.64 \cdot 10^1$	$4.50 \cdot 10^1$	$1.50 \cdot 10^2$
10^4	*Best*	$2.08 \cdot 10^2$	$6.17 \cdot 10^2$	$3.72 \cdot 10^2$	$5.90 \cdot 10^2$	$4.17 \cdot 10^{-4}$
	Median	$4.00 \cdot 10^2$	$6.88 \cdot 10^2$	$4.30 \cdot 10^2$	$6.31 \cdot 10^2$	$3.05 \cdot 10^2$
	Worst	$5.53 \cdot 10^2$	$8.36 \cdot 10^2$	$5.42 \cdot 10^2$	$8.82 \cdot 10^2$	$5.00 \cdot 10^2$
	Mean	$3.87 \cdot 10^2$	$7.04 \cdot 10^2$	$4.41 \cdot 10^2$	$6.88 \cdot 10^2$	$2.40 \cdot 10^2$
	Std	$8.48 \cdot 10^1$	$6.30 \cdot 10^1$	7.96	$9.93 \cdot 10^1$	1.59
10^5	*Best*	$2.00 \cdot 10^2$	$5.03 \cdot 10^2$	$2.00 \cdot 10^2$	$4.85 \cdot 10^2$	0
	Median	$2.00 \cdot 10^2$	$5.18 \cdot 10^2$	$3.00 \cdot 10^2$	$4.89 \cdot 10^2$	$3.00 \cdot 10^2$
	Worst	$3.20 \cdot 10^2$	$6.33 \cdot 10^2$	$5.00 \cdot 10^2$	$6.71 \cdot 10^2$	$5.00 \cdot 10^2$
	Mean	$2.25 \cdot 10^2$	$5.20 \cdot 10^2$	$3.56 \cdot 10^2$	$5.38 \cdot 10^2$	$2.33 \cdot 10^2$
	Std	$4.10 \cdot 10^1$	$2.39 \cdot 10^1$	$1.51 \cdot 10^1$	$7.66 \cdot 10^1$	$1.58 \cdot 10^2$
$3 \cdot 10^5$	*Best*	$2.00 \cdot 10^2$	$4.75 \cdot 10^2$	$2.00 \cdot 10^2$	$4.35 \cdot 10^2$	0
	Median	$2.00 \cdot 10^2$	$4.81 \cdot 10^2$	$3.00 \cdot 10^2$	$4.59 \cdot 10^2$	$3.00 \cdot 10^2$
	Worst	$3.00 \cdot 10^2$	$5.86 \cdot 10^2$	$5.00 \cdot 10^2$	$5.63 \cdot 10^2$	$5.00 \cdot 10^2$
	Mean	$2.08 \cdot 10^2$	$4.84 \cdot 10^2$	$3.56 \cdot 10^2$	$4.81 \cdot 10^2$	$2.33 \cdot 10^2$
	Std	$2.75 \cdot 10^1$	$2.14 \cdot 10^1$	$1.51 \cdot 10^1$	$4.67 \cdot 10^1$	$1.58 \cdot 10^2$

The results presented in Tables 5.2–5.5 indicate the promising performance of the new approach. It is clear that our approach performs better than the rest of the approaches on three out of four test functions. Since the selected test functions reflect different kinds of pseudo-real optimization problems, one could expect that the DASA is applicable to many real-world multi-parameter optimization problems.

The convergence rates of the DASA on functions f_3, f_9, f_{13}, and f_{15} are plotted in Figure 5.4. The rates show the average performance of the 25 runs. In the figure, the function error is plotted against the number of evaluations.

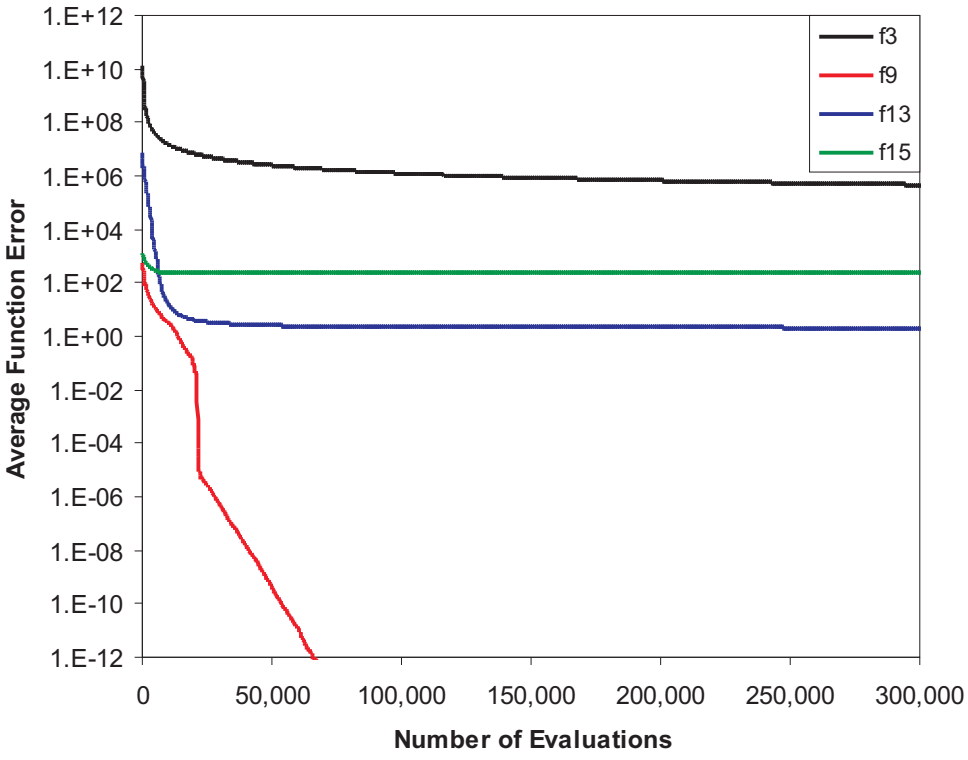

Figure 5.4 Convergence graph for the DASA on f_3, f_9, f_{13}, and f_{15} functions.

Even though it was shown that the MASA and the DASA perform well on benchmark functions, one always wonders how the algorithm will work on solving real-world applications. This question is answered in the next chapter.

6 Real-world applications

6.1 Minimizing the power losses of a universal electric motor

In this section we report on numerical experiments in optimizing the power losses of a universal electric motor for Domel Železniki - Electromotors and Household Devices from Železniki, Slovenia (see Figure 6.1) using a stigmergic algorithm.

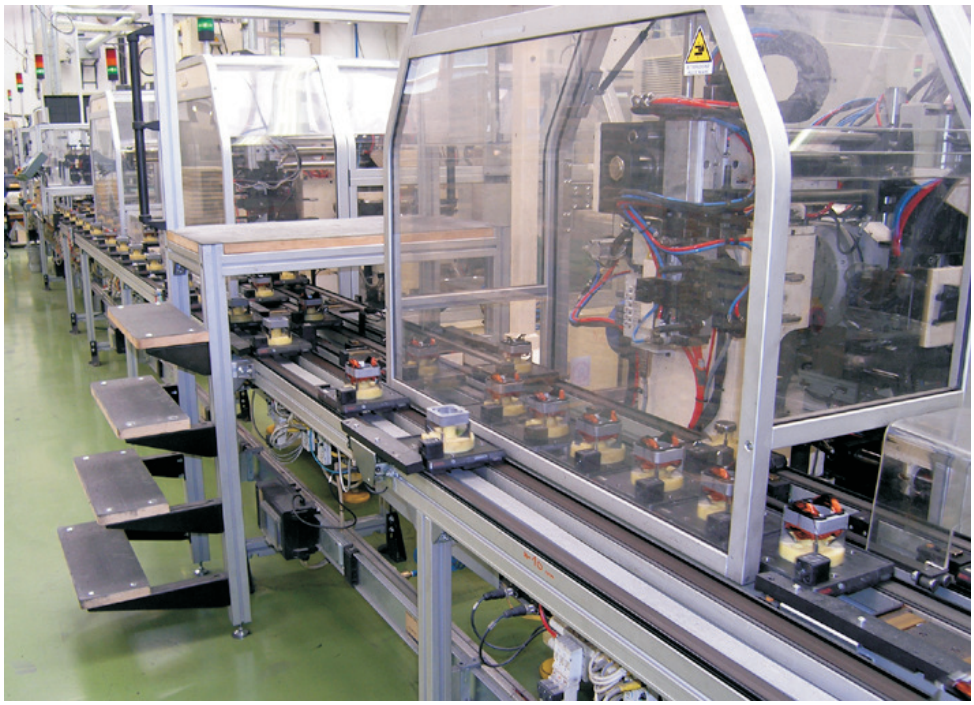

Figure 6.1 Electric motor production line. Photo courtesy of the Domel, Železniki, Slovenia.

6.1.1 Definition of the problem

Power losses are the main factor affecting the efficiency of a universal electric motor (UM). The efficiency of a UM, η, is defined as the ratio of the output power, P_{out}, to the input power, P_{inp}, and it is very dependent on various power losses.

The overall copper losses occurring in the rotor and the stator slots are as follows:

$$P_{\mathrm{Cu}} = \sum_i \left(J^2 A \rho l_{\mathrm{turn}}\right)_i, \tag{6.1}$$

where i stands for each slot, J is the current density, A is the slot area, ρ is the copper's specific resistance and l_{turn} is the length of the winding turn.

The calculation of the iron losses is less exact because iron has non-linear magnetic characteristics. The iron losses are therefore separated into two components: the hysteresis losses and the eddy-current losses. This means the motor's iron losses can be expressed with the formula

$$P_{\mathrm{Fe}} = c_{\mathrm{e}} B^2 f_{\mathrm{r}}^2 m_{\mathrm{r}} + c_{\mathrm{e}} B^2 f_{\mathrm{s}}^2 m_{\mathrm{s}} + c_{\mathrm{h}} B^2 f_{\mathrm{s}} m_{\mathrm{s}}, \tag{6.2}$$

where c_{e} is the eddy-current material constant at 50Hz, c_{h} is the hysteresis material constant at 50Hz, B is the maximum magnetic flux density, f_{r} is the frequency of the magnetic field density in the rotor, f_{s} is the frequency of the magnetic field density in the stator, m_{r} is the mass of the rotor, and m_{s} is the mass of the stator.

Three additional types of losses occurring in the UM, i.e., the brush losses, P_{brush}; the ventilation losses, P_{vent}; and the friction losses, P_{frict}; depend mainly on the speed of the motor. When optimizing the geometries of the rotor and the stator, the motor's speed is assumed to be fixed; hence P_{brush}, P_{vent}, and P_{frict} have no impact on the motor's efficiency, and so these losses are not significantly affected by the geometries of the rotor and the stator.

The output power, P_{out}, of the UM is the product of the electromagnetic torque, T, and the angular velocity, ω,

$$P_{\mathrm{out}} = T\omega, \tag{6.3}$$

where ω is set by the motor's speed, and T is the vector product of the distance from the origin and the electromagnetic force.

Taking into account all these losses and the output power, the overall efficiency of the UM can be defined as follows:

$$\eta = \frac{P_{\mathrm{out}}}{P_{\mathrm{inp}}} = \frac{P_{\mathrm{out}}}{P_{\mathrm{out}} + P_{\mathrm{Cu}} + P_{\mathrm{Fe}} + P_{\mathrm{brush}} + P_{\mathrm{vent}} + P_{\mathrm{frict}}}. \tag{6.4}$$

The rotor and the stator of the UM are constructed by stacking the iron laminations. The shape of these laminations is described by several parameters that define the rotor and stator geometries in two dimensions.

There are several invariable and variable parameters that define the rotor and stator geometries [94]. The invariable parameters are fixed; they cannot be altered, either for technical reasons (e.g., the air gap) or because of the physical constraints on the motor (e.g., the radius of the rotor's shaft). The variable parameters do not have predefined optimum values. Some variable parameters are mutually independent and without any constraints. Others are dependent, either on some invariable parameters or on mutually independent ones.

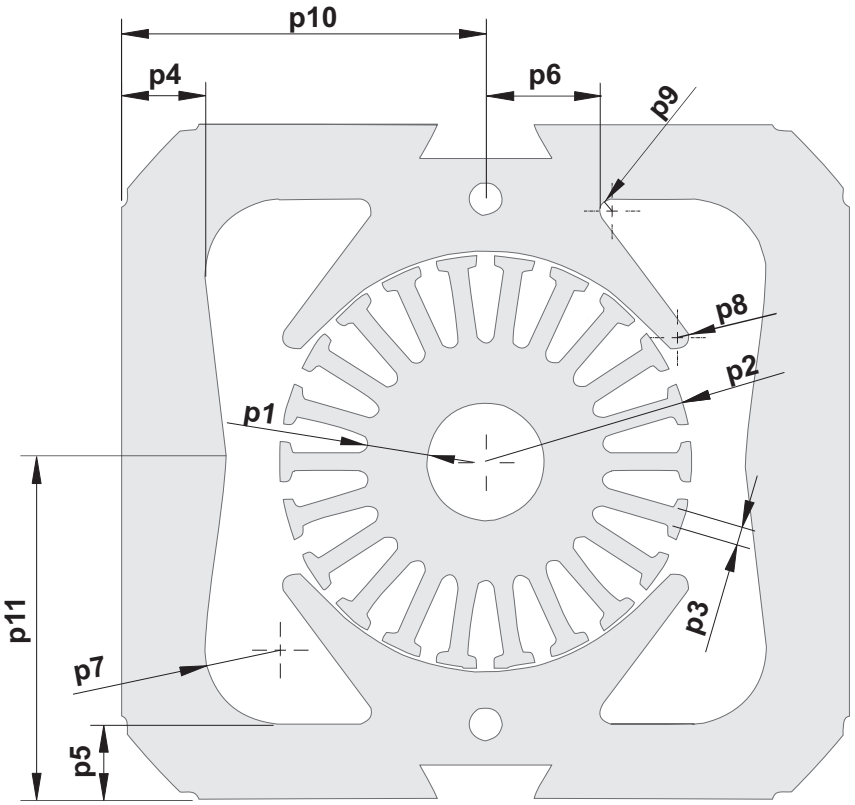

Figure 6.2 The rotor and the stator laminations with 11 mutually independent variable parameters defining the rotor and the stator geometries.

In our case, 11 mutually independent variable parameters defining the rotor and stator geometries are subject to optimization (see Figure 6.2):

- rotor yoke thickness (p_1),

- rotor external radius (p_2),
- rotor pole width (p_3).
- stator yoke horizontal thickness (p_4),
- stator yoke vertical thickness (p_5),
- stator middle-part length (p_6),
- stator internal edge radius (p_7),
- stator teeth radius (p_8),
- stator slot radius (p_9),
- stator width (p_{10}),
- stator height (p_{11}).

The optimization task is to find the geometrical parameter values that will generate the rotor and the stator geometries resulting in the minimum power losses [61].

Table 6.1 Number of possible settings for the optimized UM geometrical parameters.

	Parameter	Number of settings
p_1	rotor yoke thickness	130
p_2	rotor external radius	200
p_3	rotor pole width	45
p_4	stator yoke horizontal thickness	170
p_5	stator yoke vertical thickness	270
p_6	stator middle-part length	150
p_7	stator internal edge radius	140
p_8	stator teeth radius	37
p_9	stator slot radius	45
p_{10}	stator width	102
p_{11}	stator height	250

Predefined search intervals for the rotor and stator parameter values are used and the discrete step for all the parameters is set to 0.1 mm. Table 6.1 summarizes the number of possible settings for each parameter. In this way a search graph with $1{,}540$ vertices was created. Therefore, the size of the search space can be obtained as the product of the number of possible settings over all the parameters. It turns out that there are approximately $4.8 \cdot 10^{22}$ possible solutions.

