Ant Colony Optimization for Continuous and Mixed-Variable Domains

Krzysztof Socha
of
IRIDIA, CoDE, Université Libre de Bruxelles, CP 194/6,
Av. Franklin D. Roosevelt 50, 1050 Brussels, Belgium
ksocha@ulb.ac.be
http://iridia.ulb.ac.be

Promoteur: Prof. Marco Dorigo
Thése présentée en vue de l'obtention du titre de Docteur en Science Appliquées.

Summary

Ant colony optimization (ACO) has become one of the popular metaheuristics used for tackling optimization problems. Its popularity grows steadily, and the area of applications constantly widens. The recent research shows that ACO is competitive and applicable to various real-world optimization problems. Similarly to many other metaheuristics, ACO has been initially developed for combinatorial optimization problems. These problems are characterized by a finite set of states that each decision variable may assume. Later, there have been attempts to extend it also to continuous problems, that is, problems where the decision variables may assume any real value from a given domain. However, these attempts were not following the original ACO formulation. Also, none of the ACO-inspired approaches proposed so far could tackle mixed-variable optimization problems, that is, problems that combine both discrete and continuous components. This means that any given decision variable may be either discrete or continuous.

In this work, we present a way to extend ACO, so that it can be applied to both continuous and mixed-variable optimization problems. We demonstrate, first, how ACO may be extended to continuous domains. We describe the algorithm proposed, discuss the different design decisions made, and we position it among other metaheuristics. Following this, we present the results of numerous simulations and testing. We compare the results obtained by the proposed algorithm on typical benchmark problems with those obtained by other methods used for tackling continuous optimization problems in the literature. Finally, we investigate how our algorithm performs on a real-world problem coming from the medical field—we use our algorithm for training neural network used for pattern classification in disease recognition.

Following an extensive analysis of the performance of ACO extended to continuous domains, we present how it may be further adapted to handle both continuous and discrete variables simultaneously. We thus introduce the first native mixed-variable version of an ACO algorithm.

Mixed-variable optimization problems are often tackled with continuous optimization algorithms by relaxing some of the constraints, and then repairing the obtained results. In order to analyze the performance of this native mixed-variable ACO algorithm, and to compare it to the performance of the continuous version of the ACO algorithm, we propose a specific benchmark problem. We investigate, for which types of problems the native mixed-variable approach should outperform the continuous-relaxation approach.

Then, we analyze and compare the performance of both continuous and mixed-variable ACO algorithms on different benchmark problems from the literature. Through the research performed, we gain some insight into the relationship between the formulation of mixed-variable problems, and the best methods to tackle them. Furthermore, we demonstrate that the performance of ACO on various real-world mixed-variable optimization problems coming from the mechanical engineering field is comparable to the state of the art.

The proposed algorithms follow closely the original ACO formulation that was initially defined for combinatorial optimization problems. We show how these fundamental ideas may be further extended to handle also other types of problems. Although there have been already attempts to apply ACO to continuous domains, they did not follow the original idea closely. Furthermore, there were no attempts to propose a mixed-variable version of the ACO metaheuristic before. The application of ACO to new types of problems while maintaining the original ideas is the main contribution of this work.

Original Contributions

The following is a summary of the main contributions presented in this work:

- **Formal and coherent definition of the combinatorial, continuous, and mixed-variable optimization problems:** While the combinatorial, continuous, and mixed-variable problems have been identified and defined before, we provide a coherent definition that is common for all of them based on the way the combinatorial problems are usually defined. This allows demonstrating how ACO algorithms may also tackle the continuous and mixed-variable optimization problems by emphasizing the crucial differences and similarities between these types of problems.

- **Ant colony optimization algorithm for continuous domains ($\mathbf{ACO_{\mathbb{R}}}$):** One of the most significant contributions of this work. We present how ACO can be extended to continuous domains with the pheromone modeled by probability density functions instead of a table. We describe the underlying idea, we also present a fully functional algorithm—$ACO_{\mathbb{R}}$—and we demonstrate its performance on a large number of benchmark problems.

- **Application of $\mathbf{ACO_{\mathbb{R}}}$ to training a neural network for pattern classification in the medical field:** We evaluate the performance of $ACO_{\mathbb{R}}$ on a real-world problem. We present how $ACO_{\mathbb{R}}$ may be applied to the problem of training neural networks for pattern classification in the medical field. We demonstrate that the performance of $ACO_{\mathbb{R}}$ is competitive, when compared to genetic algorithms, and that a hybridized version of $ACO_{\mathbb{R}}$ with a derivative-based method outperforms typical methods used for neural network training.

- **A benchmark mixed-variable optimization problem with well-controlled characteristics:** We propose a new mixed-variable benchmark problem, which provides a well-controlled environment for testing algorithms. It allows for adjusting its characteristics and difficulty level, and hence provides an excellent testing ground for mixed-variable optimization algorithms.

- **Ant colony optimization algorithm for mixed-variable domains ($\mathbf{ACO_{MV}}$):** Following the general idea of $\mathrm{ACO}_{\mathbb{R}}$, we propose a further extended version of this algorithm—$\mathrm{ACO}_{\mathbf{MV}}$—that is able to handle both continuous and discrete decision variables. We investigate its basic performance using the benchmark problem that we proposed.

- **Application of $\mathbf{ACO}_{\mathbb{R}}$ and $\mathbf{ACO_{MV}}$ algorithms to real-world engineering mixed-variable optimization problems:** We apply our mixed-variable $\mathrm{ACO}_{\mathbf{MV}}$ algorithm along with the $\mathrm{ACO}_{\mathbb{R}}$ algorithm mentioned earlier to three engineering real-world mixed-variable optimization problems. We demonstrate, how, in relation to the problem formulation, each of them has certain advantages. We compare the results obtained with those of different methods found in the literature.

Statement

This work describes an original research carried out by the author. It has not been previously submitted to the Université Libre de Bruxelles, nor to any other university for the award of any degree. Nevertheless, some chapters of this thesis are partially based on papers, which the author, together with a number of co-workers, has published or submitted for publication in the scientific literature.

A preliminary version of the overview of the ant colony optimization metaheuristic provided in Chapter 3 has been published in:

M. Dorigo and K. Socha, Ant Colony Optimization, chapter 26 of *Handbook on Approximation Algorithms and Metaheuristics*, T. Gonzalez and S.Sahni eds., CRC in the Computer & Information Science series, Chapman & Hall, 1st edition, 2007

Chapter 4 is based on the following two papers (one already published and one accepted for publication):

K. Socha, ACO for continuous and mixed-variable optimization, *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, M. Dorigo and M. Birattari and C. Blum and L. M. Gambardella and F. Mondada and T. Stützle, eds., LNCS, volume 3172, Springer-Verlag, Berlin, Germany, 2004, pp. 25–36

K. Socha and M. Dorigo, Ant colony optimization for continuous domains, *European Journal of Operations Research*, 185(3):1155–1173, 2008

The results presented in Chapter 5 are based on the following three works:

C. Blum and K. Socha, Training feed-forward neural networks with ant colony optimization: An application to pattern classification, *CD-ROM Proceedings of Hybrid Intelligent Systems Conference, HIS-2005*, 2005.

K. Socha and C. Blum, Ant Colony Optimization, chapter 8 of *Metaheuristic Procedures for Training Neural Networks*, E. Alba and R. Martí, eds., Springer-Verlag, Berlin, Germany, 1st edition, 2006, pp. 153–180

K. Socha and C. Blum, Hybrid ant algorithms applied to feed-forward neural network training: An application to medical pattern classification, *Neural Computing and Applications*, 16(3):235–248, 2007

Finally, the concept of native mixed-variable ACO algorithm and the initial results that make up the major part of Chapter 6, as well as the algorithm description included in Chapter 4, are parts of a paper submitted for publication:

K. Socha and M. Dorigo, Ant Colony Optimization for Mixed-Variable Optimization Problems, *Journal of Mechanical Design*, (submitted)

This thesis has been typeset in LaTeX, a free and efficient typesetting system. All the algorithms developed in the course of the research were implemented in R—a free alternative to the S+ programming language. All the plots and statistical analysis included in the paper were also produced using R. The experiments were run on various Linux systems. All the algorithms developed during the research are open source and have been made available to the public based on the GNU Public License (version 2 or later).