A reliable numerical simulation is a prerequisite for an automated design optimization. To evaluate the settings of the geometrical parameters of the rotor and the stator

with respect to the resulting power losses, we used the ANSYS Multiphysics finite-element-method simulation package [1]. This software provides a variety of structural, thermal, fluid and electromagnetic analyzes capabilities suitable for simulating arbitrary products under real-world conditions.

For the purpose of our study, the simulation software was connected with the optimization algorithm. The communication within the optimizer-simulator loop was made via file data transfer. To evaluate a geometry parameter setting, an optimization algorithm generated a script file containing the parameter values and activated the simulation software. This software read the parameter values from the script file, performed the finite-element calculation of the motor's electromagnetic states, assessed the power losses, and wrote the result to an output file. This result was then accepted by the algorithm, which then used it as a cost value for the evaluated solution.

If the solutions are created as random points in the search space, many of them are not feasible, i.e., they result in an impractical geometry. To estimate the proportion of infeasible solutions in the search space, a simple experiment was carried out: $30{,}000$ points were created randomly in the search space and among them only seven were found to be feasible.

The evaluation of a single solution is a time-consuming task. For example, an evaluation using the ANSYS simulation on an AMD Opteron™ 1.8-GHz computer takes approximately two minutes if the solution is feasible, otherwise it takes only a second to recognize an infeasible solution. To find an acceptable solution a few thousand evaluations are needed.

6.1.2 Optimization with the sequential MASA

As explained in Subsection 6.1.1, we can optimize 11 parameters of the UM rotor and stator geometries, but we have decided to optimize only the first ten parameters because the ratio between p_{10} and p_{11} was set constant. As a result, the size of the reduced search space is approximately $1.92 \cdot 10^{20}$ points.

For our test purposes we compared the MASA with four well-known population-based optimization methods [120], the Generational Evolutionary Algorithm (GEA) [40, 51], the Steady State Evolutionary Algorithm (SSEA) [128], the DE and the Electro-

magnetism-like Algorithm (EM) [9]. They start with an initial population of solutions. If the initial solutions are created as random points in the search space, many of them are infeasible, i.e., the result in a nonfunctional geometry. To assist the population-based methods in finding feasible solutions, they were modified to start not with a population of random solutions, but rather with a population of solutions that are random perturbations of a predefined solution. For this purpose, an engineering solution was used, specifying the UM rotor and stator geometries already used in practice.

Unlike the other applied methods that gradually improve the solutions, the MASA is a solution-construction method. Initial experiments with the MASA on the UM geometry-optimization problem have shown that this algorithm is capable of successfully navigating the search from infeasible to feasible regions. The multilevel approach significantly reduces the search space in the early stages of the exploration. This reduction enables the MASA to perform well without any background information on the feasibility of the solutions. Through stigmergy, the infeasible regions in the search space are found less attractive by the ants, and consequently the search focuses on other search regions are more likely to contain feasible solutions.

The stopping criterion for all the optimization methods was given by the number of solutions to be evaluated. It was set to $1{,}400$, and this value was chosen after considering the computational complexity of the optimization procedure. The evaluation of a single solution via the ANSYS simulation on an AMD Opteron™ 1.8-GHz computer takes approximately two minutes, which means that the execution of $1{,}400$ evaluations needs about two days. The remaining algorithm parameters were set as follows. The evolutionary algorithms GEA, SSEA and DE all used the population size of 20 and the crossover probability of 0.9. The GEA and SSEA used the mutation probability of 0.05 and the tournament size of 2. The scaling factor in the DE was set to 0.5. The EM used the population size of 20, the magnitude factor of moves of 1.3, and the local search procedure was disabled. The settings for the MASA were determined in terms of the optimization parameter with the highest number of possible values. This was the stator yoke's vertical thickness, with 270 possible settings (see Table 6.1). Accordingly, the MASA operated with seven levels; at each level, 200 evaluations were performed by

10 ants climbing down the graph, the pheromone evaporation factor, ρ, was set to 0.1, and the coarsening was implemented by merging two vertices into one.

Table 6.2 Result statistics for the optimization methods (UM power losses in watts).

Method	Best	Avg	Worst	Std
EM	134.9	141.9	148.0	3.7
SSEA	131.4	137.7	148.8	5.4
GEA	131.3	136.7	147.4	4.5
DE	129.1	132.9	139.9	3.3
MASA	114.2	128.9	135.9	7.8

The optimization methods were run 20 times. The obtained results in terms of the UM power losses are presented statistically in Table 6.2. The method performance diagrams are shown in Figure 6.3, and they are compared with the power losses of the

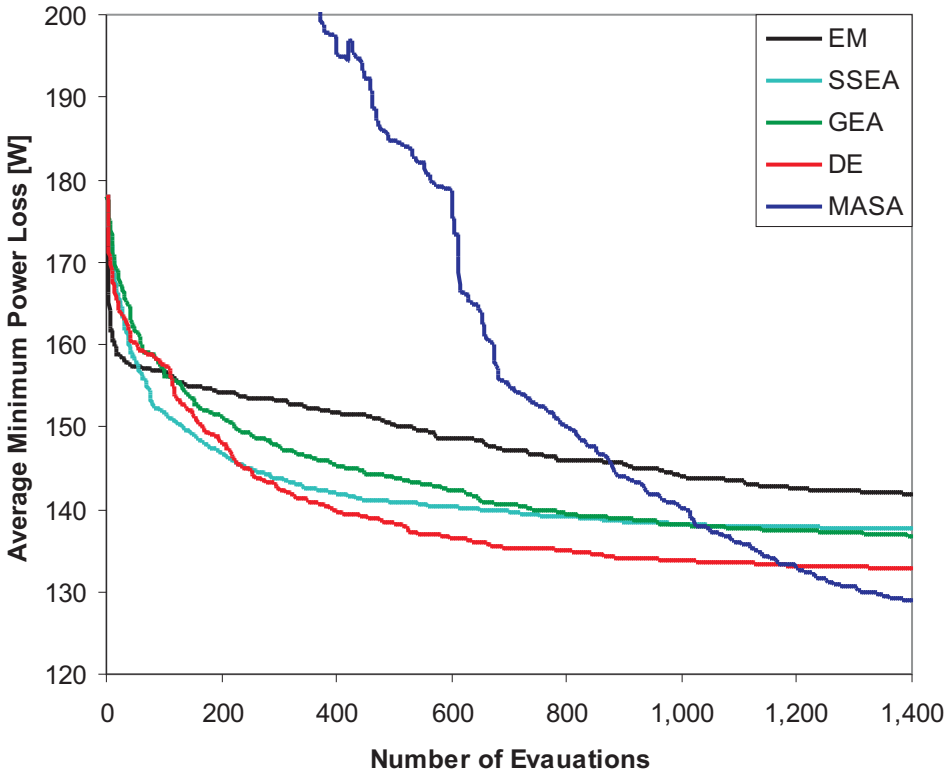

Figure 6.3 Performance of the applied methods in optimizing the UM rotor and stator geometry parameters: averages over 20 runs.

original engineering solution, which amounted to 177.9 W. The performance of the MASA is shown in Figure 6.4.

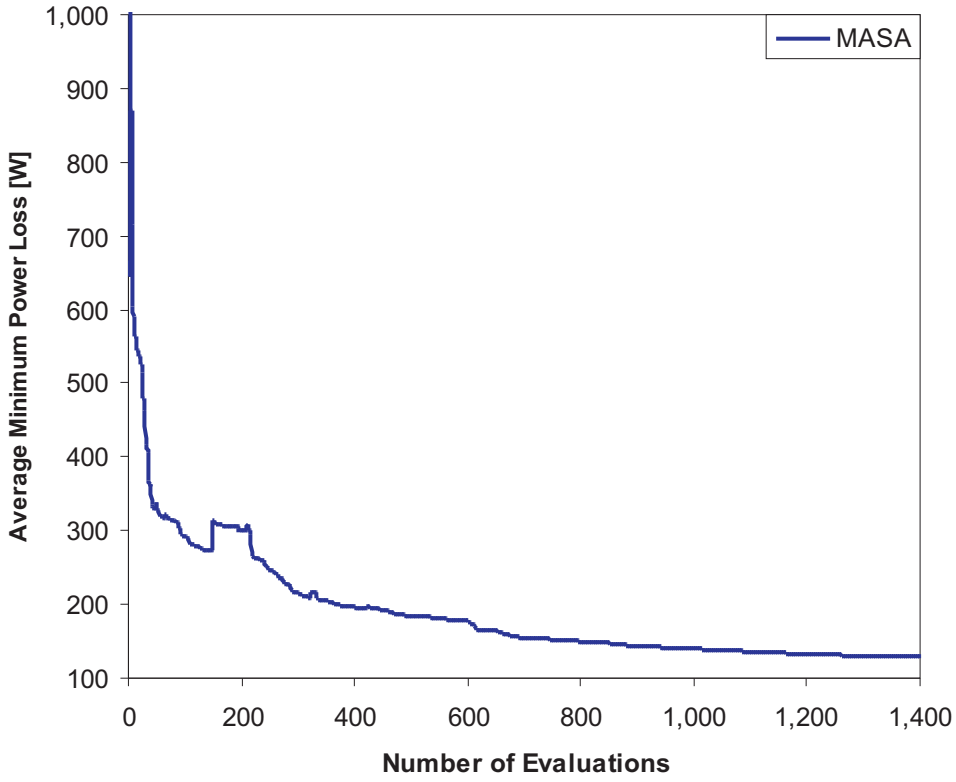

Figure 6.4 Performance of the MASA, averaged over 20 runs.

First of all the results show that all the applied methods significantly improve the original engineering design of the UM rotor and stator. Specifically, the geometry parameter settings with minimum power losses were found by the MASA. The second best method was the DE, followed by the GEA and the SSEA, which performed comparatively well, and the EM, which performed a little worse. Figure 6.3 shows the difference in the course of the optimization of the five applied methods. While the DE, the GEA, the SSEA and the EM all start with the engineering solution originally used in the motor's production and evolve rather slowly, the MASA starts with randomly created solutions that result in high power losses, but rapidly improve during the course of the run (see Figure 6.4). As can be seen in Figure 6.4, the average performance of the MASA is not monotonic, as only feasible solutions are taken into account when calcu-

lating the average. Infeasible solutions get a predefined high value of losses during the evaluation procedure and are excluded when calculating the average. Furthermore, when the MASA finds the first feasible solution, this solution typically results in high power losses and can increase the average value. This phenomenon disappears when all 20 runs reach feasible solutions. In the experiments shown, this happened after 421 evaluations.

In 6 out of 20 runs, the MASA was able to find the geometry parameter values resulting in power losses under 120 W. In the remaining 14 runs it performed similarly to the DE.

To check for any possible further improvement, the best solution from each run of every applied optimization method was subject to local search. A type of steepest-descent local search (see Algorithm 4.4) was applied and the procedure was not limited by a predefined number of steps. Rather, it stopped when a local minimum was encountered. The statistics of the improvements is given in Table 6.3.

Table 6.3 Improvement of the results with local search (UM power losses in watts).

Method	Best w/o LS	Best with LS	Avg w/o LS	Avg with LS	Avg improv.	Addit. steps
EM	134.9	133.4	141.9	139.3	2.6	153
SSEA	131.4	130.5	137.7	137.4	0.4	49
GEA	131.3	129.5	136.7	135.7	1.1	76
DE	129.1	129.1	132.9	132.5	0.4	50
MASA	114.2	111.1	128.9	126.2	2.7	116

From the results in Table 6.3 one can see that the solutions found by the evolutionary methods, i.e., the SSEA, the GEA and the DE were closer to the local minima than the solutions found by the MASA and the EM, i.e., fewer steps were needed to reach a local minimum from the solutions obtained by evolutionary methods than the MASA and the EM. Accordingly, the former could be improved with LS to a smaller extent than the latter. Moreover, even together with LS, the evolutionary methods could not find the solutions with power losses of about 115 W. They seemed to be stuck in local minima, with values of around 130 W. This was also noted for those solutions produced by the MASA that had power-loss values near 130 W before local search.

By applying selected stochastic optimization methods and locally improving the obtained solutions, we found various UM rotor and stator geometry parameter settings to minimize the power losses. The optimization procedures were, however, driven according to the results of the computer simulation. To provide a more realistic evaluation, we submitted the resulting designs to an expert designer to analyze them from the technical and production points of view. Here we compare the original engineering design, used as a starting point for the population-based methods, with two optimized designs.

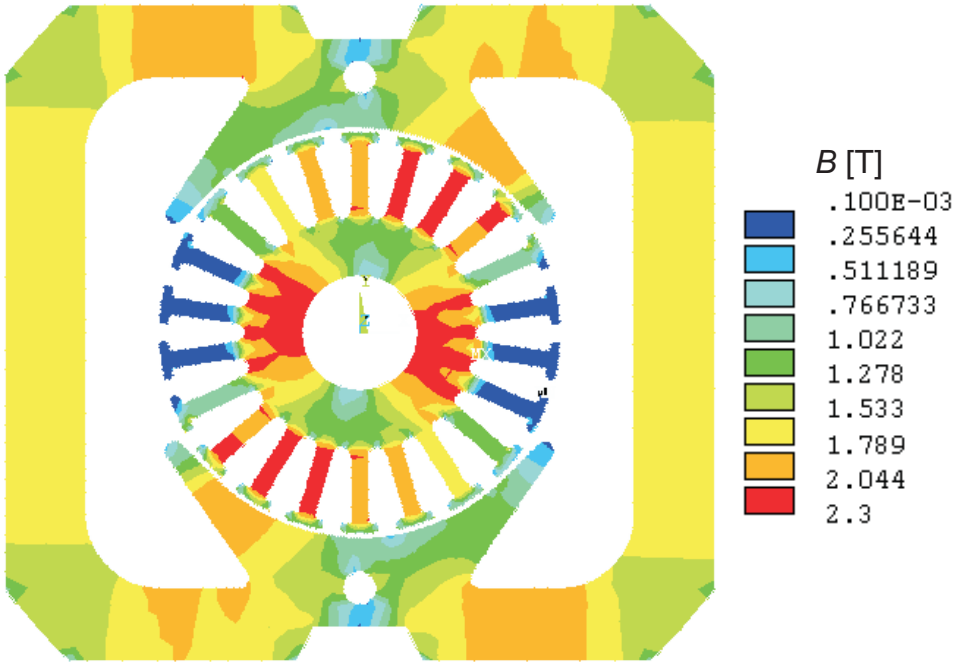

Figure 6.5 Laminations of the original engineering rotor and stator design with power losses of 177.9 W.

The engineering rotor and stator design results in power losses of 177.9 W, and can be seen in Figure 6.5. The figure shows the magnetic flux density in the laminations (higher magnetic flux density, B, causes higher power losses).

The best rotor and stator geometries obtained in the numerical optimization experiments generate power losses of 111.1 W. This was found by the MASA with LS, and is presented in Figure 6.6. These laminations have large rotor and stator slots, and therefore low copper losses and low overall power losses. The difficulty with this design is, however, the strange dimensions of the stator pole. Its narrow middle part makes

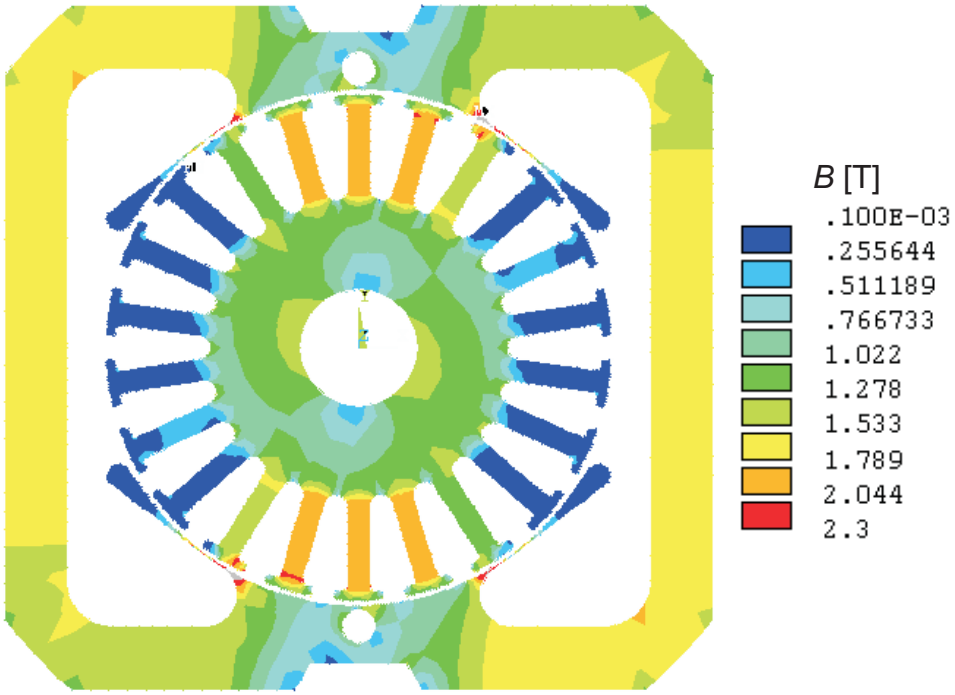

Figure 6.6 Laminations of the rotor and stator design with minimum power losses (111.1 W), as found in the optimization experiments.

it unacceptable for production. This outcome is due to the settings in the simulation script used, and it could be modified by inserting additional constraints on the stator geometry. Once they are specified, a new optimization cycle will be necessary. At this stage, however, the goal of our study is to check for possible improvements to the engineering UM design and compare the optimization methods on this problem.

Finally, we present a typical example of a feasible rotor and stator geometry (Figure 6.7). This was found by the MASA and the DE, with power losses of 129.1 W. This solution has very low iron losses in the rotor due to its small size and in spite of its high magnetic saturation. The small rotor and its saturation are compensated by large stator poles that ensure large enough a magnetic flux. This design is completely feasible from the technical and production points of view.

6.1.3 Optimization with the distributed MASA

With the distributed algorithm [115], all 11 mutually independent variable parameters defining the geometries of the rotor and the stator are subject to optimization (see Figure 6.2). The evaluated algorithm was the distributed MASA.

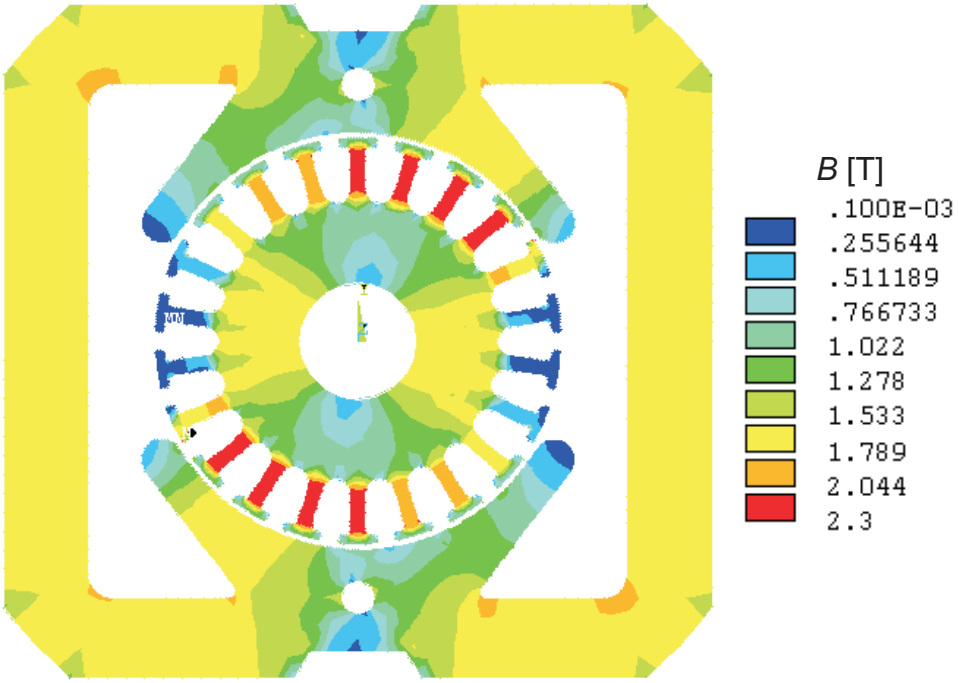

Figure 6.7 Laminations of a locally optimal rotor and stator design acceptable from the production point of view (power losses 129.1 W).

The computer platform used to perform the experiments was an 8-node cluster connected via a Giga-bit switch. Each node consisted of two AMD Opteron™ 1.8-GHz processors, 2 GB of RAM, and the Microsoft® Windows® XP operating system.

We implemented the MASA in Borland® Delphi™ using the TCP/IP protocol for communications between the server and the clients. The implementation also includes a graphical user interface to assist the user in selecting the appropriate parameters of the algorithm (see Figure 6.8).

The algorithm settings remained the same as with the sequential version. For different number of processors ($N = 1, 2, 4$ or 8) the algorithm was run 20 times, and the solution quality and the computational time were compared.

In the experiments, the stopping criterion for the MASA was given by the number of solutions to be evaluated. It was set to 2,240 evaluations per run (which takes approximately one day), and this value was chosen after considering the computational complexity of the optimization procedure. The other algorithm parameters were set as follows. The MASA operated with seven levels. At each level the ants in the sub-colony climb down the graph until the 'level ending condition' is not met (320 evaluations per

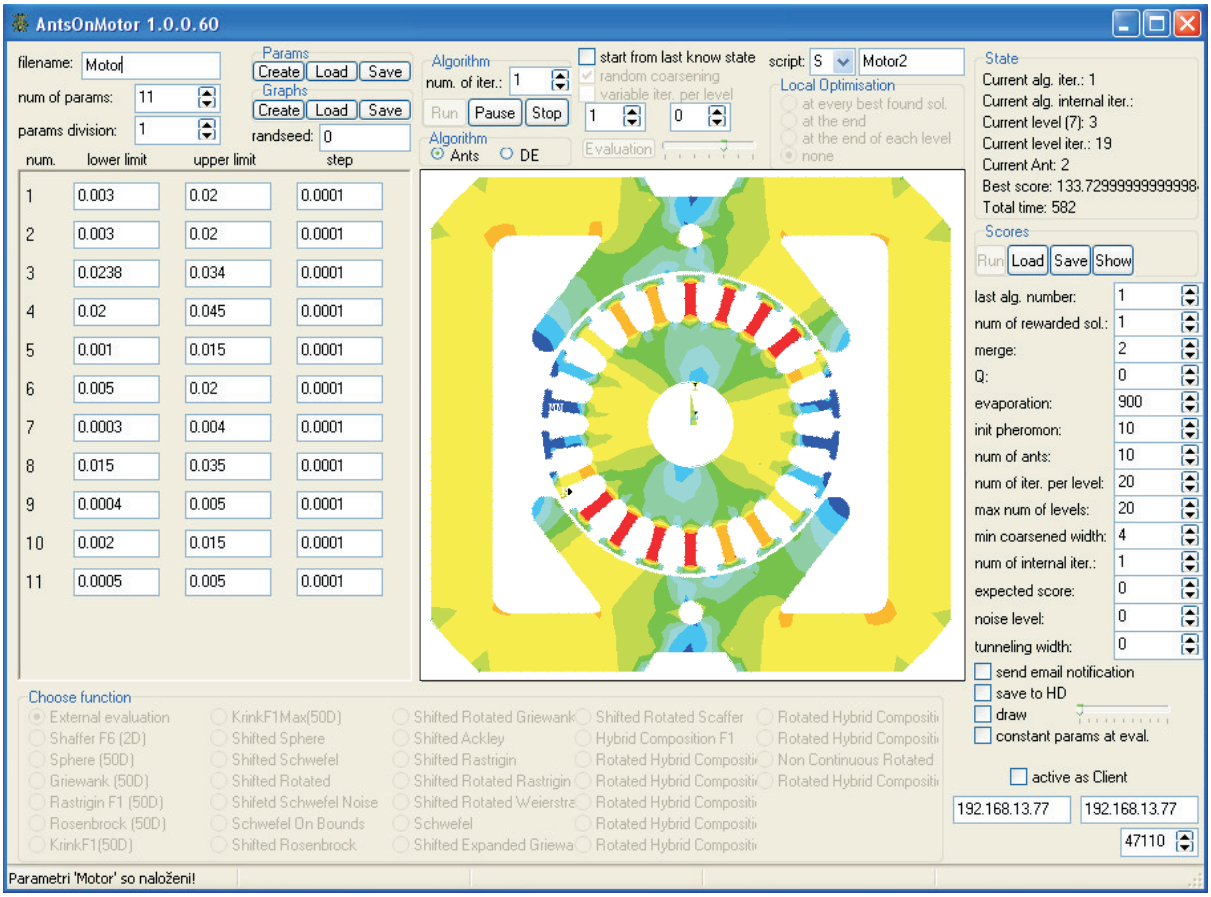

Figure 6.8 GUI used for setting the MASA options.

level in all sub-colonies together). The total number of ants in all the sub-colonies is 16, while the number of ants in each sub-colony depends on the number of processors, as shown in Table 6.4.

Table 6.4 Distribution of ants per sub-colonies according to the number of processors.

	Number of processors			
	1	2	4	8
Number of sub-colonies	1	2	4	8
Ants per sub-colony	16	8	4	2

In Table 6.5 it is clear that the MASA optimizes the UM design with power losses in the range from 120.63 W to 138.46 W. Even the worst solution is still much better than the one found by the expert designer (177.9 W), and which is used in production.

Table 6.6 shows that with distributed computing it is possible to drastically reduce the computation time. For $N = 1$ the average time needed to produce an optimized

Table 6.5 An UM design by the MASA (power losses in watts).

	Number of processors			
	1	2	4	8
Best	120.63	122.26	122.86	129.30
Worst	128.71	129.74	138.46	137.27
Mean	124.45	125.28	129.36	133.81
Std	2.33	2.83	4.85	2.63

solution is 24 hours and 42 minutes, while for $N = 8$ the time is reduced to 2 hours and 52 minutes.

Table 6.6 An UM design by the MASA (computation time in hh:mm).

	Number of processors			
	1	2	4	8
Best	21:35	10:24	4:14	2:08
Worst	25:52	13:21	6:36	3:18
Mean	24:42	11:43	5:33	2:52
Std	1:22	1:01	0:41	0:21

A more detailed examination of Table 6.7 reveals a slight degradation of the solution quality with an increasing number of processors. The main reason for this is the enormous number (approximately 99.98%) of infeasible solutions in the search space and the fact that the evaluation of a feasible solution takes 120 times longer than the evaluation of an infeasible solution.

Table 6.7 Speed up and quality degradation from mean values.

	Number of processors			
	1	2	4	8
Quality degradation [%]	0	0.67	3.95	7.52
Speed up	1	2.11	4.45	8.62

Let us consider the examples with one and eight processors. In the first case ($N = 1$) 16 ants climb down simultaneously, while in the second case ($N = 8$) 2 ants climb down simultaneously in each processor. If in the first case only one ant finds a feasible solution, the rest of the 15 ants wait for this ant to finish the solution evaluation and do not search for new, feasible solutions. But already in the next climb down all 16 ants

use this newly acquired information (gathered from the ant that made the previous climb down). So, at most 15 searches for a feasible solution were made without this information. Now, let us take into account the second case. If again only one ant finds a feasible solution, then just the ant on the same processor waits, but all the rest of the ants (14) search for new, feasible solutions undisturbed. Here it is possible that none of the 14 ants uses this information. For example, if in the first climb down of the current level one ant finds a feasible solution it takes approximately 120 seconds to evaluate it. In the mean time on all the other processors the climb downs continue and if no ant finds a feasible solution, all of the predetermined number of evaluations per level are done without using the knowledge gathered by the evaluation of the solution of the first ant. This scenario cannot happen in the first case (where $N = 1$), where at most 15 searches were lost.

In Table 6.7 we see that the increase in speed is greater than N, which confirms our statement from the previous paragraph, that the number of infeasible solutions increases with the number of processors.

6.1.4 Optimization with the sequential DASA

Like in Subsection 6.1.2, we also optimized only 10 parameters (the ratio between p_{10} and p_{11} was set constant) of the UM rotor and stator geometries. The DASA, like all the other tested algorithms, apart from the MASA, starts its search with the engineering solution. The stopping criterion was also set to $1{,}400$ evaluations. The DASA parameters were set as follows: the number of ants was set to 10, the pheromone evaporation factor, ρ, was set to 0.2, and the maximum parameter precision, ε, was set to 10^{-3} (the same as with the rest of the algorithms).

Table 6.8 Result statistics for the optimization methods (UM power losses in watts).

Method	Best	Avg	Worst	Std
MASA	114.2	128.9	135.9	7.8
DASA	113.8	129.5	136.7	7.6

The optimization method was run 20 times. The obtained results in terms of the UM power losses are presented statistically in Tables 6.8 and 6.9.

Table 6.9 Improvement of the results with local search (UM power losses in watts).

Method	Best w/o LS	Best with LS	Avg w/o LS	Avg with LS	Avg improv.	Addit. steps
MASA	114.2	111.1	128.9	126.2	2.7	116
DASA	113.8	112.1	129.5	129.0	0.5	40

The performance of the DASA is shown in Figure 6.9. We can see that the results are comparable. Nevertheless, here is the first time one can see that the MASA has a small advantage over the DASA. The MASA found the better solution than the DASA. In addition, on average the solutions obtained by MASA are also better.

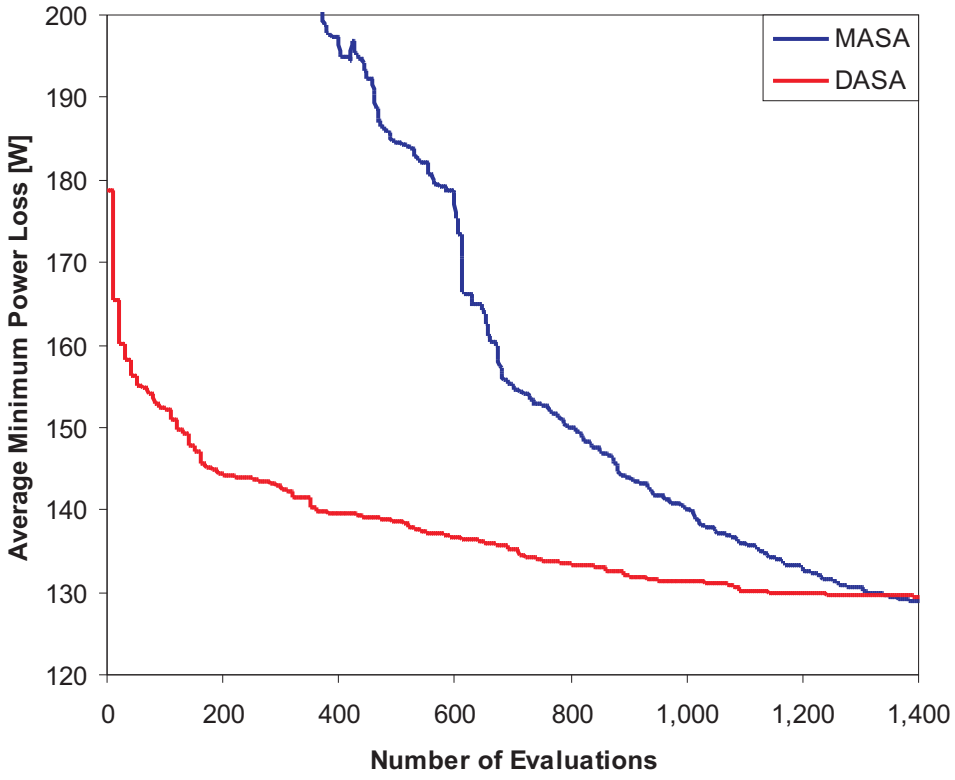

Figure 6.9 Performance of the DASA, averaged over 20 runs.