Acknowledgments

It took me some time to write this thesis, and this would not have been possible without the finantial and moral support of several individuals and institutions.

The financial support has been indispensable, so I start with the financial supporters and contributors. First, I would not have had even started, had it not been for Prof. Janusz Filipiak, the founder and president of ComArch S.A. It was thanks to the opportunity provided by Prof. Filipiak that I first came to *Université Libre de Bruxelles* (ULB), in 2000. ComArch S.A. has partially supported me during 2000 and 2001, while in preparation for starting the Ph.D., I have obtained the *Diplôme d'Etudes Approfondis* (DEA).

Merely coming to Belgium would hardly enable me to start working on this thesis, had it not been for Prof. Paul Van Binst of *Service Télématique et Communication* (STC) at ULB, who encouraged my interest in research and put me in touch with Prof. Hugues Bersini of *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle* (IRIDIA), also at ULB. Prof. Van Binst not only supported me in the idea of pursuing the Ph.D., but also ensured the financial means to do so by supporting my application for a fellowship from ULB, which I was receiving between October 2001 and October 2005. STC has also supported financially my visits to several conferences and seminars throughout these years.

Once I met Prof. Hugues Bersini at IRIDIA, the real path to writing this thesis has begun. Prof. Bersini was kind enough to listen to my ideasand support my interest in research. He also have introduced me to Prof. Marco Dorigo, who eventually became the supervisor of my thesis. This allowed me to get involved in the research activities in IRIDIA. Initially, while I was involved in the *Metaheuristics Network*, a Research Training Network funded by the Improving Human Potential Programme of the CEC, my work was partially supported by the grant HPRN-CT-1999-00106. Later, my work was also partially supported by the "ANTS" project, an "Action de Recherche Concertée" funded by the Scientific Research Directorate of the French Community of Belgium. Thanks to a friendly care of Prof. Bersini and financial support provided by or through IRIDIA, I managed to attend summer schools, as well as numerous conferences and seminars.

Apart from the necessary financial support, my thesis would have never been created, if it was not for numerous people, who encouraged me, supported me, and helped me with their knowledge or sometimes even a good word. My adventure with research had really started when working on my Master Degree thesis in AGH-Technical University of Cracow in 1999. Thanks to encouraging guidance and friendly support of Dr. Marek Kisiel-Dorohinicki, I learned the basics of good research and co-authored my first conference paper. Dr. Kisiel-Dorohinicki was the one who introduced me to the magic realm of LaTeX, publications, and research work.

Prof. Paul Van Binst was the next key figure in my road to writing this thesis. It was his never-subsiding enthusiasm, encouragement, and support that helped me to finally reach the end of this road. It was through him that I got to know Prof. Hugues Bersini, Prof. Marco Dorigo, and all the other wonderful people in IRIDIA and beyond. It was him, who supported me through the whole time. It was him, who was always open for most interesting and educating discussions about science, world, and life.

On the scientific level, I owe most of my thanks to Prof. Marco Dorigo, and other members of IRIDIA, with whom I closely collaborated. Prof. Dorigo has been always open to my ideas and always ready to provide advice to improve and guide my research. I was extremely fortunate to be able to freely choose the direction of my research with Prof. Dorigo consistently providing his support. Also, he has been patient enough to read all my papers and significantly contribute to their clarity and quality. Thanks to his high standards, I have learned a lot about high quality research.

From all the other members of IRIDIA, I would like to thank in particular Dr. Michael Sampels, Dr. Christian Blum, Dr. Joshua Knowles, Dr. Thomas Stützle, and Dr. Mauro Birattari. Through numerous discussions and working together, I have learned a lot about metaheuristics, optimization, statistics, as well as paper-writing. The collaboration and discussions with Dr. Philip Kostuch, Max Manfrin, and Prasanna Balaprakash has also had an important influence for my research. During the *Metaheuristics Network* project, I have also had the opportunity and pleasure to work with Prof. Ben Paechter, Dr. Olivia Rossi-Doria, both of Napier University, and Dr. Marco Chiarandini of Darmstadt University of Technology. Of course, I must also thank all the other members of both IRIDIA and STC, who have always been very supportive and an ensured excellent working atmosphere.

Last, (but definitely not least!), I would like to thank my family. In particular, I want to thank my wife Aga for being very patient and supportive. I could always rely on her in case of doubts, and she was the one that convinced me that I should really give it a try in the first place.Definitelly, this thesis would have never been finished, if it was not for her! Additionally, I would like to thank her also for preparing some of the figures that I use in this thesis. I also thank my son Emil for providing the necessary motivation for finally finishing the thesis. Without him, it would have taken even longer! I would like also to thank very much my parents, who have always believed that I can do it.

Contents

"Writing in English is the most ingenious torture ever devised for sins committed in previous lives."
— James Joyce

Chapter 1

Introduction

In the early days of computer science, the concept of *algorithm* was formalized through the works of Alan Touring [Turing, 1936] and Alonzo Church [Church, 1936], as *a procedure for solving mathematical problems.* Since then, many algorithms have been proposed and developed. One of the fundamental goals of computer science has been defining algorithms that are able to find provably *optimal* solutions (so called *exact* algorithms) within *reasonable run times.* However, it was observed that for some problems such algorithms could not be found.

A branch of computer science, called *computational complexity theory*, focused on investigating how well algorithms scale with increasing size of the problems. The scaling has been defined by the amount of the resources (e.g., memory, execution time) needed by an algorithm to solve a given problem. The research led to discovering that algorithms scale differently for different types of problems. The so-called *complexity classes* have been defined to describe the types of problems that demonstrate similar scaling characteristics. In particular, the complexity class P identifies the problems, which may be solved by a *deterministic machine in polynomial time.* These types of problems may be usually successfully tackled with exact algorithms even for large problem instances.

Another complexity class, NP, describes the problems that may be solved in polynomial time only by *non-deterministic machine.* The problems belonging to this class have the property that a solution may be easily evaluated. It remains an open question whether the set of P and NP are the same, or P is just a subset of NP. Many problems exist, for which a polynomial-time algorithm has not yet been found. For larger instances of such problems, the exact algorithms become impractical due to excessive resources (time and/or memory) required to tackle them. For an in-depth introduction to complexity theory aspects and history, we refer the reader to [Spiser, 2005].

However, for many real-world optimization problems, it is not necessary to guarantee finding an optimal solution. Often it is sufficient to find a reasonably good (or *approximate*) solution using limited resources available. Hence, researchers tried relaxing some of the requirements posed initially for the algorithms—*approximate methods* were born.

Approximate methods employ various strategies for finding good solutions. They are often dependent on particular characteristics of the problem being solved. The name chosen to describe them was *heuristics*, from the Greek word *heurisko*, which means "I find". A heuristic is usually defined as *a replicable method or approach for directing one's attention in learning, discovery, or problem solving.* Originally, the heuristics developed in computer science were very problem-, or even instance-dependent. Further research led to creation of more robust, general methods so that they may be applicable for solving various different problems. These more general and improved heuristic methods were called *metaheuristics.*

Since these early days, there have been many metaheuristics proposed in the literature. It is worth mentioning evolutionary algorithms (EA) [Fogel et al., 1966; Holland, 1975; Schwefel, 1981; Fogel, 1995], tabu search (TS) [Glover, 1989, 1990], simulated annealing (SA) [Kirkpatrick et al., 1983; Cemy, 1985], iterated local search (ILS) [Ramalhinho-Lourenço et al., 2002], and ant colony optimization (ACO) [Dorigo, 1992; Dorigo and Stützle, 2004].