6.2 Optimizing the cooling process in continuous steel casting

Continuous casting is the dominant production technology in modern steel plants (see Figure 6.10). It is a complex metallurgical process in which liquid steel is cooled and shaped into semi-manufactures of the desired dimensions. To achieve the proper quality of the cast steel, it is essential to control the metal flow and heat transfer during the casting process. They depend on numerous parameters, such as the casting temperature, the casting speed and the coolant flows. Finding optimal values of the process parameters is difficult since different, often conflicting, criteria may be applied, the

Figure 6.10 An industrial continuous casting machine. Photo courtesy of the Rukki Steel, Finland.

number of possible parameter settings is high, and the parameter tuning through real-world experimentation is not feasible because of the costs and safety risks.

In this section we report on numerical experiments in optimizing secondary coolant flows on a casting machine of the Rukki Steel plant in Finland using a stigmergic algorithm on the Grid. Here we encounter multi-parameter optimization [69].

6.2.1 Definition of the problem

Figure 6.11 shows a schematic picture of a continuous casting machine. In the continuous casting process the molten steel is poured into a bottomless mold that is cooled with an internal water flow. The cooling in the mold extracts heat from the molten steel and initiates the formation of the solid shell. The shell formation is essential for the support of the slab after it exits the mold. After the mold the slab enters into the secondary cooling area in which it is cooled by water (mist) sprays. The secondary cooling region is divided into cooling zones, where the amounts of cooling water can be controlled separately.

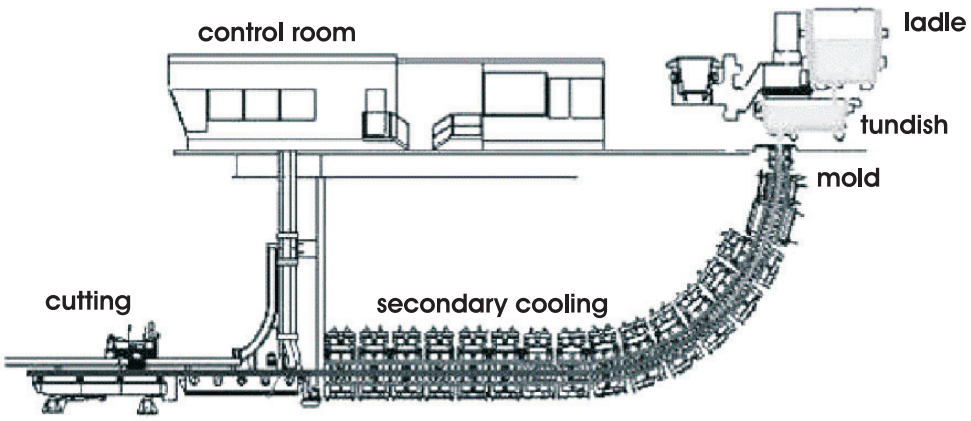

Figure 6.11 A schematic view of the continuous casting machine.

The secondary cooling area of the considered casting device is divided into nine zones. In each zone, cooling water is dispersed to the slab at the center and corner positions. Target temperatures are specified for the slab center and corner in every zone. The water flows should be tuned in such a way that the resulting slab-surface temperatures match the target temperatures. Formally, an objective function is introduced to measure the differences between the actual and target temperatures. It is defined as

$$c(T) = \frac{1}{2}\left(\sum_{i=1}^{N_\mathrm{Z}} l_i (T_i^{\mathrm{center}} - T_i^{\mathrm{center}*})^2 + \sum_{i=1}^{N_\mathrm{Z}} l_i (T_i^{\mathrm{corner}} - T_i^{\mathrm{corner}*})^2\right), \tag{6.5}$$

where N_Z denotes the number of zones, l_i is the length of the i-th zone, T_i^{center} and T_i^{corner} are the slab center and corner temperatures, while $T_i^{\mathrm{center}*}$ and $T_i^{\mathrm{corner}*}$ are the respective target temperatures in zone i. The optimization task is to minimize the objective function over possible cooling patterns (water flow settings). The water flows cannot be set arbitrarily, but according to the technological constraints. For each water flow, the minimum and maximum values are prescribed.

Table 6.10 The target temperatures and water flow intervals.

Pos.	Zone no.	T^* [°C]	Parameter	$p_i^{\min}$ [m^3/h]	$p_i^{\max}$ [m^3/h]
	1	1,050	p_1	7.1	26.1
	2	1,040	p_2	22.8	57.5
c	3	980	p_3	13.3	39.9
e	4	970	p_4	1.5	7.9
n	5	960	p_5	2.7	10.0
t	6	950	p_6	0.8	6.5
e	7	940	p_7	0.7	5.9
r	8	930	p_8	1.0	5.8
	9	920	p_9	1.2	6.2
	1	880	p_{10}	7.1	26.1
	2	870	p_{11}	22.8	57.5
c	3	810	p_{12}	13.3	39.9
o	4	800	p_{13}	1.2	3.5
r	5	790	p_{14}	2.4	4.4
n	6	780	p_{15}	2.4	2.9
e	7	770	p_{16}	0.7	5.9
r	8	760	p_{17}	1.0	5.8
	9	750	p_{18}	1.2	6.2

Table 6.10 shows an example of the prescribed target temperatures and water flow intervals (defined at the Rukki Steel, Raahe, Finland) for continuous casting of the selected steel grade analyzed in this study. The slab cross-section in this case was 1.70 m $\times$ 0.21 m and the casting speed 0.23 m/s.

The evaluation of the cooling patterns and their assessment with respect to the objective function (Eq. 6.5) was done by means of a numerical simulator [81]. Its principal task is to dynamically track the temperature field in the slab as a function of the

process parameters. It involves a 3D model of the slab and a finite-element numerical approximation. In this study it was applied under the assumption of steady-state caster operation, and the search for optimal cooling patterns performed in the off-line manner. A single simulator run takes about 25 seconds on a 1.8-GHz AMD Opteron™ computer.

6.2.2 Optimization with the sequential MASA

Due to the fact that the sequential MASA equals (with the settings and results) the distributed MASA when there is only one client used, we decided to merge them in the next subsection.

6.2.3 Optimization with the distributed MASA

It is informative to calculate the number of possible parameter settings (see Table 6.11). For a parameter p_i from the interval $[p_i^{\min}, p_i^{\max}]$ with step size p_i^{step}, there are

$$s_i = \left\lceil \frac{p_i^{\max} - p_i^{\min}}{p_i^{\text{step}}} \right\rceil + 1 \tag{6.6}$$

Table 6.11 Discrete parameter values used in the coolant flows optimization.

Pos.	Zone no.	Parameter	p_i^{step} [m^3/h]	s_i
	1	p_1	0.1	191
	2	p_2	0.1	348
c	3	p_3	0.1	267
e	4	p_4	0.1	65
n	5	p_5	0.1	74
t	6	p_6	0.1	58
e	7	p_7	0.1	53
r	8	p_8	0.1	49
	9	p_9	0.1	51
	1	p_{10}	0.1	191
	2	p_{11}	0.1	348
c	3	p_{12}	0.1	267
o	4	p_{13}	0.1	24
r	5	p_{14}	0.1	21
n	6	p_{15}	0.1	6
e	7	p_{16}	0.1	53
r	8	p_{17}	0.1	49
	9	p_{18}	0.1	51

values possible, and the total number of settings is

$$s = \prod_{i=1}^{D} s_i, \tag{6.7}$$

where D is the number of parameters. This results in $s = 4.7 \cdot 10^{33}$ for step size $p_i^{\text{step}} = 0.1$ m^3/h.

The computer platform used to perform the experiments was an 8-node Grid connected via a Giga-bit switch, each node consisted of two AMD Opteron™ 1.8-GHz processors and 2 GB of RAM. In our case, we decided to use a homogeneous Grid, so we could estimate the speed up. But this is not a requirement for real-world applications. The clients and the server used a standard TCP/IP protocol to communicate between each other.

In the experiments, the stopping criterion for the DMASA was given by the number of solutions to be evaluated. It was set to 768 evaluations per run (which takes approximately 10 hours), and this value was chosen after considering the computational complexity of the optimization procedure. Other algorithm parameters were set as follows. The DMASA operated with eight levels, at each level the ants in the sub-colony climb down the graph until the "level ending condition" is not met (96 evaluations per level in all sub-colonies together). The total number of ants in all sub-colonies is 8, while the number of ants in each sub-colony depends on the number of nodes in the Grid, as shown in Table 6.12.

Table 6.12 Distribution of ants over the processors.

	Number of nodes in the Grid			
	1	2	4	8
Number of sub-colonies	1	2	4	8
Ants per sub-colony	8	4	2	1

For different number of processors ($N = 1, 2, 4$ or 8), we ran our algorithm 20 times and compared solution quality and computational time.

In Table 6.13 we show the best solution found [32]. This is the solution we were able to obtain with every run, regardless of the number of processors used. This is why we

Table 6.13 The optimized temperatures and parameter values with best known solution $c(T')$ in coolant flows optimization.

Pos.	Zone no.	T' [°C]	Parameter	p'_i [m^3/h]
	1	1,048.5	p_1	17.2
	2	1,039.3	p_2	33.0
c	3	978.3	p_3	20.9
e	4	977.6	p_4	7.9
n	5	960.3	p_5	9.2
t	6	950.0	p_6	5.8
e	7	940.0	p_7	3.1
r	8	930.1	p_8	3.2
	9	919.8	p_9	2.1
	1	880.5	p_{10}	8.4
	2	863.3	p_{11}	22.8
c	3	803.5	p_{12}	15.6
o	4	836.0	p_{13}	3.5
r	5	804.7	p_{14}	4.4
n	6	783.0	p_{15}	2.8
e	7	773.8	p_{16}	1.3
r	8	760.9	p_{17}	1.0
	9	720.6	p_{18}	1.2
$c(T') = 9{,}070.4$				

concentrated only on the time needed to compute the solution and not on the solution quality.

Table 6.14 Computation time (in hh:mm) needed by the DMASA in coolant flows optimization.

	Number of nodes in the Grid			
	1	2	4	8
Min	8:43	5:06	4:11	3:02
Max	10:25	7:56	6:36	5:15
Avg	9:38	6:30	5:02	3:58
Std	0:33	0:41	0:34	0:39

Table 6.14 shows the computational times needed by the DMASA on the different number of nodes in the Grid. It is clear that we managed to decrease the time quite

noticeably, from almost ten hours on one node to under four hours on a Grid with eight nodes. The standard deviation remained almost the same throughout the experiments.

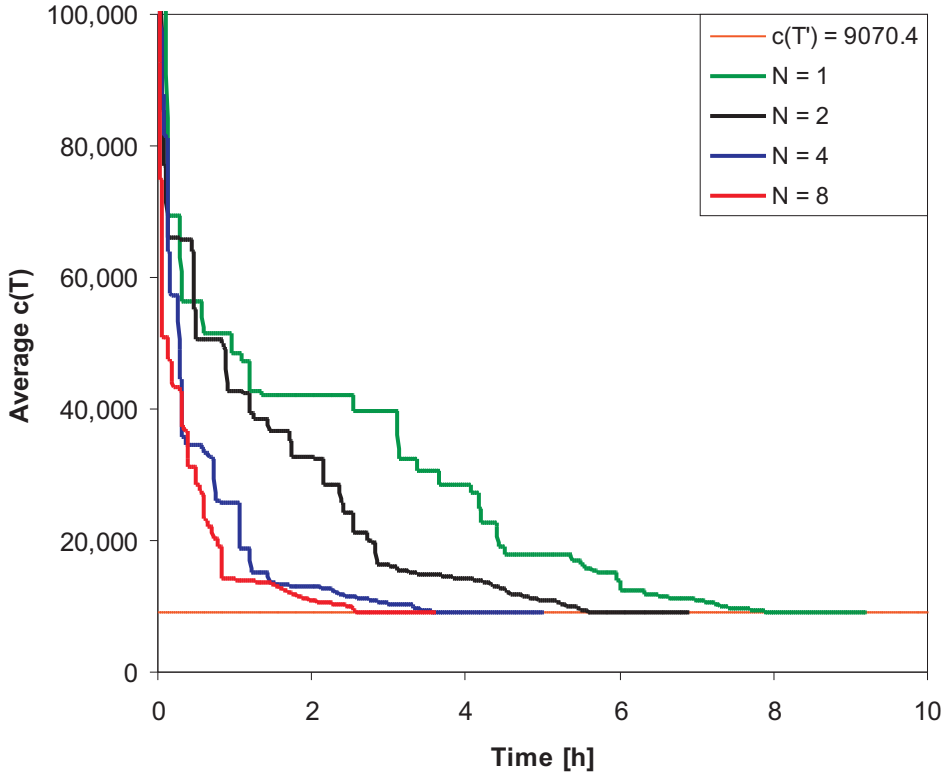

Figure 6.12 Convergence of the DMASA at different levels of distribution on the cooling-process optimization problem.

Figures 6.12 and 6.13 show the convergence and speed up of the DMASA achieved for a different number of nodes. We can see that the convergence is improved with a larger number of nodes and that the speed up is not linear, but still quite evident.

6.2.4 Optimization with the sequential DASA

Optimization of coolant flows in steel continuous casting is another real-world problem where we decided to test the DASA. In comparison with the previous subsections we decided to extend the parameter constraints (all are set from 0 to 100) because they were obviously preventing the algorithms from exploiting all the possibilities. For this reason we ran the DASA in continuous mode (ε was set to 10^{-12}) and compared it to the DE and the MASA. The accuracy of the MASA was set to maximum possible regarding

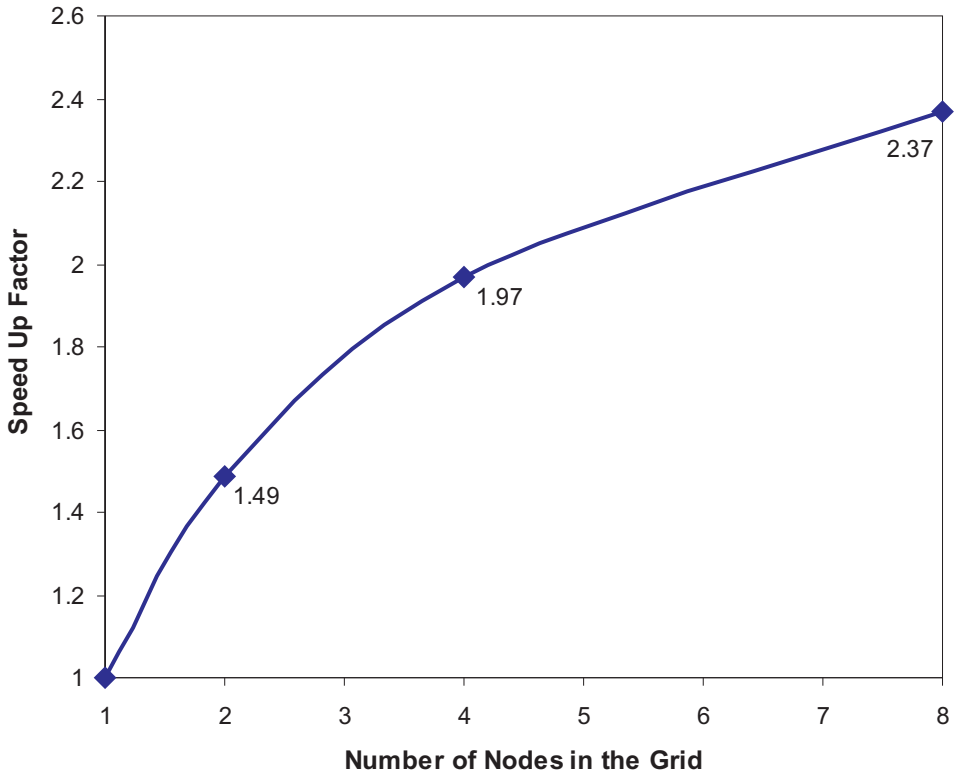

Figure 6.13 Speed up of the DMASA on the cooling-process optimization problem.

the computer's RAM (in our case 10^{-4}). The settings for the algorithms were as follows: for the DASA the number of ants was set to 10 and the pheromone evaporation factor, ρ, was set to 0.2. For the DE the population size was set to 20, the crossover probability was set to 0.3, and the scaling factor was set to 0.5. The MASA operated with 25 levels, at each level the ants in the colony climb down the graph until the "level ending condition" is not met (250 evaluations per level). The number of ants was set to 10, the pheromone evaporation factor, ρ, was set to 0.75, and the coarsening was implemented by merging two vertices into one.