Indeed, one of the most recent of them, and a very actively developed one, is ant colony optimization. ACO was inspired by the ants' foraging behavior [Dorigo, 1992]. Through the observation of the behavior of real ants, a combinatorial optimization algorithm was proposed. The algorithm was found to be an efficient method for tackling various combinatorial optimization problems (COPs). Examples of COPs include scheduling, vehicle routing, timetabling, and others. These problems are characterized in particular by the fact that the solution is created from a finite set of available components. We define the combinatorial optimization problems more formally in Chapter 2. Also, a more in-depth description of the ant colony optimization metaheuristic for combinatorial problems is presented in Chapter 3 of this thesis.

However, real-world problems may not always be described as combinatorial problems. Metaheuristics often focus on these type of problems, as they are easier to define and tackle using numerical methods and digital computer. In the real-world often some or all of the decision variables are continuous—i.e., able to assume any real value within a certain range. This results in either *continuous*—if all variables are continuous—or *mixed-variable* problems, if only some of them are continuous. For such problems, algorithms dedicated to combinatorial optimization may not be used efficiently. Also, these problems require different representation of the variables. The continuous variables are usually represented in digital machines using the floating point representation. While it generally allows for easier handling of real values, it is in fact also only an approximation, as the digital computers are not able to effectively represent real values.

There has been a significant amount of research done in the direction of either proposing new metaheuristics which can handle these types of problems, or adapting existing ones. While there are some metaheuristics that have been developed from the beginning for continuous (or mixed-variable) problems, many have been initially developed to handle combinatorial optimization problems only. These were then often adapted so that they could also tackle the continuous and (in some cases) mixed-variable optimization problems.

Since the emergence of ACO as a combinatorial optimization tool, attempts have also been made to use it for tackling continuous problems. However, applying the ACO metaheuristic to continuous domains was not straightforward, and the methods proposed often took inspiration from ACO, but did not follow it exactly. Contrary to those earlier approaches, this thesis presents a way to extend ACO to continuous and mixed-variable domains without the need to make any major conceptual change to its original structure.

Continuous and mixed-variable optimization is hardly a new research field. There exist numerous algorithms that were proposed for tackling these types of problems. In order to have a proper perspective on the performance of our proposed algorithms, we compare them not only to other ant-related methods, but also to other metaheuristics and other algorithmic approaches such as direct search methods, derivative methods, and others.

In this thesis, we provide a clear description of the proposed extension of ACO first to continuous domains and then to mixed-variable domains. We discuss the design decisions made, and explain how the proposed algorithms compare to others known in the literature. We run multiple experiments on typical benchmark problems, but also on example real-world problems. Finally, we show that ACO extended to continuous and mixed-variable optimization is a competitive approach.

1.1 Structure of the Thesis

This introductory Chapter 1 is followed by Chapters 2 and 3, which provide additional background material needed for the reader to fully understand the main body of the thesis.

Chapter 2 introduces the notion of *optimization*. It also defines formally the combinatorial, continuous, and mixed-variable optimization problems using the same common framework. It explains what are the characteristics of these different problems, and how this impacts the design of algorithms that are used to tackle them. It provides examples of both: the problems and the popular methods for solving them, giving additionally a general classification of the methods used.

Chapter 3 provides an overview of the ant colony optimization metaheuristic. First, the inspiring biological roots are presented. Several experiments with real ants are described. Then, it is presented how an optimization algorithm metaphor may be drawn from the behavior of real ants. The basic ideas of ant colony optimization are presented, followed by examples of practical implementations of ACO algorithms.

Following these introductory chapters, the main contribution of this thesis is presented in Chapters 4, 5, and 6. First, in Chapter 4, the fundamental ideas of the extension of ant colony optimization to continuous domains are presented. Then, a practical implementation of an algorithm is described, followed by the results of several experiments and comparisons. The proposed algorithm is compared to many others on several benchmark test functions. Chapter 5 continues the empirical evaluation of the proposed algorithm. It presents a practical application of an ACO algorithm extended to continuous domains on a problem of neural network training for pattern recognition in the medical field. In the process, the proposed continuous algorithm is compared to genetic algorithms and dedicated derivative-based methods for neural network training.

Chapter 6 explains how the proposed ACO algorithm may be further adapted to be able to handle simultaneously both continuous and discrete variables. We also discuss different types of formulation of the mixed-variable problems, and we show how this impacts the choice of a potential algorithm to tackle them. We propose a new benchmark problem for evaluating performance of native mixed-variable optimization algorithms in a controlled environment. Finally, we demonstrate the performance of the mixed-variable ACO on several real-world engineering problems and compare the results obtained to those reported in the literature.

Finally, Chapter 7 wraps up the main body of the thesis and provides a summary of the contributions and an analysis of the results obtained. It also gives an idea of future research directions.

Following the main body of the thesis, there are also several appendixes that complement the research reported in the thesis and provide additional details, which will make it easier to replicate the results obtained.

Appendix A explains in more detail, how the proposed ACO algorithm for continuous domains handles the possible correlation among decision variables. Additionally, Appendix B presents an alternative way of describing the proposed $\text{ACO}_{\mathbb{R}}$ algorithm, one that was used in some initial publications [Socha and Dorigo, 2008; Socha and Blum, 2006, 2007]. Last, Appendix C provides overview of the code of the algorithms developed in the course of the research. This allows an interested reader to continue experimentation and (hopefully) further improve the performance.

Chapter 2

Discrete, Continuous, and Mixed-Variable Optimization

The thesis aims at presenting a novel approach to tackling continuous and mixed-variable optimization problems with ant colony optimization. Before presenting the main topic of the thesis, it is crucial to define what we mean by optimization in general, why it makes sense to differentiate between discrete, continuous and mixed-variable optimization, and finally why ant colony optimization may be used for tackling these types of problems.

From a purely mathematical perspective, an *optimization problem* may be defined as follows [Boyd and Vandenberghe, 2004]:

Definition 2.1 *Given a function* $f : \mathbf{S} \rightarrow \mathbb{R}$*, find* $\mathbf{X}^* \in \mathbf{S} : \forall_{\mathbf{X} \in \mathbf{S}}\ f(\mathbf{X}^*) \leq f(\mathbf{X})$ *(minimization) or* $f(\mathbf{X}^*) \geq f(\mathbf{X})$ *(maximization).*

Function f *is called the* objective function*, its domain* $\mathbf{S}$ *is called the* search space*, and the elements of* $\mathbf{S}$*, are called* feasible solutions*. A feasible solution* $\mathbf{X}$ *is a vector of* optimization variables $\mathbf{X} = \{X_1, X_2, ..., X_n\}$*. A feasible solution* $\mathbf{X}^*$ *that minimizes (maximizes) the objective function is called an* optimal solution*.*

Note that maximization over an objective function f is equivalent to minimization over the function $-f$. Hence, without any loss of generality, in the remainder of this work we will limit ourselves to the case of minimization.

In general, there exist several families (or classes) of optimization problems, which are characterized by particular forms of the objective functions, search spaces, or constraints. In this work, we use the division into classes based on how the optimization problems differ in the definition of their search spaces.

We consider three types of optimization problems depending on the characteristic of the search space:

- *discrete optimization problems*—problems in which all the optimization variables $X_i, i = 1, ..., n$ are discrete, i.e., belong to a countable set, $X_i \in \mathbf{D}_i, i = 1, ..., n$;
- *continuous optimization problems*—problems in which all the optimization variables $X_i, i = 1, ..., n$ are continuous, $X_i \in \mathbb{R}, i = 1, ..., n$;
- *mixed-variable optimization problems*—problems in which p out of $n = p + q$ variables are discrete, $X_i \in \mathbf{D}_i, i = 1, ...p$, and q are continuous $X_i \in \mathbb{R}, i = p + 1, ..., p + q$.

In the remaining part of the thesis we often mention *combinatorial optimization problems*. Combinatorial optimization problems are in fact a subset of *discrete* optimization problems characterised by finite size of their domain. Since there are many problems of this type, and several algorithms have been proposed to deal exclusively with such combinatorial problems (notably, the original formulation of ACO), we focus on this type of discrete problems.

This classification is based on the fact that solving these three different classes of problems poses different difficulties and often requires different approaches. While some methods work well on continuous optimization problems, they may not be used for combinatorial optimization problems, and *vice versa*. In turn, mixed-variable optimization problems pose yet another type of difficulties due to the fact that they combine both discrete and continuous optimization variables.

It is important to note that it is not always possible to find the optimal solution to a given problem. In the case of combinatorial optimization, often, in order to ensure that a candidate solution is an optimal one, it must be compared to all other possible solutions. Unfortunately, in some cases the number of available solutions is (although finite) very large, and hence it is impossible (or at least impractical) to ensure that a given solution is the optimal one. Similarly, in the case of continuous optimization problems, it may be impossible to find an exact solution with analytical methods.

In such cases, the aim is to find a *near-optimal* solution, i.e., a solution that is *sufficiently close* to the optimal solution. What exactly *sufficiently close* means, depends on the problem at hand. In those cases, in which exact or analytical methods may not be used or their use is impractical, the most common approach is to use the *metaheuristics* [Glover and Kochenberger, 2003] such as evolutionary algorithms (EAs) [Fogel, 1995], simulated annealing (SA) [Kirkpatrick et al., 1983], tabu search (TS) [Glover, 1989, 1990], ant colony optimization (ACO) [Dorigo and Stützle, 2004], and others. These methods often allow finding near-optimal solutions in reasonable time.

The remainder of this chapter is divided into three main sections, each focusing on one of the three different classes of optimization problems, as they have been listed above: Section 2.1 focuses on combinatorial optimization, Section 2.2 deals with continuous optimization, and finally Section 2.3 summarizes the mixed-variable optimization problems. We describe the fundamental characteristics of the different optimization problem classes, give examples of specific real-world applications, and shortly present the various methods used for tackling these types of problems. Section 2.4 concludes the chapter.

2.1 Combinatorial Optimization

The name given to this type of optimization problems, i.e., *combinatorial*, comes from the fact that such problems may be expressed as those of finding a *permutation* or *combination* of a finite set of elements. Combinatorial optimization problems are therefore characterized by a finite set of possible solutions. This of course does not imply that they must be easy to solve, as the size of the set of solutions may be very large.

Following the definition of the optimization problem in the previous section, a model of a *combinatorial optimization problem* (COP), may be formally defined as follows:

Definition 2.2 *A model $P = (\mathbf{S}, \mathbf{\Omega}, f)$ of a COP consists of:*

- *a search space $\mathbf{S}$ defined over a finite set of discrete decision variables and a set $\mathbf{\Omega}$ of constraints among the variables;*
- *an objective function $f : \mathbf{S} \rightarrow \mathbb{R}$ to be minimized.*

The search space $\mathbf{S}$ is defined as follows: Given is a set of discrete variables X_i, $i = 1, ..., n$, with possible values $v_i^j \in \mathbf{D}_i = \{v_i^1, ..., v_i^{|\mathbf{D}_i|}\}$. A solution $s \in \mathbf{S}$—i.e., a complete assignment in which each decision variable has a value assigned—that satisfies all the constraints in the set $\mathbf{\Omega}$, is a feasible solution of the given COP. If the set $\mathbf{\Omega}$ is empty, P is called an unconstrained problem model, otherwise it is said to be constrained. A solution $s^* \in \mathbf{S}$ is called a global optimum if and only if: $f(s^*) \leq f(s)\ \forall_{s \in \mathbf{S}}$. The set of all globally optimal solutions is denoted by $\mathbf{S}^* \subseteq \mathbf{S}$. Solving a COP requires finding at least one $s^* \in \mathbf{S}^*$.

2.1.1 Applications

Combinatorial optimization is the process of finding one or more optimal solutions in a well defined discrete problem space. Such problems occur in almost all fields of management (e.g., finance, marketing, production, scheduling, inventory control, facility location and layout, and database management), as well as in many engineering disciplines (e.g., optimal design of waterways or bridges, VLSI-circuitry design and testing, the layout of circuits to minimize the area dedicated to wires, design and analysis of data networks, solid-waste management, determination of ground states of spin-glasses, determination of minimum energy states for alloy construction, energy resource-planning models, logistics of electrical power generation and transport, the scheduling of lines in flexible manufacturing facilities, and problems in crystallography).

The versatility of application of the combinatorial optimization model stems from the fact that in many practical problems, activities and resources, such as machines, airplanes and people, are indivisible. Also, many problems have only a finite number of alternative choices and consequently can appropriately be formulated as combinatorial optimization problems.

Many optimization problems can be represented by a network (or graph) that is defined by nodes and by edges connecting those nodes. Many practical problems arise around physical networks such as city streets, highways, rail systems, communication networks, and integrated circuits. In addition, there are many problems that can be modeled as networks even when there is no underlying physical network. For instance, one can think of the assignment problem where one wishes to assign a set of machines to a set of jobs in a way that minimizes the cost of the assignment. Here, one set of nodes represents the machines to be assigned, another set of nodes represents the possible jobs, and there is an edge connecting a machine to a job if that machine is capable of performing the given job.

In addition, there are many graph-theoretic problems that examine the properties of the underlying graph or network. Such problems include the traveling salesman problem, where one wishes to find a shortest Hamiltonian cycle in a graph. Other graph problems include the vertex coloring problem, the object of which is to determine the minimum number of colors needed to color each vertex of a graph in order that no pair of adjacent nodes share the same color; the maximum clique problem, whose goal is to find the largest subgraph of the original graph such that every node is connected to every other node in the subgraph; and the minimum cut problem, whose goal is to find a minimum weight collection of edges that (if removed) would disconnect a set of nodes s from a set of nodes t.

Although these combinatorial optimization problems on graphs might appear, at first glance, to be interesting mathematically but to have little application to the decision making in management or engineering, their domain of applicability is extraordinarily broad. For example, the traveling salesman problem has applications in routing and scheduling, in large scale circuitry design and in strategic defense, while both the clique problem and the minimum cut problem have important implications for the reliability of large systems. An overview of combinatorial optimization problems may be found in [Cook et al., 1998].

2.1.2 Methods

Solving combinatorial optimization problems, i.e., finding an optimal solution to such problems, can be a difficult task. The difficulty arises from the fact that unlike linear programming, for example, whose feasible region is a convex set, in combinatorial problems one must search a lattice of feasible points. Thus, unlike linear programming, where, due to the convexity of the problem, we can exploit the fact that any locally optimal solution is a global optimum, combinatorial optimization problems have many local optima and finding a global optimum to the problem requires one to prove that a particular solution dominates all feasible points by arguments other than the calculus-based derivative approaches of convex programming.

In general, we may divide the methods used to tackle the combinatorial optimization problems into the following two groups:

- *exact methods*—methods that guarantee finding an optimal solution to a given problem (but not always in reasonable time); and
- *heuristic methods*—methods that usually do not guarantee finding an optimal solution, but usually provide a *near-optimal* one in reasonable time.

In the following two subsections, we shortly present some of the most popular exact and heuristic methods for tackling combinatorial optimization problems.

Exact Methods

There are (at least) three different approaches for solving combinatorial optimization problems exactly, although they are frequently combined into *hybrid* procedures in computational practice:

- enumerative techniques,
- relaxation and decomposition techniques,
- cutting planes approaches based on polyhedral combinatorics.

Enumerative approaches: The simplest approach to solving a pure combinatorial optimization problem is to enumerate all finitely many possibilities. However, due to the *combinatorial explosion* with the increase of the problem size, only the smallest instances could be solved by such an approach. Sometimes one can implicitly eliminate many possibilities by domination or feasibility arguments. Besides straight-forward or implicit enumeration, the most commonly used enumerative approach is called *branch-and-bound*, where the *branching* refers to the enumeration part of the solution technique and *bounding* refers to the fathoming of possible solutions by comparison to a known upper or lower bound on the solution value [Land and Doig, 1960]. To obtain an upper bound on the problem, the problem is relaxed in a way which makes the solution to the relaxed problem, relatively easy to find.