Because of a much larger search space we decided to set the stopping criterion, for all the optimization methods, at $5{,}000$ evaluations. Due to a priori chosen number of evaluations, we were forced to omit the LS from the MASA. We ran the algorithms 20 times, where we compared the solution quality and the algorithm convergency.

The obtained results in terms of the deviation from the target temperatures are presented statistically in Table 6.15.

Table 6.15 Result statistics for the optimization methods (deviation from the target temperatures).

Method	Best	Avg	Worst	Std
MASA	1,677.6	6,616.0	15,303.14	3,941.0
DE	0.1241	0.5670	2.2197	0.4465
DASA	0.0021	0.0136	0.0493	0.0129

The convergence of the DASA, the DE, and the MASA is shown in Figure 6.14.

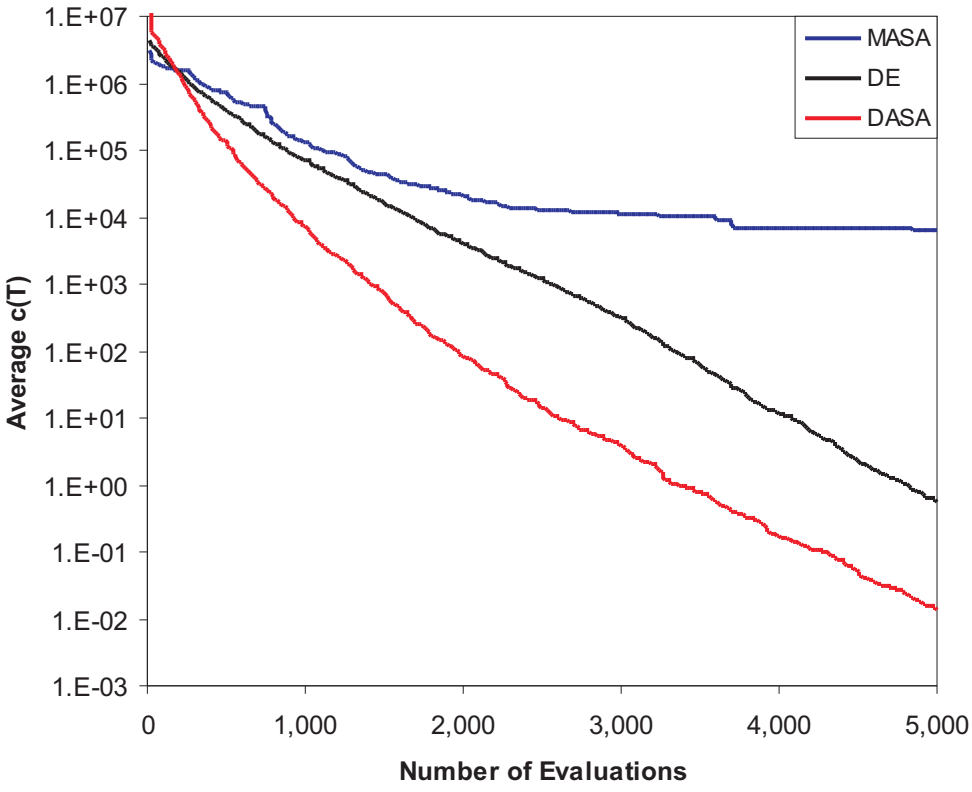

Figure 6.14 Convergence of the DASA, the DE, and the MASA on the cooling-optimization problem, averaged over 20 runs.

From Table 6.15 and Figure 6.14 it is clear that the DASA is superior to both the DE and the MASA, in terms of convergence and solution quality.

7 Conclusion and discussion

Ant colonies have already been used to solve various combinatorial optimization problems, such as the traveling salesman, quadratic assignment, job-shop scheduling, vehicle and network routing. But only a few attempts have been made when it comes to numerical optimization, for example, the Extended ACO for continuous and mixed-variable problems.

The multiple ant-colonies approach

As shown in Chapter 3, the ant-based method called the Multiple Ant-Colony Algorithm (MACA) can be successfully used to solve an example of the clustering problem called the mesh-partitioning problem, which is also one of the hard combinatorial optimization problems. We investigated variants of the MACA for mesh partitioning to suggest improvements to this algorithm and to evaluate them experimentally. The basic MACA performed very well on small- or medium-sized graphs. With larger graphs, which are often encountered in mesh partitioning, we had to use a multilevel or hybrid method to produce results that were competitive with the results obtained with other algorithms. Both the multilevel and hybrid MACA performed very well on almost all graphs. Both were quite similar in producing the best results. The only difference was in the standard deviation of the results [76], which was in favor of the hybrid method.

The multilevel ant-stigmergy approach

In Chapter 4 we presented a novel method based on stigmergy for solving discrete numerical (multi-parameter) optimization problems.

We proposed a general approach for the transformation of a multi-parameter problem into a search-graph representation. Here, each longest path in the search graph represents one possible solution, and all the longest paths together represent the whole solution space of the multi-parameter problem. For an efficient search of the solution space we used the multilevel approach. We call this method the Multilevel Ant-Stigmergy Algorithm (MASA).

We evaluated the performance of the MASA compared to the Differential Evolution (DE) in terms of their applicability as numerical optimization techniques. The comparison was performed with several widely used benchmark functions. It was found that for low-dimensional functions the performance was comparable, while for higher dimensions the MASA outperformed the DE in all functions with the exception of one.

Since the optimization of noisy functions is a common problem occurring in various applications we also evaluated the MASA compared to the DE on noisy benchmark functions. For evaluation purposes we used three different degrees of sampling: low, medium, and high. We observed that the impact of the higher degree on the performance of the DE is greater than for the MASA.

The differential ant-stigmergy approach

Finally, we proposed an extension of the ant-colony optimization metaphor for the continuous domain presented in Chapter 5. This new approach was named the Differential Ant-Stigmergy Algorithm (DASA) and was studied on a set of new multi-parameter benchmark functions defined in 2005.

The algorithm was compared with a number of evolutionary optimization algorithms including the Covariance Matrix Adaptation Evolutionary Strategy, the DE, the Real-Coded Memetic Algorithm, and the Continuous Estimation of Distribution Algorithm.

The results obtained indicate a promising performance for the new approach. One can see that it performs better than the rest of the approaches on three out of four test functions. Since the selected test functions reflect different kinds of pseudo-real optimization problems, one can expect that the DASA is applicable to many real-world

multi-parameter optimization problems.

Real-world applications

To better understand how the newly proposed algorithms work on real-world problems, we tested them on two different examples.

In the first real-world problem, we optimized the power losses of a universal electric motor for Domel – Electromotors and Household Devices, Železniki, Slovenia. The goal of the optimization was to find the geometrical parameter values that would generate the rotor and the stator geometries that produce the minimum power losses.

We compared the MASA to four stochastic optimization methods in the computer-assisted design of the rotor and stator geometries. The primary design goal was to minimize the power losses, however, the resulting designs were also evaluated by an expert designer from the point of view of feasibility for use in regular production. The compared methods were the Generational Evolutionary Algorithm, the Steady-State Evolutionary Algorithm, the DE, and the Electromagnetism-like Algorithm. They were applied to optimize the geometry parameters of a universal electric motor already in regular production. The optimization procedures were navigated by a numerical evaluation of the candidate solutions.

The output of this comparison can be summarized in several important findings. Principally, all the tested optimization methods were able to significantly improve on the original engineering design. Among the tested methods, the MASA generated the minimum power-loss designs. Its additional advantage, revealed on this problem, was the capability to successfully perform the optimization from random starting points, which was not the case with the other methods. In our opinion, the superiority of the MASA arises from its multilevel search feature.

The analysis of the optimized universal electric rotor and stator geometries by an expert designer revealed some deficiencies in the designs with very low power losses. Although perfect with respect to the primary design goal, they turned out to be infeasible for regular production because of the limitations of the manufacturing process.

This result calls for a refinement of the optimization script with additional constraints and a new optimization cycle.

Due to the time-consuming solution the evaluation of the sequential version of the algorithm is inefficient from the point of view of time. This led us to the distributed version of the MASA. The experimental results show that with distributed computing the computation time can be drastically decreased (from one day to a few hours) without any noticeable decrease in the quality of the solution.

Even though the quality of the solutions decreases slightly when more processors are used, the solutions obtained were still better than the ones found using some other sequential stochastic algorithms, and considerably better than the one manually designed by expert engineers.

Since the DASA is also capable of solving a discrete numerical optimization we run it on this problem too. Surprisingly, it was capable of finding best solutions in the same range as the MASA, even though it started from experts' solution like all the other tested algorithms. Even though, on average, the results were a little worse than the MASA's. This result further confirmed our belief that this algorithm has great potential.

The second real-world optimization problem was the optimization of the coolant flow settings for the continuous casting of steel at the Rukki Steel, Raahe, Finland. Proper cooling is the key to higher product quality. Optimization of cooling is nowadays mainly performed through virtual experimentation involving numerical process simulators and advanced optimization techniques.

In this study, algorithms for optimizing 18 cooling water flows for a casting machine operating under steady-state conditions were evaluated. Due to parameter constraints imposed by the steel plant, the MASA always found currently best known solution. Still, the time needed was very long. So again there was a requirement for a distributed MASA. The results indicate the importance of the applied discrete search space [32] and suggest the construction of a hybrid algorithm to find near-optimal solutions in a smaller number of solution evaluations.

We have shown that with distributed computing we can drastically decrease the computation time (from half of a day to a few hours) without any decrease in the quality of solutions.

Here, we also tested the DASA. But this time we relaxed the parameter constraints (so the algorithm could not reach their boundaries, like previously with MASA, and as a result finish the search too early) and let it run in continuous mode. The DASA was compared to the DE and the MASA. The dominance of the DASA was shown again. It outperformed the DE and the MASA in terms of the convergence and the minimum solution found.

Aspects of the algorithms

One of the important questions that needs to be answered in this dissertation is:

"Which algorithm—the MASA or the DASA—should we choose for a particular problem?"

Let us look at the advantages and disadvantages of each algorithm.

The MASA is an algorithm that is bound to problems that can be put into discrete form, which is an important limitation we have to be aware of. Even if a problem can be put into discrete form (i.e., search graph), one has to be careful about the size of the search graph. In the case of very large search graphs, one has to check if there is enough memory for it. So, the accuracy that can be achieved with the algorithm is limited by the memory capacity of the computer. Next, we noticed that the complexity of the algorithm is much higher than the DASA's. This is no handicap, if the evaluation itself is very time consuming, but if the evaluation is a simple function, then this could become a real obstacle. The last disadvantage that we see is in the use of the multilevel approach, which somehow limits the convergence. Because the algorithm has to go through all the levels it might happen that the algorithm stagnates for some time at a level before moving to the next one. As a consequence, the calculations/evaluations are spent unnecessarily. But, on the other hand, the multilevel approach gives this algorithm an edge. As presented in the problem of optimizing the electric-motor power

losses, one can see that with the use of the multilevel approach we were able to find good solutions, even in a search space that has very low proportion of feasible solutions. With the MASA we were able to find solutions that could not be obtained with the other tested algorithms.

On the other hand, for the DASA we observed that the majority of the MASA's limitations disappear. Most importantly, there is no need to put a problem into discrete form. There are no more accuracy problems, because the accuracy is now limited by the computer hardware (floating-point accuracy) and not the size of the search graph. The algorithm's complexity is lower than the MASA's. A question arises, if there are any disadvantages when using the DASA? An example that was shown in our experiments is the starting-point limitation. When we use the DASA on a search space that has low proportion of feasible solutions, it has the same problems as the rest of the tested algorithms, with the exception of the MASA. The DASA has to start from an already-known solution, and, as a result, it is limited to the search subspace in which the known solution resides. This can be considered a disadvantage compared to the MASA.

Now, we are able to answer the question posed at the beginning of this subsection.

"For the problems with search space, which has low proportion of feasible solutions, we recommend using the MASA; for all other problems we would suggest using the DASA."

Of course, the experiments made in this dissertation do not cover all possible problems, but they do give us some indications about how each of the algorithms work under certain conditions.

7.1 Contributions to science

The main contributions to science are the two novel approaches to solving hard combinatorial and numerical optimization problems that are extended from the ant-colony optimization method. Inside the newly developed optimization methods there are the following specific contributions.

In the first approach we have introduced a new way of transforming a multi-parameter optimization problem into a problem of finding the cheapest path. The

multilevel approach was successfully used for solving numerical optimization problems. Also, a distributed version of the optimization method was realized in a multiprocessor environment.

In the second approach we have introduced another way of representing a multiparameter optimization problem as a graph, which allowed us to solve continuous numerical optimization problems.

We implemented both numerical optimization methods, the MASA and the DASA, and used them on artificial and real-world optimization problems, which gave improved results.

7.2 Future work

Like almost every metaheuristic algorithm, the MASA and DASA have some parameters that need to be tuned. During the experimentation we did not fine-tune the algorithms parameters (but only make a limited number of experiments to find satisfying settings), so there is a definite need to make a study of how different parameter settings can influence the algorithm.

The used benchmark functions and two real-world problems are not enough to conclude which algorithm is better suited for a given type of problems. Therefore, additional experiments are needed.

As we have seen the DASA performed very well on most of the problems that we tried it on. So the obvious way to go is a further research/development of this algorithm and extending it with: a distributed version for faster problem solving, a multilevel approach for better search space exploration, and multi-objective optimization capabilities.

Acknowledgements

I am grateful to a number of people and organizations that have supported me during the course of this work. First and foremost, my heartily profound thanks, gratitude and appreciation are addressed to my PhD superviser, Assist. Prof. Dr. Bogdan Filipič, and my co-superviser, Assist. Prof. Dr. Jurij Šilc, for their encouragement, help and kind support. Their invaluable technical and editorial advice, suggestions, discussions and guidance were a real support when completing this dissertation.

Furthermore, I would like to express my deepest gratitude and thanks to all the staff members of the Computer Systems Department at the Jožef Stefan Institute for contributing to such an inspiring and pleasant atmosphere.

I also thank the Slovenian Research Agency for the financial support that was contributed by the national research program for young researchers.

I thank my parents for laying the foundations for this work. And I want to thank my Tina, for all the support that she has given me during the years I have been working on this dissertation.

And lastly to Dr. Paul McGuiness for proof-reading this dissertation.

References

[1] *ANSYS User's Manual, ANSYS version 5.6*. ANSYS Inc. Canonsburg, PA, 2000.

[2] Arnett, R. H. *American Insects: A Handbook of the Insects of America North of Mexico*. Van Nostrand Rehinhold, New York, 1985.

[3] Auger, A., and Hansen, N. A restart CMA evolution strategy with increasing population size. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (Edinburg, UK, September 2-5 2005).

[4] Bahreininejad, A., Topping, B. H. V., and Khan, A. I. Finite element mesh partitioning using neural networks. *Advances in Engineering Software 27*, 1-2 (1996), 103–115.