All commercial branch-and-bound algorithms relax the problem by dropping the integrality conditions and solve the resultant continuous linear programming problem over the set P. If the solution to the relaxed linear programming problem satisfies the integrality restrictions, the solution obtained is optimal. If the linear program is infeasible, then so is the integer program. Otherwise, at least one of the integer variables is fractional in the linear programming solution. One chooses one or more such fractional variables and *branches* to create two or more subproblems, which exclude the prior solution but do not eliminate any feasible integer solutions. These new problems constitute *nodes* on a branching tree, and a linear programming problem is solved for each node created. Nodes can be fathomed if the solution to the subproblem is infeasible, satisfies all of the integrality restrictions, or has an objective function value worse than a known integer solution.

Lagrangian Relaxation and Decomposition Methods: Relaxing the integrality restriction is not the only possible approach to relaxing the problem. An alternative approach is to take a set of *complicating* constraints into the objective function in a Lagrangian fashion (with fixed multipliers that are changed iteratively). This approach is known as Lagrangian relaxation [Fisher, 1981]. By removing the complicating constraints from the constraint set, the resulting sub-problem is frequently considerably easier to solve. The latter is a necessity for the approach to work because the subproblems must be solved repetitively until optimal values for the multipliers are found. The bound found by Lagrangian relaxation can be tighter than that found by linear programming, but only at the expense of solving subproblems in integers, i.e., only if the subproblems do not have the integrality property[1]. Lagrangian relaxation requires that one understands the structure of the problem being solved in order to then relax the constraints that are *complicating.* A related approach, which attempts to strengthen the bounds of Lagrangian relaxation, is called Lagrangian decomposition [Guignard and Kim, 1987]. This method consists of isolating sets of constraints so as to one obtains separate, easy problems to solve over each of the subsets. The dimension of the problem is increased by creating linking variables which link the subsets. All Lagrangian approaches are problem dependent and no underlying general theory has evolved.

Since each of the decomposition approaches described above provide a bound on the integer solution, they can be incorporated into a branch-and-bound algorithm, instead of the more commonly used linear programming relaxation. However, these algorithms are special-purpose algorithms in that they exploit the *constraint pattern* or special structure of the problem.

Cutting Plane algorithms based on polyhedral combinatorics: In 1935, Weyl established the fact that a convex polyhedron can alternatively be defined as the intersection of finitely many half spaces or as the convex hull plus the conical hull of some finite number of vectors or points. If the data of the original problem formulation are rational numbers, then Weyl's theorem implies the existence of a finite system of linear inequalities whose solution set coincides with the convex hull of the mixed-integer points in S which we denote $conv(S)$. Thus, if we can list the set of linear inequalities that completely define the convexification of S, then we can solve the integer programming problem by linear programming. Gomory derived in 1958 a *cutting plane* algorithm [Gomory, 1958] for integer programming problems which can be viewed as a constructive proof of Weyl's theorem, in this context.

More recently a method called *branch-and-cut* [Padberg and Rinaldi, 1991] has been derived based on those earlier ideas. The major components of this algorithm con-

[1]A problem has the integrality property if the solution to the Lagrangian problem is unchanged, when the integrality restriction is removed

sist of automatic reformulation heuristic procedures, which provide *good* feasible integer solutions, and cutting plane procedures which tighten the linear programming relaxation to the combinatorial problem under consideration—all of which is embedded into a tree-search framework as in the branch-and-bound approach to integer programming. Whenever possible, the procedure permanently fixes variables (by reduced cost implications and logical implications) and does comparable conditional fixing throughout the search-tree. These four components are combined so as to guarantee optimality of the solution obtained at the end of the calculation. However, the algorithm may also be stopped early to produce suboptimal solutions along with a bound on the remaining error. The cutting planes generated by the algorithm are facets of the convex hull of feasible integer solutions or good polyhedral approximations thereof and as such they are the *tightest cuts* possible. Lifting procedures assure that the cuts generated are valid throughout the search tree which aids the search process considerably and is a substantial difference to traditional (Gomory) cutting-plane approaches.

Metaheuristics

The heuristics solution approaches are the techniques for obtaining *good* but not necessarily optimal solutions to combinatorial optimization problems quickly and, in general, without any guarantee as to their *closeness* to an optimal solution. Heuristics are, however, important for a variety of reasons. They may provide the only usable solution to very difficult optimization problems for which the current exact algorithms are incapable of providing an optimal solution in reasonable times. When heuristics are used within an exact algorithm, they provide a bound to fix variables and to define branches on a search-tree.

The main general heuristic methods—known as metaheuristics—often used for tackling combinatorial optimization problems include evolutionary algorithms (EAs) [Fogel, 1995], simulated annealing (SA) [Kirkpatrick et al., 1983], tabu search (TS) [Glover, 1989, 1990], ant colony optimization (ACO) [Dorigo and Stützle, 2004], and others. We give here a brief description of the more popular methods.

Evolutionary Algorithms: They are a class of population-based metaheuristic optimization algorithms, which use mechanisms inspired by biological evolution, such as natural selection, mutation, and recombination [Fogel, 1995]. Each member of the population represents a candidate solution to the optimization problem. The fitness function is used to define the quality of each such solution. Specific evolutionary algorithms, such as genetic algorithms (GA) [Goldberg, 1989], genetic programming (GP) [Koza, 1992], or evolution strategy (ES) [Beyer and Schwefel, 2002] differ in the way they implement various *genetic operators* (such as for instance selection, cross-over, or mutation) in order to *evolve* the population towards *better* solutions.

We shortly present the basic mechanism of operation of evolutionary algorithms using as an example a genetic algorithm. The set of solutions of the problem are stored in the form of population. Each member of the population stores information about a single (but complete) solution to a given problem. The solution is encoded—in case of genetic algorithms, usually a binary string is used for encoding the solution. From the population of available solutions, an algorithm chooses a subset of solutions with the use of a selection operator. These selected solutions are then used for deriving new solutions through either mutation or recombination operators. The recombination operator in case of genetic algorithms is usually realized as a form of cross-over. Two (or more) solutions from the *parent* population are *crossed-over* in order to create two (or more) *children*. The mutation operator works on single elements of the population. It allows to introduce small variations in the values of solutions—usually with very small probability. Finally, the subset of individuals from among the old population and the set of newly generated ones, is kept as the new population.

Estimation of Distribution Algorithms: Algorithm first proposed by Mühlenbein and Paaß [Mühlenbein and Paaß, 1996]. Although usually estimation of distribution algorithms (EDAs) are considered to be part of the family of evolutionary algorithms, they differ sufficiently from all the others that it makes sense to present them separately. In EDA the two parents recombination process is replaced by generating new solutions according to the probability distribution of all promising solutions of the previous generation.

In EDAs the problem specific interactions among the variables of individuals are taken into consideration. In evolutionary algorithms in general, the interactions are not directly addressed, whereas in EDAs the interrelations are expressed explicitly through the joint probability distribution associated with the individuals of variables selected at each generation. The probability distribution is calculated from a database of selected individuals of previous generation. The selection methods used in genetic algorithms may be used here. Then sampling this probability distribution generates offspring. Neither crossover nor mutation is applied in EDAs.

Simulated Annealing: An algorithm independently invented by S. Kirkpatrick, C.D.

Gelatt and M.P. Vecchi in 1993 [Kirkpatrick et al., 1983], and by V. Cerny in 1985 [Cerny, 1985]. The inspiration (as well as the name) comes from the process of *annealing* known in metallurgy. It is a technique of heating and controlled cooling of materials to increase size of crystals and reduce their defects. The heating causes the atoms in the material to obtain more kinetic energy, and hence be able to move away from their positions of local minimal energy. The slow cooling gives them a chance to find a low internal energy state.