[5] Baños, R., Gil, C., Ortega, J., and Montoya, F. G. Multilevel heuristic algorithm for graph partitioning. In *Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM* (University of Essex, England, UK, April 14-16 2003), G. R. Raidl, S. Cagnoni, J. J. R. Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, E. Marchiori, J.-A. Meyer, and M. Middendorf, Eds., vol. 2611 of *Lecture Notes in Computer Science*, EvoNet, Springer-Verlag, pp. 143–153.

[6] Barnard, S. T., and Simon, H. D. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience 6*, 2 (April 1994), 101–117.

[7] Belal, M., Gaber, J., El-Sayed, H., and Almojel, A. Swarm intelligence. In *Handbook of bioinspired algorithms and applications* (Boca Raton, London, New York, 2006), S. Olariu and A. Y. Zomaya, Eds., Computer and information science series, Chapman & Hall/CRC, pp. 55–63.

[8] Bilchev, G., and Parmee, I. C. The ant colony metaphor for searching continuous design spaces. In *Evolutionary Computing, AISB Workshop* (Sheeld, UK, April 3-4 1995), T. C. Fogarty, Ed., vol. 993 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 25–39.

[9] Birbil, Ş. I., and Fang, S.-C. An electromagnetism-like mechanism for global optimization. *Journal of Global Optimization 25*, 3 (2003), 263–282.

[10] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys 35*, 3 (September 2003), 268–308.

[11] Bonabeau, E., Dorigo, M., and Theraulaz, G. *Swarm Intelligence*. Oxford University Press, Oxford, 1999.

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

[13] Caro, G. D., and Dorigo, M. Antnet: Distributed stigmergic control for communications networks. *Journal of Artificial Intelligence Research 9* (1998), 317–365.

[14] Chen, L., Shen, J., Qin, L., and Fan, J. A method for solving optimization problem in continuous space using improved ant colony algorithm. In *Data Mining and Knowledge Management: Chinese Academy of Sciences Symposium CASDMKM 2004* (Beijing, China, July 12-14 2004), Y. Shi, W. Xu, and Z. Chen, Eds., vol. 3327 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 61–70.

[15] Colorni, A., Dorigo, M., and Maniezzo, V. Distributed optimization by ant colonies. In *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life (ECAL)* (Paris, France, December 11-13 1991), F. J. Varela and P. Bourgine, Eds., MIT Press, Cambridge, MA, pp. 134–142.

[16] Cook, R. D., Malkus, D. S., Plesha, M. E., and Witt, R. J. *Concepts and Applications of Finite Element Analysis*, 4th ed. John Wiley & Sons, October 2001.

[17] Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A. *Combinatorial Optimization*. John Wiley & Sons, New York, 1997.

[18] Corne, D., Dorigo, M., and Glover, F. *New Ideas in Optimization*. McGraw-Hill, New York, 1999.

[19] Deb, K., Anand, A., and Joshi, D. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation 10*, 4 (December 2002), 371–395.

[20] Denby, B., and Le Hégarat-Mascle, S. Swarm intelligence for optimisation problems. *Nuclear Instruments and Methods 502*, 2-3 (April 21 2003), 364–368.

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

[22] Dorigo, M. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992. In Italian.

[23] Dorigo, M., Bonabeau, E., and Theraulaz, G. Ant algorithms and stigmergy. *Future Generation Computer Systems 16*, 9 (June 2000), 851–871.

[24] Dorigo, M., and Gambardella, L. M. A study of some properties of ant-q. Tech. Rep. IRIDIA-1996-4, Université Libre de Bruxelles, Belgium, 1996.

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

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

[27] Dorigo, M., and Stützle, T. *Ant Colony Optimization*. The MIT Press, Cambridge, Massachusetts, July 2004.

[28] Dréo, J., and Siarry, P. A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In *Ant Algorithms, Third International Workshop, ANTS 2002* (Brussels, Belgium, September 12-14 2002), M. Dorigo, G. D. Caro, and M. Sampels, Eds., vol. 2463 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 216–227.

[29] Farhat, C., and Lesoinne, M. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *International Journal for Numerical Methods in Engineering 36*, 5 (March 1993), 745–764.

[30] Feo, T. A., and Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6* (1995), 109–133.

[31] Fiduccia, C. M., and Mattheyses, R. M. A linear time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference* (Las Vegas, NV, 1982), IEEE Press, Piscataway, NJ, pp. 175–181.

[32] Filipič, B., and Laitinen, E. Model-based tuning of process parameters for steady-state steel casting. *Informatica 29*, 4 (November 2005), 491–496.

[33] Flynn, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computing 21*, 9 (September 1972), 948–960.

[34] Fogel, L. J., Owens, A. J., and Walsh, M. J. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

[35] Gambardella, L. M., and Dorigo, M. Ant-q: a reinforcement learning approach to the travelling salesman problem. In *Proceedings of the Twelfth International Conference on Machine Learning, ML-95* (Palo Alto, CA, 1995), Morgan Kaufmann, pp. 252–260.

[36] Gambardella, L. M., and Dorigo, M. HAS-SOP: Hybrid ant system for the sequential ordering problem. Tech. Rep. IDSIA-11-97, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Manno-Lugano, Switzerland, January 1997.

[37] Gambardella, L. M., and Dorigo, M. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing 12*, 3 (July 2000), 237–255.

[38] Glover, F. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research 13*, 5 (1986), 533–549.

[39] Glover, F., and Laguna, M. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

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

[41] Goss, S., Aron, S., Deneubourg, J.-L., and Pasteels, J. M. Self-organized shortcuts in the Argentine ants. *Naturwissenschaften 76*, 12 (1989), 579–581.

[42] Grassé, P.-P. La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes* sp. La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux 6*, 41–81 (1959). In French.

[43] Guntsch, M., and Branke, J. Ant colony optimization. In *Handbook of bioinspired algorithms and applications* (Boca Raton, London, New York, 2006), S. Olariu and A. Y. Zomaya, Eds., Computer and information science series, Chapman & Hall/CRC, pp. 41–54.

[44] Guntsch, M., and Middendorf, M. A population based approach for ACO. In *Applications of Evolutionary Computing, EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN* (Kinsale, Ireland, April 3-4 2002), S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, Eds., vol. 2279 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 72–81.

[45] Gutjahr, W. J. A graph-based ant system and its convergence. *Future Generation Computer Systems 16*, 8 (2000), 873–888.

[46] Gutjahr, W. J. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters 82*, 3 (2002), 145–153.

[47] Hadeli, K. P., Valckenaers, P., Kollingbaum, M., and Van Brussel, H. Multi-agent coordination and control using stigmergy. *Computers in Industry 53*, 1 (January 2004), 75–96.

[48] Handl, J., and Meyer, B. Improved ant-based clustering and sorting in a document retrieval interface. In *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN VII)* (Granada, Spain, 2002), J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J. L. Fernández-Villacañas Martn, and H.-P. Schwefel, Eds., vol. 2439 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 913–923.

[49] Hansen, N., and Ostermeier, A. Adapting arbitrary normal mutation distribution in evolution strategies: The covariance matrix adaptation. In *Proceedings of 1996 IEEE International Conference on Evolution Computation (ICEC '96)* (Nagoya, Japan, May 20-22 1996), IEEE, pp. 312–317.

[50] Hendrickson, B., and Leland, R. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing* (San Diego, CA, December 1995), p. 28.

[51] Holland, J. H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[52] Holland, O., and Melhuish, C. Stimergy, self-organization, and sorting in collective robotics. *Artificial Life 5*, 2 (1999), 173–202.

[53] Kadłuczka, P., and Wala, K. Tabu search and genetic algorithms for the generalized graph partitioning problem. *Control and Cybernetics 24*, 4 (1995), 459–476.

[54] Karlson, P., and Lüscher, M. Pheromones: a new term for a class of biologically active substances. *Nature 183*, 4653 (January 1959), 55–56.

[55] Karypis, G., and Kumar, V. Analysis of multilevel graph partitioning. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing* (San Diego, CA, December 1995), p. 29.

[56] Karypis, G., and Kumar, V. Multilevel k-way partioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing 48*, 1 (1998), 96–129.

[57] Kennedy, J., and Eberhart, R. C. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks* (Perth, Australia, December 1995), vol. IV, IEEE Service Center, Piscataway, NJ, pp. 1942–1948.

[58] Kirkpatrick, S., Gelatt, C., and Vecchi, M. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[59] Korošec, P. Evaluation of heuristic partitioning algorithms. Undergraduate Thesis, Faculty of Computer and Information Science, University of Ljubljana, Slovenia, September 2001. In Slovenian.

[60] Korošec, P. Metaheuristic solving of the optimization problem with ant colonies. Master's thesis, Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia, September 2004. In Slovenian.

[61] Korošec, P., Papa, G., and Šilc, J. Optimization algorithms inspired by electromagnetism and stigmergy in electro-technical engineering. *WSEAS Transactions on Information Science and Applications 5*, 2 (May 2005), 587–591.

[62] Korošec, P., Robič, B., and Šilc, J. Algorithms for solving the partitioning problem. In *Zbornik desete Elektrotehniške in računalniške konference (ERK 2001)* (Portorož, Slovenia, September 24-26 2001), B. Zajc, Ed., vol. B, Slovenska sekcija IEEE, pp. 23–26. In Slovenian.

[63] Korošec, P., and Šilc, J. Multilevel optimization of graph bisection with pheromonoes. In *Proceedings of the International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)* (Ljubljana, Slovenia, October 11-12 2004), B. Filipič and J. Šilc, Eds., Jožef Stefan Institute, pp. 73–80.

[64] Korošec, P., and Šilc, J. The multilevel ant stigmergy algorithm: An industrial case study. In *Proceedings of the 8th Joint Conference on Information Sciences* (Salt Lake City, UT, July 21-25 2005), pp. 475–478.

[65] Korošec, P., and Šilc, J. The multilevel stigmergic algorithm for numerical optimization. In *Zbornik štirinajste Elektrotehniške in računalniške konference (ERK 2005)* (Portorož, Slovenia, September 26-28 2005), B. Zajc and A. Trost, Eds., vol. B, Slovenska sekcija IEEE, pp. 49–52. In Slovenian.

[66] Korošec, P., and Šilc, J. A performance comparison of ant stigmergy and differential evolution for numerical optimization. In *Proceedings of the 7th International Conference on Artificial Evolution (EA 2005)* (Lille, France, October 26-28 2005).

[67] Korošec, P., and Šilc, J. The multilevel ant stigmergy algorithm for numerical optimization. *Facta Universitatis. Series Electronics and Energetics 19*, 2 (August 2006), 247–260.

[68] Korošec, P., and Šilc, J. Real-parameter optimization using stigmergy. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006)* (Ljubljana, Slovenia, October 9-10 2006), B. Filipič and J. Šilc, Eds., Jožef Stefan Institute, pp. 73–84.

[69] Korošec, P., Šilc, J., Filipič, B., and Laitinen, E. Ant stigmergy on the grid: optimizing the cooling process in continuous steel casting. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)* (Rhodes, Greece, April 25-29 2006), IEEE.

[70] Korošec, P., Šilc, J., and Robič, B. An ant-colony-optimization approach to the mesh-partitioning problem. In *Parallel Numerics '02. Theory and Applications* (2002), R. Trobec, P. Zinterhof, M. Vajteršic, and A. Uhl, Eds., University of Salzburg and Jožef Stefan Institute, pp. 123–132.

[71] Korošec, P., Šilc, J., and Robič, B. An experimental study of an ant-colony algorithm for the mesh-partitioning problem. *Parallel and Distributed Computing Practices 5*, 3 (April 2002), 313–320.

[72] Korošec, P., Šilc, J., and Robič, B. Applying the ant-colony algorithm for mesh-partitioning refinement. In *Proceedings of the 7th Joint Conference on Information Sciences (JCIS 2003)* (Research Triangle Park, NC, September 26-30 2003).

[73] Korošec, P., Šilc, J., and Robič, B. Mesh partitioning: a multilevel ant-colony-optimization algorithm. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003)* (Nice, France, April 22-26 2003), IEEE Computer Society.

[74] Korošec, P., Šilc, J., and Robič, B. A multilevel ant-colony-optimization algorithm for mesh partitioning. *International Journal of Pure and Applied Mathematics 5*, 2 (2003), 143–159.

[75] Korošec, P., Šilc, J., and Robič, B. Mesh-partitioning with the multiple ant-colony algorithm. In *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence* (Brussels, Belgium, September 5-8 2004), M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, Eds., vol. 3172 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 430–431.

[76] Korošec, P., Šilc, J., and Robič, B. Solving the mesh-partitioning problem with an ant-colony algorithm. *Parallel Computing 30*, 5-6 (May/June 2004), 785–801.

[77] Korošec, P., Šilc, J., and Robič, B. Solving the mesh-partitioning problem with an ant-colony algorithm: corrigendum. *Parallel Computing 30*, 8 (August 2004), 919–921.

[78] Korošec, P., Šilc, J., and Robič, B. Population-based methods as a from meta-heuristic combinatorial optimization. *Electrotechnical Review 72*, 4 (2005), 214–219.

[79] Krink, T., Filipič, B., Fogel, G. B., and Thomsen, R. Noisy optimization problems - a particular challenge for differential evolution? In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC2004)* (Portland, OR, June 19-23 2004), vol. 1, IEEE, Piscataway, pp. 332–339.

[80] Krink, T., and Ursem, R. K. Parameter control using the agent based patchwork model. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC2000)* (La Jolla, CA, July 16-19 2000), vol. 1, IEEE, Piscataway, pp. 77–83.

[81] Laitinen, E. Online control of secondary cooling in steel continuous casting process. In *Proceedings of the First Invited COST 526 Conference: Automatic Process Optimization in Materials Technology* (Morschach, Switzerland, May 30-31 2005), D. Büche and N. Hofmann, Eds., pp. 174–182.

[82] Lang, K., and Rao, S. A flow-based method for improving the expansion or conductance of graph cuts. In *Integer Programming and Combinatorial Optimization* (New York, NY, June 7-11 2004), vol. 3064 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 325–337.

[83] Langham, A. M. E., and Grant, P. W. Using competing ant colonies to solve k-way partitioning problems with foraging and raiding strategies. In *Proceedings of the Fifth European Conference on Artificial Life (ECAL)* (Lausanne, Switzerland, September 13-17 1999), D. Floreano, J.-D. Nicoud, and F. Mondada, Eds., vol. 1674 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 621–625.

[84] Linde, Y., Buzo, A., and Gray, R. M. An algorithm for vector quantizer design. *IEEE Transactions on Communications 28*, 1 (January 1980), 84–95.

[85] Lüscher, M. Air-conditioned termite nests. *Scientific American 205* (1961), 138–145.

[86] Maniezzo, V. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing 11*, 4 (1999), 358–369.

[87] Marais, E. N. *Die Siel van die Mier (The Soul of the Ant)*, fifth ed. J.L. van Schaik, Pretoria, South Africa, 1948. First published in 1937.

[88] Martin, J. G. Subproblem optimization by gene correlation with singular value decomposition. In *Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO 2005)* (Washington DC, 2005), ACM Press, pp. 1507–1514.

[89] Metaheuristics Network. http://www.metaheuristics.net/, visited in July 2006.

[90] Molina, D., Herrera, F., and Lozano, M. Adaptive local search parameters for real-coded memetic algorithms. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (Edinburg, UK, September 2-5 2005).

[91] Montemanni, R., Gambardella, L. M., Rizzoli, A. E., and Donati, A. V. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization 10*, 4 (2005), 327–343.

[92] Nocedal, J., and Wright, S. J. *Numerical Optimization*. Springer, New York, Berlin, Heidelberg, 1999.

[93] Osman, I. H., and Laporte, G. Metaheuristics: A bibliography. *Annals of Operations Research 63* (1996), 513–623.

[94] Papa, G., and Koroušić-Seljak, B. An artificial intelligence approach to the efficiency improvement of a universal motor. *Engineering Applications of Artificial Intelligence 18*, 1 (February 2005), 47–55.

[95] Pothen, A., Simon, H. D., and Liou, K.-P. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications 11*, 3 (July 1990), 430–452.

[96] Randall, M., and Lewis, A. A parallel implementation of ant colony optimization. *Journal of Parallel and Distributed Computing 62*, 9 (2002), 1421–1432.

[97] Rechenberg, I. *Evolution Strategies: Optimization of Technical Systems by Means of Biological Evolution*. Fromman-Holzboog, Stuttgart, Germany, 1973. In German.

[98] Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. *Engineering Optimization Methods*. Wiley, New York, 1983.

[99] Robič, B., Korošec, P., and Šilc, J. Ant colonies and the mesh-partitioning problem. In *Handbook of bioinspired algorithms and applications* (Boca Raton, London, New York, 2006), S. Olariu and A. Y. Zomaya, Eds., Computer and information science series, Chapman & Hall/CRC, pp. 285–303.

[100] Rönkkönen, J., Kukkonen, S., and Price, K. V. Real-parameter optimization with differential evolution. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (Edinburg, UK, September 2-5 2005).

[101] Roth, M., and Wicker, S. Termite: Ad-hoc networking with stigmergy. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2003)* (San Francisco, CA, December 2003), vol. 22, pp. 2937–2941.

[102] Schoonderwoerd, R., Holland, O. E., Bruten, J. L., and Rothkrantz, L. J. M. Ant-based load balancing in telecommunications networks. *Adaptive Behavior 4*, 2 (1996), 169–207.

[103] Shephard, M. S., Flaherty, J. E., de Cougny, H. L., Ozturan, C., Bottasso, C. L., and Beall, M. W. Parallel automated adaptive procedures for unstructured meshes. Tech. Rep. 1995-11, The Scientific Computation Research Center (SCOREC), Rensselaer Polytechnic Institute (RPI), Troy, NY, 1995.

[104] Socha, K. ACO for continuous and mixed-variable optimization. In *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence* (Brussels, Belgium, September 5-8 2004), M. Dorigo, M. Birat-

tari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, Eds., vol. 3172 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 25–36.

[105] Soper, A. J., Walshaw, C., and Cross, M. A combined evolutionary search and multilevel approach to graph partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00)* (Las Vegas, NV, July 8-12 2000), L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H.-G. Beyer, Eds., Morgan Kaufmann, pp. 674–681.

[106] Storn, R., and Price, K. V. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR 95-012, International Computer Science Institute, Berkley, CA, March 1995.

[107] Storn, R., and Price, K. V. Differential evolution – a fast and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11*, 4 (1997), 341–359.

[108] Stützle, T. *Local Search Algorithms for Combinatorial Problems-Analysis, Algorithms and New Applications*. PhD thesis, Darmstadt University of Technology, Department of Computer Science, Germany, 1998.

[109] Stützle, T. Parallelization strategies for ant colony optimization. In *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature (PPSN V)* (Amsterdam, The Nederlands, September 27-30 1998), A. E. Eiben, T. Bäck, H.-P. Schwefel, and M. Schoenauer, Eds., vol. 1498 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 722–741.

[110] Stützle, T., and Dorigo, M. ACO algorithms for the traveling salesman problem. In *Evolutionary Algorithms in Engineering and Computer Science* (April 1999), K. Miettinen, M. Mäkelä, P. Neittaanmäki, and J. Périaux, Eds., Wiley, New Jersey, pp. 163–183.

[111] Stützle, T., and Dorigo, M. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In *Handbook of Metaheuristics* (2002), F. Glover and G. A. Kochenberger, Eds., vol. 57 of *International Series in Operations Research & Management Science*, Kluwer Academic Publishers, Norwell, MA, pp. 251–285.

[112] Stützle, T., and Dorigo, M. A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation 6*, 4 (2002), 358–365.

[113] Stützle, T., and Hoos, H. H. Improvements on the ant system: Introducing max-min ant system. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms* (1997), Springer Verlag, pp. 245–249.

[114] Sunganthan, P. N., Hansen, N., Liang, J. J., Chen, Y. P., Auger, A., and Tiwari, S. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Tech. Rep., Nanyang Technological University, Singapore, May 2005.

[115] Šilc, J., and Korošec, P. The distributed stigmergic algorithm for multi-parameter optimization. In *Parallel Processing and Applied Mathematics, 6th International Conference, PPAM 2005, Poznań, Poland, September 11-14, 2005. Revised Selected Papers* (2006), R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, Eds., vol. 3911 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 92–99.

[116] Šilc, J., Korošec, P., and Robič, B. An experimental evaluation of modified algorithms for the graph partitioning problem. In *Proceedings of the 17th International Symposium on Computer and Information Sciences (ISCIS XVII)* (Orlando, FL, Octo-

ber 28-30 2002), I. Cicekli, N. K. Cicekli, and E. Gelenbe, Eds., CRC Press, Boca Raton, pp. 120–124.

[117] Šilc, J., Korošec, P., and Robič, B. Combining vector quantization and ant-colony algorithm for mesh-partitioning. In *Parallel Processing and Applied Mathematics, 5th International Conference, PPAM 2003, Czestochowa, Poland, September 7-10, 2003. Revised Papers* (2004), R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Wasniewski, Eds., vol. 3019 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 113–118.

[118] Tao, L., Zhao, Y. C., Thulasiraman, K., and Swamy, M. N. S. Simulated annealing and tabu search algorithms for multiway graph partition. *Journal of Circuits, Systems, and Computers 2*, 2 (June 1992), 159–185.

[119] Tsutsui, S. Ant colony optimization for continuous domain with aggregation pheromones metaphor. In *Proceedings of the 5th International Conference on Recent Advances in Soft Computing* (Nottingham, UK, December 2004), pp. 207–212.

[120] Tušar, T., Korošec, P., Papa, G., Filipič, B., and Šilc, J. A comparative study of stochastic optimization methods in electric motor design. *Applied Intelligence* (2007). To appear.

[121] Vesterstrøm, J., and Thomsen, R. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 Congress on Evolutionary Computation (CEC2004)* (Portland, OR, June 19-23 2004), vol. 1, IEEE, Piscataway, pp. 1980–1987.

[122] von Frisch, K. *The Dance Language and Orientation of Bees*. Harvard University Press, 1967.

[123] Voronoi, G. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik 133* (1907), 97–178.

[124] Voß, S., Martello, S., Osman, I. H., and Roucairol, C. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[125] Walshaw, C. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research 131*, 1-4 (October 2004), 325–372.

[126] Walshaw, C., and Cross, M. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing 22*, 1 (2000), 63–80.

[127] Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[128] Whitley, L. D. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms* (San Mateo, CA, 1989), J. D. Schaffer, Ed., Morgan Kaufman, pp. 116–121.

[129] Wright, A. H. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms - 1* (San Mateo, CA, 1991), G. J. E. Rawlins, Ed., Morgan Kaufman, pp. 205–218.

[130] Yuan, B., and Gallagher, M. Experimental results for the special session on real-parameter optimization at CEC 2005: A simple, continuous EDA. In *Proceedings*

of the 2005 IEEE Congress on Evolutionary Computation (Edinburg, UK, September 2-5 2005).

[131] Zomaya, A. Y., Andreson, J. A., Fogel, D. B., Milburn, G. J., and Rozenberg, G. Non-conventional computing paradigms in the new millennium. *IEEE/AIP Computing in Science and Engineering 3*, 6 (November/December 2001), 82–99.

[132] Żola, J., and Wyrzykowski, R. Application of genetic algorithm for mesh partitioning. In *Proceedings of the 5th International Workshop on Parallel Numerics - ParNum2000* (Bratislava, Slovakia, 2000), pp. 209–217.

Index

List of Algorithms

List of Figures

List of Tables

Stigmergija kot pristop k metahevristični optimizaciji

Optimizacija je zaradi svoje praktične pomembnosti še vedno predmet intenzivnih raziskav. Za veliko optimizacijskih problemov je značilno, da njihova računska zahtevnost z velikostjo problema eksponentno narašča. Za ta razred problemov, znan v teoriji izračunljivosti kot NP, obstaja vrsta neeksaktnih postopkov reševanja, ki sicer ne zagotavljajo optimalnosti rešitev, vendar v sprejemljivem (polinomskem) času dajejo rešitve, ki so zadosti blizu optimalnim. Mednje sodijo tudi t. i. metahevristične metode, kot so lokalno preiskovanje, simulirano ohlajanje, iskanje s tabuji, evolucijski algoritmi in optimizacija s kolonijami mravelj. Pod metahevristiko razumemo strategijo, ki usmerja preiskovanje rešitev v postopku optimizacije, pri čemer sta njena cilja računska učinkovitost in čim boljša rešitev problema. Metahevristike so ponavadi stohastične, kar pomeni, da pri preiskovanju uporabljajo verjetnostna pravila in so njihovi koraki odvisni od psevdonaključnih števil. Metahevristike niso problemsko specifične, lahko pa izkoriščajo problemsko specifično znanje za povečanje učinkovitosti. Uveljavljena načina pospešitve optimizacijskega postopka sta poleg uporabe metahevristik še večnivojski način reševanja in izvajanje na vzporednih ali porazdeljenih računalniških sistemih. Za reševanje zahtevnih problemov numerične in kombinatorične optimizacije je zato smiselno uporabiti večnivojske metahevristične optimizacijske algoritme in jih prirediti za vzporedno ali porazdeljeno izvajanje.

Na področju kombinatorične in numerične optimizacije so se uveljavile številne metahevristične metode, med njimi tudi mnoge, ki delujejo po vzorih iz narave. Tak primer je tudi stigmergija, to je način komuniciranja v porazdeljenih sistemih, kjer komponente sistema izmenjujejo informacije preko spreminjanja svojega lokalnega okolja in zaznavanja sprememb v njem. V naravi je ta pojav značilen za nekatere vrste družabnih žuželk (mravlje, čebele, termite, ose) in je temelj njihovega skupinskega reševanja problemov. Termin stigmergija je 1959 prvi uvedel francoski biolog Grassé,

ko je raziskoval gradnjo termitnjakov. Stigmergija je skovanka iz grških besed stigma ($\sigma\tau\iota\gamma\mu\alpha$ = spodbuda) in ergon ($\varepsilon\rho\gamma\sigma\nu$ = delo, produkt dela).

V zadnjem desetletju se ta koncept uveljavlja tudi v računalniškem reševanju problemov z doslej znanimi primeri uporabe v robotiki, večagentnih sistemih, komunikaciji v omrežjih in kombinatorični optimizaciji. Znani algoritem optimizacije s kolonijami mravelj (angl. Ant-Colony Optimization, ACO) se zgleduje po stigmergiji med mravljami v naravi in s posnemanjem odlaganja in zaznavanja feromona učinkovito usmerja preiskovanje prostora rešitev.

To metodologijo, ki je bila doslej uporabljena le na kombinatoričnih optimizacijskih problemih, smo izpopolniti tako, da je uporabna kot splošna metoda za reševanje zveznih numeričnih optimizacijskih problemov.

Pristop z več kolonijami mravelj

Optimizacijska metoda, ki temelji na stigmergiji pri mravljah, se lahko uspešno uporabi za reševanje problema razdelitve grafa, ki je eden izmed NP-težkih kombinatoričnih optimizacijskih problemov. Na osnovi te smo razvili algoritem z več kolonijami mravelj in ga poimenovali MACA (iz angl. Multiple Ant-Colony Algorithm). Raziskovali smo nekatere različice algoritma MACA za problem razdelitve mreže. Poleg osnovnega algoritma smo predlagali več izboljšav in jih ovrednotili. Osnovni algoritem se je izkazal na manjših mrežah; pri večjih mrežah, ki so pogostejše, smo morali uporabiti večnivojsko in hibridno inačico osnovnega algoritma. Tako smo dosegli rezultate, ki so primerljivi z ostalimi, iz literature znanimi algoritmi za razdelitev mreže. Večnivojski in hibridni algoritem sta se izkazala za odlična pri skoraj vseh testnih mrežah. Oba sta dajala zelo podobne najboljše rezultate. Edina omembe vredna razlika je bila v razpršenosti rezultatov, ki je bila manjša pri hibridnem algoritmu.

Večnivojski pristop s stigmergijo mravelj

Večnivojski algoritem s stigmergijo mravelj, imenovan MASA (iz angl. Multilevel Ant-Stigmergy Algorithm), je nova metoda za reševanje diskretnih numeričnih (več-

parametrskih) optimizacijskih problemov, ki temelji na stigmergiji. Predlagali smo splošni pristop k preslikavi večparametrskega optimizacijskega problema v problem iskanja najcenejše poti v preiskovalnem grafu. Vsaka najdaljša pot v preiskovalnem grafu predstavlja možno rešitev; vse najdaljše poti skupaj pa predstavljajo celoten prostor rešitev večparametrskega problema. Za učinkovito preiskovanje prostora rešitev smo ponovno uporabili večnivojski pristop.

Učinkovitost algoritma MASA smo primerjali z učinkovitostjo diferencialne evolucije (angl. Differential Evolution). Primerjavo smo izvedli na nekaterih splošno znanih testnih funkcijah. Izkazalo se je, da sta pri nizkodimenzionalnih funkcijah algoritma enakovredna, medtem ko je pri visokodimenzionalnih funkcijah algoritem MASA dal boljše rezultate od diferencialne evolucije na skoraj vseh testnih funkcijah.

Dober približek praktičnim problemom so t. i. šumne testne funkcije. Zato smo primerjali algoritem MASA in diferencialno evolucijo tudi na naboru šumnih testnih funkcij. Pri vrednotenju algoritma smo se odločili, da bomo obravnavali tri stopnje šumnosti: nizko, srednjo in visoko. Ugotovili smo, da je vpliv stopnje šumnosti na kakovost rešitev izrazit predvsem pri rešitvah, dobljenih z diferencialno evolucijo, medtem ko je pri algoritmu MASA ta vpliv mnogo manjši.

Diferencialni pristop s stigmergijo mravelj

Nazadnje smo predlagali še razširitev optimizacije s kolonijami mravelj na zvezno domeno. Ta novi pristop smo imenovali diferencialni algoritem s stigmergijo mravelj in ga krajše označili DASA (iz angl. Differential Ant-Stigmergy Algorithm). Preučevali smo ga na novih, v letu 2005 objavljenih večparametrskih testnih funkcijah.

Algoritem smo primerjali s številnimi evolucijskimi optimizacijskimi algoritmi: evolucijsko strategijo s prilagajanjem kovariančne matrike (angl. Covariance Matrix Adaptation Evolution Strategy), diferencialno evolucijo, memetskim algoritmom z realnim kodiranjem rešitev (angl. Real-Coded Memetic Algorithm) in algoritmom z zvezno oceno porazdelitve (angl. Continuous Estimation of Distribution Algorithm).

Dobljeni rezultati so obetavni in nakazujejo visoko učinkovitost našega algoritma DASA. V treh od štirih primerov testnih funkcij se je algoritem izkazal za boljšega od

ostalih. Ker so bile testne funkcije izbrane tako, da so pokrivale širok spekter praktičnih optimizacijskih problemov, lahko sklepamo, da je algoritem DASA primeren za reševanje splošnih večparametrskih optimizacijskih problemov v praksi.

Praktična uporaba

V želji, da bi ovrednotili delovanje novo predlaganih algoritmov na praktičnih primerih, smo se odločili, da jih preizkusimo na dveh zahtevnih problemih s področja elektrotehnike in metalurgije.