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random *neighborhood* solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called by analogy—the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly *downhill* as T approaches zero. The allowance for *uphill* moves saves the method from becoming stuck in local minima.

Tabu Search: An algorithm generally attributed to Glover [Glover, 1989, 1990]. Tabu search uses a local or neighborhood search procedure to iteratively move from a solution $\mathbf{x}$ to a solution $\mathbf{x}'$ in the neighborhood of $\mathbf{x}$, until some stopping criterion has been satisfied. To explore regions of the search space that would be left unexplored by the local search procedure and—by doing this—escape local optimality, tabu search modifies the neighborhood structure of each solution as the search progresses. The solutions admitted to $N^*(\mathbf{x})$, the new neighborhood, are determined through the use of a special memory structure—the tabu list. The search progresses by iteratively moving from a solution x to a solution x' in $N^*(\mathbf{x})$. Solutions in the tabu list are excluded from $N^*(\mathbf{x})$. Other tabu list structures prohibit solutions that have certain attributes (e.g., traveling salesman problem (TSP) solutions that include certain arcs) or prevent certain moves (e.g., an arc that was added to a TSP tour cannot be removed in the next n moves). Selected attributes in solutions recently visited are labelled tabu-active. Solutions that contain tabu-active elements are tabu. This type of short-term memory is also called recency-based memory.

Iterated Local Search: According to its name, it is an algorithm that iteratively uses a certain local search technique [Ramalhinho-Lourenço et al., 2002]. The main idea is the change of the search space. Instead of searching in the space of all possible solutions, the algorithm searches in the space of locally optimal solutions. Iterated local search (ILS) generates at each iteration a starting solution, and then uses an additional local search mechanism, which returns locally optimal solutions. The returned solution is used to create a new starting solution by the ILS. The new starting solution is usually created through a *perturbation*—a modification of the given solution.

The nature of the perturbation, as well as the specific local search used, depend on the problem at hand. ILS, in spite of its apparent simplicity, has been shown to perform quite well on number of problems.

Ant Colony Optimization: Finally, one of the metaheuristics used for tackling combinatorial optimization problems is ant colony optimization. Since this algorithm is the main topic of this work, a comprehensive introduction to ACO is presented in a separate chapter (Chapter 3).

2.1.3 Choosing the Suitable Method

Each of the two main types of methods presented in the previous section has its advantages and disadvantages. These must be taken into consideration when choosing a method to tackle a given problem. In this section, we shortly compare the main advantages and disadvantages of these methods. We also discuss which characteristics of the problem, as well as other conditions related to the situation in which the problem is solved, have influence on the choice of a particular method.

When choosing a method to tackle a given combinatorial optimization problem, several issues must be taken into consideration. On one hand, the characteristics of the methods have to be taken into account, such as, for instance:

- how quick it gives the results,
- what is the quality of the results,
- whether the method provides any guarantee about the quality of the solution found,
- how difficult it is to implement,
- how well does it scale with problem size,
- etc.

On the other hand, one has to also consider the characteristics of the problem at hand, as well as the situation in which the problem is solved:

- what is the size of the problem,

- what solution quality is needed,
- how much time is there to implement the method,
- how much time is there to run the algorithm to obtain the results,
- etc.

Only after analyzing the characteristics of the methods available and the given problem formulation, as well as defining the situation in which the problem is to be solved, one may be able to choose the method that is most suitable.

In the following, we present the main characteristics of the two main types of methods for tackling combinatorial optimization problems, i.e, the exact methods and the metaheuristics.

Advantages and Disadvantages

While the exact methods, as the name suggests, allows to solve the combinatorial problems exactly, metaheuristics give a solution that is close to the optimum. The exact methods guarantee finding the optimum solution, however the time required to find it is usually quite long. In turn, metaheuristics do not guarantee anything about the quality of the solutions found, but are able to find *reasonably good* solutions in a small amount of time.

The term *reasonably good* is rather vague, and in most cases it can not be defined precisely. How good is in fact the solution found by a metaheuristic depends on several factors. Among others, it depends on: the type of the problem, its size, the amount of time available to find the solution, and the actual implementation of the metaheuristic used. Nevertheless, metaheuristics are often used to tackle combinatorial optimization problems. The main reasons for this is that either the time available to solve a combinatorial problem is very short, or the dimension of the problem is such that the use of exact methods is impractical. Also, in the majority of the real world situations a *reasonably good* solution is sufficient. There is usually no need to have provably optimum solution. However, if this is indeed what is needed, than exact methods should be used.

2.2 Continuous Optimization

Similarly to a combinatorial optimization problem (COP), also a model for *continuous optimization problem* (CnOP) may be formally defined:

Definition 2.3 *A model $Q = (\mathbf{S}, \mathbf{\Omega}, f)$ of a CnOP consists of:*

- *a search space $\mathbf{S}$ defined over a finite set of continuous decision variables and a set $\mathbf{\Omega}$ of constraints among the variables;*
- *an objective function $f : \mathbf{S} \rightarrow \mathbb{R}$ to be minimized.*

The search space $\mathbf{S}$ is defined as follows: Given is a set of continuous variables $X_i,\ i = 1, ..., n$ with possible values $v_i \in \mathbf{D}_i \subseteq \mathbb{R}$. A solution $s \in \mathbf{S}$—i.e., a complete assignment, in which each decision variable has a value assigned—that satisfies all the constraints in the set $\mathbf{\Omega}$, is a feasible solution of the given CnOP. If the set $\mathbf{\Omega}$ is empty, Q is called an unconstrained problem model, otherwise it is called a constrained one. A solution $s^* \in \mathbf{S}$ is called a global optimum if and only if: $f(s^*) \leq f(s)\ \forall_{s \in \mathbf{S}}$. The set of all globally optimal solutions is denoted by $\mathbf{S}^* \subseteq \mathbf{S}$. Solving a CnOP requires finding at least one $s^* \in \mathbf{S}^*$.

The main difference between the combinatorial and continuous optimization problems, is the fact that the search space is not finite. Each of the continuous decision variables may assume an infinite number of values. Of course, solving such problems using computers imposes certain limitations, as computers—being digital in nature—can only represent a finite number of values. One could think that this reduces the problem to a combinatorial optimization one. This is however not really the case. In the case of combinatorial optimization, the set of available values is predefined before starting the optimization. The size of this set has a significant influence on the difficulty of finding an optimal or near-optimal solution. In case of continuous optimization problems, the algorithms use an efficient and flexible floating point representation of real-valued variables. This allows finding solutions with the required accuracy in a more efficient way.

2.2.1 Applications

The area of applications for continuous optimization is very wide. Many real-world problems and processes may be presented in the form of a continuous optimization problem. Typical examples include designing optimal shapes (such as wings, turbines, and others) or choosing values of continuous parameters for various industrial processes (e.g., temperature, pressure, etc.). An example presented in Chapter 5 illustrates how a continuous optimization algorithm may be used to train artificial neural network, which in turn is used for medical diagnosis. Of course, continuous optimization algorithms are also often used to tackle functions that may be defined using relatively simple mathematical formulas. These may vary from simple test functions, to complex mathematical descriptions of various processes. Examples include engine design, power plants design, or computer simulations of many other processes.

It is important to notice that certain problems may be transformed from combinatorial to continuous form by relaxing the requirement of having the values from a finite set. In the evaluation of the objective function, the continuous values may be then rounded to the nearest value from the initial set. Similarly, it is also possible to transform the continuous problem into the combinatorial one by dividing the continuous domains into a set of values. This however is only possible if the continuous domain is bounded (i.e., the lower and upper bounds are defined).

2.2.2 Methods

Similarly to the case of combinatorial optimization problems, the algorithms available for solving continuous optimization problems may be divided into the exact and the approximate ones.

Analytical Approach

Solving a continuous optimization problem exactly means to minimize analytically a given objective function f. This is not always easy or possible, as analytical method requires certain conditions to be fulfilled. For multivariate functions[2] of the form $f(x_1, ..., x_n) = f(\mathbf{x}) \in \mathbb{R}$, $\mathbf{x} \in \mathbf{D}^n \subset \mathbb{R}^n$, the gradient $\bigtriangledown f = \frac{d}{dx} f$ and the Hessian $\mathbf{H} = \frac{d^2}{dx^2} f$ need to be found. They are given respectively by:

[2]In the case of a univariate function $f(x)$, simply its first and second derivatives, $f'(x)$ and $f''(x)$ need to be calculated.

$$\bigtriangledown f = \frac{d}{dx} f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \quad \text{and} \quad \mathbf{H} = \frac{d^2}{dx^2} f = \begin{bmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} f & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f & \cdots & \frac{\partial^2}{\partial x_n \partial x_n} f \end{bmatrix}. \tag{2.1}$$

The extreme points of the function f satisfy the condition $\bigtriangledown f(\mathbf{x}) = (0, \ldots, 0)$. However, in order to distinguish between the maxima, minima, and saddle points, it is necessary to find the signs of the eigenvalues of the Hessian $\mathbf{H} = \frac{d^2}{dx^2} f$. This is usually accomplished through eigenvalue/eigenvector decomposition of the Hessian[3]:

$$\mathbf{H} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \quad \text{with} \quad \mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n), \tag{2.2}$$

where matrix $\mathbf{U}$ contains the eigenvectors, and vector $\mathbf{\Lambda}$ contains the eigenvalues. Then, the type of extreme point depends on the sign of the eigenvalues in the following way:

- $\forall_{i \in \{1,\ldots,n\}} \lambda_i > 0 \rightarrow$ maximum,
- $\forall_{i \in \{1,\ldots,n\}} \lambda_i < 0 \rightarrow$ minimum,
- $\exists_{i,j \in \{1,\ldots,n\}} \lambda_i < 0 \wedge \lambda_j > 0, \rightarrow$ saddle point,
- else $\rightarrow$ further studies are necessary.

It is important to notice that such analytical approach is possible only if the gradient $\bigtriangledown f$ and the Hessian $\mathbf{H}$ can be found analytically. This is obviously possible only if the function f is given in the form of a mathematical formula, and it is *not* possible when, for instance, the function f is only known as *empirical data*, which is often the case in real-world problems.

Numerical Methods Based on Gradient Descent

Since, as mentioned earlier, symbolic computation is often not easy or practical, a number of methods have been proposed to perform numerical function minimization. Among them, there is a number of derivative-based methods, which we shortly present here.

[3]Note that such decomposition is only possible if $f \in \mathbb{C}^2$.

Steepest Descent: One of the simplest methods of numerical derivative-based function minimization is the steepest descent method, also often called the *gradient descent* [Arfken, 1985]. This method relays on the fact that the first derivative, i.e., the gradient $\bigtriangledown f$ (see Equation 2.1) can be calculated for chosen points $\mathbf{x} \in \mathbf{D}^n$.

In steepest descent, the search starts with a randomly chosen solution $\mathbf{x}$. In each iteration, the current solution $\mathbf{x}$ is replaced by a solution $\mathbf{x}'$:

$$\mathbf{x}' = \mathbf{x} - \epsilon \bigtriangledown f \tag{2.3}$$

where ϵ is a coefficient sufficiently small, so that $f(\mathbf{x}') < f(\mathbf{x})$. The value of the coefficient ϵ may change at each iteration step.

The steepest descent method is guaranteed to converge to a local optimum, but its convergence is sometimes quite slow.

Conjugate Gradient Method: This is a more advanced method when compared to the previous one. It uses *conjugate directions* instead of local gradient for going downhill. Successive one-dimensional minimizations are performed along conjugate directions with each direction being used only once per iteration [Bulirsch and Stoer, 1991]. The first direction is taken as in case of the steepest descent method:

$$d_0 = - \bigtriangledown f(\mathbf{x}_0) \tag{2.4}$$

Each subsequent conjugate direction is then calculated with the following formula:

$$d_{i+1} = - \bigtriangledown f(\mathbf{x}_{i+1}) + \frac{\bigtriangledown f(\mathbf{x}_{i+1})^T \bigtriangledown f(\mathbf{x}_{i+1})}{\bigtriangledown f(\mathbf{x}_i)^T \bigtriangledown f(\mathbf{x}_i)} d_i \tag{2.5}$$

The convergence of the conjugate gradients method is usually faster than the previously described steepest descent method.

Newton's Method: Isaac Newton has worked out his method in 1669. Since then, several versions of Newton's method have been proposed [Fujita and Yamaguti, 1981; Peitgen, 1989]. The basic one for finding a minimum of the function $f(\mathbf{x})$ requires the knowledge of both the gradient $\bigtriangledown f$ and the Hessian matrix $\mathbf{H}$ (see Equation 2.1). Instead of minimizing the objective function f directly, the Newton's method minimizes the quadratic polynomial approximation Q around the solution $\mathbf{x}_0$, assuming that the minimum exists as solution $\mathbf{x}_{min}$:

$$Q(\mathbf{x}) = f(\mathbf{x}_0) + \bigtriangledown f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2!}(\mathbf{x} - \mathbf{x}_0)\mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)^T \tag{2.6}$$

A minimum of $Q(\mathbf{x})$ occurs where $\bigtriangledown Q(\mathbf{x}) = (0, \ldots, 0)$, what can be written in the following way:

$$\bigtriangledown Q(\mathbf{x}) = \bigtriangledown f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)\mathbf{H}(\mathbf{x}_0) = (0, \ldots, 0) \tag{2.7}$$

Assuming that solution $\mathbf{x}_0$ is close to the solution $\mathbf{x}_{min}$ (where the minimum occurs), then $\mathbf{H}(\mathbf{x}_0)$ is invertible and the above equation may be solved for $\mathbf{x}$, which in turn can be used as solution $\mathbf{x}_1$—the next approximation of the solution $\mathbf{x}_{min}$:

$$\mathbf{x}_1 = \mathbf{x}_0 - \frac{\bigtriangledown f(\mathbf{x}_0)}{\mathbf{H}(\mathbf{x}_0)} \tag{2.8}$$

The advantages of the Newton method include quite fast convergence (quadratic convergence for quadratic functions) and better error approximation than the gradient methods. There are also some disadvantages. It requires calculation of second order derivatives and inversion of the Hessian matrix, which is usually quite computationally intensive. Also, the initial starting solution should be chosen reasonably close to the actual minimum. Alternatively, there are also the *quasi*-Newton methods, which use an approximation of the Hessian matrix [Brojden, 1967; Dennis Jr. and More, 1977].

Direct Search Methods

Following a short overview of the analytical approach to continuous function optimization and derivative-based approximation methods, we will now present the direct methods that do not require any additional information about the objective function f, apart from the ability to evaluate it at chosen points.

Simplex: An algorithm originally proposed by Nelder and Mead [Nelder and Mead, 1965]. *Simplex* is a generalized n-dimensional triangle. To minimize an n-dimensional function, the algorithm uses simplexes composed of $(n+1)$ points. Initially, the points for the simplex are chosen randomly. Then, at each iteration the simplex is modified by removing the point for which the value of the objective function f is the worst. Instead, a new point is added using the reflection, expansion, and contraction operators. Reflection creates a new point as a reflection of the worst point on the other side of the n-dimensional plane formed by all the simplex points with the exception of the worst one. Expansion causes the simplex to become bigger, and contraction—respectively—smaller.

The simplex algorithm, although quite simple, has proven to be a very robust method for local function minimization. It is currently rarely used alone, as there are other more powerful methods (such as the Powell's method described next). However, it is often used in combination with other methods (e.g., metaheuristics) to form hybrid approaches.

Powell's Method: In Powell's algorithm [Powell, 1964], the procedure begins at a starting point $\mathbf{x}_0$, and each iteration of the algorithm consists of $(n+2)$ successive exact line searches, that is exploration in a straight line. Initially, the first $(n+1)$ line searches are along the n coordinate axes. The $(n+2)$-nd line search goes from the point obtained from the first line search through the best point (obtained at the end of the $(n+1)$ line searches). If the function is quadratic, this will locate the optimum. If it is not, then the search is continued with one of the first n directions replaced by the $(n+1)$-th direction, and the procedure is repeated until a stopping criterion is met.