V prvem primeru smo optimirali izgube moči v univerzalnem elektromotorju, ki ga izdeluje podjetje Domel iz Železnikov. Največ izgub moči elektromotorja nastane v navitju, rotorju in statorju. S primerno oblikovanim rotorjem in statorjem se da izgube občutno zmanjšati. Zato smo se lotili optimiranja parametrov, ki določajo geometrijo rotorja in statorja. Čeprav je geometrija rotorja in statorja določena z več deset parametri, smo ugotovili, da je le 11 neodvisnih, in ti so bili predmet optimiranja. Kljub temu pa pri danih tehnoloških omejitvah (območje parametrov in njihova toleranca) naraste problem na približno $4.8 \cdot 10^{22}$ možnih nastavitev parametrov.

Pomemben del v procesu optimiranja je simulator, ki ovrednoti vsako rešitev. V primeru elektromotorja smo uporabili programski paket ANSYS.

Rezultate pridobljene z algoritmom MASA smo primerjali z rezultati štirih stohastičnih optimizacijskih metod. Kot smo že omenili, je bil primarni cilj zmanjšanje izgub moči, toda končni rezultat je bil ocenjen tudi z vidika primernosti za proizvodnjo. Primerjali smo naslednje metode: generacijski evolucijski algoritem (angl. Generational Evolutionary Algorithm), evolucijski algoritem s stabilnim stanjem (angl. Steady-State Evolutionary Algorithm), diferencialno evolucijo in algoritem po vzoru iz elektromagnetizma (angl. Electromagnetism Algorithm).

Primerjavo lahko povzamemo z naslednjimi pomembnejšimi ugotovitvami. Vsi optimizacijski algoritmi so uspeli občutno izboljšati osnovno inženirsko rešitev. Med njimi je algoritem MASA našel najboljšo rešitev. Prednost algoritma MASA pred ostalimi izhaja, po našem mnenju, iz večnivojskega pristopa.

Algoritem MASA se je na tem problemu izkazal tudi zato, ker ni potreboval začetne rešitve. Vsi ostali algoritmi namreč delujejo na osnovi neke začetne populacije rešitev, do katere pa je bilo v tem primeru težko priti, ker je bil delež dopustnih rešitev zelo majhen. Z naključnim izborom do dopustnih rešitev praktično nismo prišli, tako da smo za začetno populacijo uporabili znano inženirsko rešitev in na osnovi nje izpeljane različice.

Analiza primernosti optimirane geometrije elektromotorja za proizvodnjo je pokazala nekatere pomanjkljivosti. Čeprav je bila oblika rotorja in statorja v skladu s tehnološkimi omejitvami parametrov, se je izkazalo, da je neizvedljiva zaradi omejitev v proizvodnem procesu (pretanek zob statorja). Rešitev tega problema bi bila dopolnitev opisa elektromotorja v okolju ANSYS.

Ker je ovrednotenje vmesnih rešitev pri zaporedni izvedbi algoritma MASA časovno zelo zahtevna, smo se odločili, da realiziramo porazdeljeno različico algoritma MASA. Rezultati, dobljeni s tako preurejenim algoritmom, so pokazali, da drastično zmanjšamo čas izračuna, iz enega dneva na nekaj ur. Pri tem ni bilo opazne razlike v kakovosti končnega rezultata.

Kljub nekoliko nižji kakovosti dobljenih rezultatov pri računanju na osmih procesorjih so bili ti še vedno boljši od tistih, ki so jih pridobili ostali (zaporedni) optimizacijski algoritmi.

Ker je algoritem DASA sposoben reševati tudi diskretne numerične optimizacijske probleme, smo se odločili, da ga preverimo tudi na problemu optimizacije izgub moči v elektromotorju. Izkazalo se je, da je uspel najti rezultate podobne tistim, dobljenim z algoritmom MASA. Zanimivo je, da je tudi algoritem DASA, tako kot zgoraj omenjeni optimizacijski algoritmi, ki delujejo na osnovi neke začetne populacije rešitev, začel iz inženirske rešitve, a je le on vračal podobne rezultate kot algoritem MASA. Čeprav so bili rezultati algoritma DASA v povprečju malenkost slabši od rezultatov algoritma MASA, nas je ta rezultat dodatno prepričal v zmožnosti algoritma DASA.

Drugi praktični optimizacijski problem je bilo optimiranje nastavitev ohlajanja med kontinuiranim ulivanjem jekla v jeklarni Rukki Steel v mestu Raahe na Finskem. Pravilno ohlajanje je ključ do višje kakovosti jekla. Dandanes se optimiranje izvaja z uporabo numerične simulacije procesa in naprednih optimizacijskih tehnik.

Algoritem MASA smo uporabili pri optimiranju 18 vodnih hladilnih prh (parametrov) na livni napravi, pri čemer smo se omejili na stabilne delovne pogoje. Zaradi omejitev, ki so bile postavljene s strani jeklarne, je algoritem MASA vedno našel optimalno vrednost vseh 18 parametrov, za kar je algoritem v povprečju potreboval 12 ur. To nas je vodilo k uporabi porazdeljenega algoritma MASA.

Pokazali smo, da lahko s porazdeljenim algoritmom MASA občutno skrajšamo čas računanja, z dvanajst na tri ure. Pri tem je ostala kakovost rezultatov enaka.

Tudi ta praktični problem smo reševali z algoritmom DASA. Tokrat smo se odločili, da sprostimo omejitve parametrov. V primerjavi z diferencialno evolucijo in algoritmom MASA je algoritem DASA vračal kakovostnejše rezultate in prednjačil z izrazitejšo konvergenco.

Značilnosti algoritmov

Pomembno vprašanje, na katerega smo v tej disertaciji poskušali odgovoriti, se je glasilo: "Kateri algoritem, MASA ali DASA, naj uporabimo za dani problem?" Da smo lahko odgovorili na to vprašanje, smo najprej analizirali prednosti in slabosti obeh algoritmov, ki jih tu povzemamo.

Glavna slabost oz. omejitev algoritma MASA je, da je potrebno v primeru zvezne optimizacije prevesti reševani problem v diskretno obliko (t. i. preiskovalni graf). Velikost preiskovalnega grafa določa želena natančnost parametrov, ki jih optimiramo. V primeru zelo velikih preiskovalnih grafov smo omejeni z velikostjo spomina v računalniku. Torej je natančnost, ki jo lahko dosežemo s tem algoritmom, omejena. Nadalje smo opazili, da sta časovna in prostorska zahtevnost algoritma MASA mnogo večji od zahtevnosti algoritma DASA. To ni omembe vredna omejitev, kadar je vrednotenje rešitve časovno zahtevno, kar je sicer v praktičnih problemih pogost pojav. Pri vrednotenju enostavnih funkcij pa bi ta zahtevnost algoritma MASA prišla bolj do izraza. Opazili smo še, da uporaba večnivojskega pristopa na nek način omejuje konvergenco. Ker je algoritem MASA zasnovan tako, da se pri prehajanju z nivoja na nivo na vsakem od njih zadržuje določen čas (določeno število ovrednotenj vmesnih rešitev), se lahko zgodi, da na nekem nivoju stagnira (ne izboljšuje vmesnih rešitev).

Posledica je množica nepotrebnih ovrednotenj, kar vpliva na konvergenco. Toda po drugi strani daje večnivojski pristop algoritmu MASA tudi prednost. Izkazalo se je, da smo z uporabo večnivojskega pristopa uspeli najti dobro rešitev tudi v zelo redkem preiskovalnem prostoru, kar je bil že omenjeni primer optimizacije elektromotorja. Z algoritmom MASA smo uspeli najti rezultate, ki so bili nedosegljivi za ostale optimizacijske algoritme.

Večina omejitev, ki veljajo za algoritem MASA, ne velja za algoritem DASA. Pri algoritmu DASA ni bilo potrebe po prevedbi problema v diskretno obliko in s tem tudi ni bilo več težav z natančnostjo. Ta je bila omejena le še s strojno opremo (natančnost aritmetike s plavajočo vejico). Kot smo že omenili, sta časovna in prostorska zahtevnost algoritma DASA mnogo manjši od zahtevnosti algoritma MASA. Slabost algoritma DASA v primerjavi z algoritmom MASA je, da potrebuje začetno rešitev.

Odgovor na začetno zastavljeno vprašanje se glasi: "Za primere problemov, kjer je preiskovalni prostor redek in začetna rešitev ni poznana, priporočamo uporabo algoritma MASA, za vse ostale probleme pa predlagamo uporabo algoritma DASA."

Čeprav so poskusi, ki smo jih naredili, obsegali le del teoretičnih in praktičnih optimizacijskih problemov, so vseeno pokazatelj o tem, kako se bosta predlagana algoritma obnašala v določenih pogojih.

Kot večina metahevrstičnih algoritmov imata tudi algoritma MASA in DASA nekaj parametrov, katerih vrednosti je potrebno uglasiti. Med poskusi, ki smo jih opravili, nismo posvečali posebne pozornosti uglaševanju vrednosti parametrov, zato menimo, da je potrebna študija, ki bi pokazala, kako različne nastavitve vplivajo na delovanje obeh algoritmov. Nadalje menimo, da bi bilo smiselno algoritem DASA, ki je dobro deloval na skoraj vseh problemih, prirediti tudi za vzporedno ali porazdeljeno izvajanje.

Prispevki k znanosti

Glavni prispevek disertacije k znanosti sta izvirna metahevristična postopka za reševanje težkih kombinatoričnih in numeričnih optimizacijskih problemov, ki sta razvita na osnovi optimizacije s kolonijami mravelj. Specifični prispevki v okviru razvitih optimizacijskih metod so naslednji:

S prvo metodo (algoritmom MASA) smo uvedli nov način prevedbe večparametrskega optimizacijskega problema v problem iskanja najcenejše poti v preiskovalnem grafu. Večnivojski pristop je bil uspešno uporabljen za reševanje numeričnih optimizacijskih problemov. Realizirana je bila tudi vzporedna izvedba optimizacijske metode v večprocesorskem okolju.

Z drugo metodo (algoritmom DASA) smo uvedli drugačen način prevedbe večparametrskega optimizacijskega problema v graf, ki nam je omogočal reševanje zveznih numeričnih optimizacijskih problemov.

Obe metodi smo tudi uspešno uporabili na umetnih in praktičnih optimizacijskih problemih ter z njima izboljšali doslej znane rezultate.

Biography

Peter Korošec was born on the 21st of January 1977 in Ljubljana. After finishing elementary school he attended Gimnazija Vič in Ljubljana. He continued his education at the Faculty of Computer and Information Science, University of Ljubljana. While studying at the faculty he obtained a scholarship from the "Jožef Stefan" Institute in Ljubljana. In 2001 he graduated with an undergraduate thesis called "Evaluation of Heuristic Partitioning Algorithms", under the supervision of Professor Borut Robič. Since 2002 he has been with the Computer Systems Department of the "Jožef Stefan" Institute. From 2002 to 2006 he was a young researcher mentored by Assistant Professor Jurij Šilc, researching modern optimization methods, mainly ant-colony optimization. In 2004 he obtained his master's degree in Computer and Information Science with a thesis entitled "Metaheuristic Solving of the Optimization Problem with Ant Colonies", also under the supervision of Professor Borut Robič. The results of his work were published in domestic and international journals and presented at conferences in Slovenia and abroad. He has published more than 30 papers.

Življenjepis

Peter Korošec se je rodil 21. januarja 1977 v Ljubljani. Po končani osnovni šoli je obiskoval gimnazijo Vič v Ljubljani. Šolanje je nadaljeval na Fakulteti za računalništvo in informatiko na Univerzi v Ljubljani. V času univerzitetnega študija je bil štipendist Instituta "Jožef Stefan" v Ljubljani. Leta 2001 je opravil diplomsko nalogo z naslovom: "Vrednotenje hevrističnih razdelitvenih algoritmov", pod mentorstvom prof. dr. Boruta Robiča. Od leta 2002 je zaposlen na Odseku za računalniške sisteme Instituta "Jožef Stefan". V letih od 2002 do 2006 se je kot mladi raziskovalec pod vodstvom doc. dr. Jurija Šilca ukvarjal z razvojem sodobnih optimizacijskih metod, predvsem optimizacije s kolonijami mravelj. Leta 2004 je opravil magistrsko nalogo iz računalništva in informatike z naslovom: "Metahevristično reševanje optimizacijskega problema s kolonijami mravelj", prav tako pod mentorstvom prof. dr. Boruta Robiča. Rezultate svojega dela je objavil v domačih in mednarodnih revijah ter jih predstavil na konferencah v Sloveniji in tujini. Skupaj ima objavljenih več kot 30 del.

Selected publications Izbrane objave

- Tea Tušar, Peter Korošec, Gregor Papa, Bogdan Filipič, Jurij Šilc, "A comparative study of stochastic optimization methods in electric motor design", *Applied Intelligence*, 2007. To appear.
- Peter Korošec, Jurij Šilc, "The multilevel ant stigmergy algorithm for numerical optimization", *Facta Universitatis. Series Electronics and Energetics*, vol. 19, no. 2, pp. 247–260, 2006.
- Jurij Šilc, Peter Korošec, "The distributed stigmergic algorithm for multi-parameter optimization", In: Parallel processing and applied mathematics: 6th International Conference, PPAM 2005, Poznań, Poland, September 11-14, 2005: revised selected papers, (*Lecture Notes in Computer Science*, vol. 3911), Berlin, Heidelberg, 2006, pp. 92–99.
- Borut Robič, Peter Korošec, Jurij Šilc, "Ant colonies and the mesh-partitioning problem", In: *Handbook of Bioinspired Algorithms and Applications*, (Chapman & Hall/CRC computer and information science series), Stephan Olariu, Albert Y. Zomaya, eds., Boca Raton, London, New York, Chapman & Hall/CRC, 2006, pp. 285–303.
- Peter Korošec, Jurij Šilc, Borut Robič, "Population-based methods as a form of metaheuristic combinatorial optimization", *Elektrotehniški vestnik*, vol. 72, no. 4, pp. 214–219, 2005. In Slovenian.
- Peter Korošec, Jurij Šilc, Borut Robič, "Solving the mesh-partitioning problem with an ant-colony algorithm: [Parallel computing 30, (2004), 785-801]: corrigendum", *Parallel Computing*, vol. 30, pp. 919–921, 2004.
- Peter Korošec, Jurij Šilc, Borut Robič, "Solving the mesh-partitioning problem with an ant-colony algorithm", *Parallel Computing*, vol. 30, pp. 785–801, 2004.
- Peter Korošec, "Metaheuristic Solving of the Optimization Problem with Ant Colonies", Master's Thesis, Ljubljana, 2004. In Slovenian.
- Peter Korošec, Jurij Šilc, Borut Robič, "Mesh-partitioning with the multiple ant-colony algorithm", In: Ant colony optimization and swarm intelligence: 4th International Workshop, ANTS 2004, Brussels, Belgium, September 5-8, 2004: proceedings, (*Lecture Notes in Computer Science*, vol. 3172), Berlin, Heidelberg, New York, Springer, 2004, pp. 430–431.
- Jurij Šilc, Peter Korošec, Borut Robič, "Combining vector quantization and ant-colony algorithm for mesh-partitioning", In: Parallel processing and applied mathematics: 5th International Conference, PPAM 2003, Częstochowa, Poland, September 7-10, 2003: revised papers, (*Lecture Notes in Computer Science*, vol. 3019), Berlin [etc.], Springer, 2004, pp. 113–118.
- Peter Korošec, Jurij Šilc, Borut Robič, "A multilevel ant-colony optimization algorithm for mesh partitioning", *International Journal of Pure and Applied Mathematics*, vol. 5, pp. 143–159, 2003.
- Peter Korošec, Jurij Šilc, Borut Robič, "An experimental study of an ant-colony algorithm for the mesh-partitioning problem", *Parallel and Distributed Computing Practices*, vol. 5, pp. 313–320, 2002.
- Peter Korošec, "Evaluation of Heuristic Partitioning Algorithms", Undergraduate Thesis, Ljubljana, 2001. In Slovenian.