Powell has pointed out that this procedure requires modification if the acceleration directions become close to being linearly dependent. He reported that this possibility has been found to be serious if the function depends on more than five variables. Hence, a test should be performed before each new vector is to replace one of the previous n ones, in order to ensure that this does not happen.

This method has been called one of the most efficient and reliable of the direct search methods. The reason is its relative simplicity and quadratic termination property. It is worth to notice that the conjugate gradient method was proved to be as effective as Powell's method, but it requires the gradient information to work.

Metaheuristics

Last, but not least, metaheuristics have been also used for continuous optimization. One of the primary reasons to use metaheuristics for this purpose is the fact that they may be able to find a global optimum of the problem at hand, and not only the local one. While some of the algorithms presented earlier are quite efficient in locating local optima, none of them has been designed for global optimization. This is why metaheuristics become an interesting alternative.

There have been many metaheuristics proposed for tackling continuous optimization problems. In the previous section, we have already presented some of them. It is important to note here that while there were some metaheuristics that were developed with the continuous optimization in mind, most of them have been adapted to continuous optimization based on their counterparts initially developed for combinatorial optimization. Indeed, the main contribution of this work is such a metaheuristic—ant colony optimization applied to continuous optimization problems. Chapter 4 contains a more detailed discussion, how the algorithm proposed here compares to other popular metaheuristics.

Metaheuristics for continuous optimization are often hybridized with local search methods presented earlier in this section. This allows them to focus on global optimization, while the local search methods, such as gradient-based or direct search methods, help them in finding local optimums.

We now briefly present those metaheuristics that have been explicitly designed for continuous optimization.

Evolution Strategy: This method was developed by Rechenberg [Rechenberg, 1973] and Schwefel [Schwefel, 1981]. Initially the method was not intended to be used as an algorithm for computers, but rather as a method to find optimal parameter settings in laboratory experiments. Nowadays, this method is considered to be one of the evolutionary algorithms, which we already briefly presented. An interesting element of the design of evolution strategy is the fact that its parameters (i.e., the parameters of the algorithm itself) are optimized along with the so-called *object parameters*, i.e., the parameters of the process being optimized.

Each member of the population contains a vector of parameters being optimized. Each element of this vector is a real number, and each such element is either an object parameter o_p, or a strategy parameter s_p. Usually, the only parameter of the strategy is the step-size of the mutation—one for each object parameter. New solutions are created through element-wise mutation through addition of normally distributed random numbers. For each object parameter, a corresponding strategy parameter is used to describe the standard deviation of the normal distribution used:

$$\mathbf{o_p} \leftarrow \mathbf{o_p} + N(0, \mathbf{s_p}), \tag{2.9}$$

where $\mathbf{o_p}$ is a vector of object parameters, $\mathbf{s_p}$ is a vector of respective strategy parameters, and $N(0, s_p)$ is a normal distribution with $\mu = 0$, and $\sigma = s_p$.

There exist numerous versions of the evolution strategy, which differ mostly through the way the new solutions are chosen from the old population and the newly generated children. The first and simplest one, but also one that is still quite often used due to its simplicity and effectives, is the so-called (1+1)-ES. Its population consists of a single individual. A child is generated from this single individual through mutation. If the child is better than the parent—it replaces the parent, otherwise, the parent stays unchanged. More generally, λ mutants can be generated and compete with the parent, called $(1+\lambda)$-ES. In a $(1, \lambda)$-ES the best mutant becomes the parent of the next generation while the current parent is always disregarded. Contemporary derivatives of evolution strategy often use a population of μ parents and also recombination as an additional operator (called $(\mu/\rho, \lambda)$-ES). This is believed to make them less prone to get stuck in local optima.

Differential Evolution: This is another example of an evolutionary algorithm. Differential evolution has been proposed relatively recently by Storn and Price [Storn and Price, 1995, 1997]. It operates on a population of vectors, i.e., solutions to the given problem. Each vector consists of a series of real-valued parameters to be optimized. The methods works as follows. For each vector $\mathbf{x_i}, i = 1, \dots, n$ from the current population, a trial vector $\mathbf{v_i}$ is created:

$$\mathbf{v_i} = \mathbf{x_{r_1}} + F(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}), \tag{2.10}$$

where $r_1, r_2, r_3 \in (1, \dots, n)$ are random integer numbers that are mutually different, as well as different from i. F is a real-valued factor, which controls the amplification of differential variation.

Additionally, certain elements of each newly created vector $\mathbf{v_i}$ are replaced by elements of the original vector $\mathbf{x_i}$ to form the final vectors $\mathbf{u_i}$. This aims at increasing the diversity of the parameter vectors and is a mechanism similar to cross-over in other evolutionary algorithms. Finally, the resulting vectors $\mathbf{u_i}$ are compared to the original vectors $\mathbf{x_i}$ and only the better ones are retained.

Iterated Density Estimation Evolutionary Algorithms: Yet another instance of an evolutionary algorithm. Iterated density estimation evolutionary algorithm (IDEA) is an extension of the estimation of distribution algorithms (EDA) [Mühlenbein and Paaß, 1996], which were already presented in Section 2.1.2. The IDEA framework has been proposed by Bosman and Thierens [Bosman and Thierens, 1999, 2000] as a method for extending EDAs also to the continuous domain.

Similarly to EDA, in IDEA a population of individuals containing current solutions to the given problem is used to build a probability distribution, which in turn is used to generate the new population. In case of continuous optimization problems, the normal or *normal kernel* distribution is used. In the first case, an n-dimensional normal distribution is fitted using all the elements of the current population. In the second case, an n-dimensional normal distribution is built for each point from the current population and then all of these are superimposed to form the normal kernel distribution. While in case of using just a single normal distribution both parameters (i.e., μ and σ)[4] are fitted based on all the available solutions, in the case of normal kernel distribution, the procedure is more complex. The parameter μ of each normal distribution is set to the coordinates of the points making up the current population. The parameters σ are chosen per dimension, based on the range of values assumed in this dimension by all the solutions in the population. Once the probability distribution is established a new population is created through sampling this distribution.

Particle Swarm Optimization: This is a population-based stochastic optimization technique developed by Eberhart and Kennedy [Kennedy and Eberhart, 1995]. The inspiration for the algorithm was the social behavior of bird flocking and fish schooling.

Although particle swarm optimization (PSO) shares many similarities with evolutionary computation techniques, it is considered to be rather part of *swarm intelligence* methods. The system is initialized with a population of random solutions and searches for optima by updating subsequent generations. Unlike typical evolutionary algorithm,

[4]Note that both μ and σ are vectors of cardinality n.

PSO has no evolutionary operators such as cross-over or mutation. In PSO individuals called *particles* representing the solutions to the given optimization problem, fly through the problem search space by following certain attractors. These attractors may be represented by the best known position of the given particle, or the best position found globally by the whole swarm.

The PSO concept consists in iteratively changing (in discrete steps) the velocity vectors of the particles towards (usually) a combination of the particle best and global best locations. Additionally, a certain inertia may be added to the particle in order to improve performance.

2.2.3 Choosing the Suitable Method

As in the case of combinatorial optimization problems, each of the types of methods for continuous optimization has its advantages and disadvantages. These must be taken into consideration, when choosing a method to tackle a given problem. In this section, we shortly compare the main advantages and disadvantages of these methods.

Differently from what happens with combinatorial optimization problems, not all the methods available for continuous optimization may always be used. Their usefulness is subject to several limitations related to the way the problem is formulated, and depending whether the solution sought is a global or local optimum.

In the following, we present the main characteristics of the types of methods for tackling continuous optimization problems and explain when each of them can be used.

Advantages and Disadvantages

The analytical approach is the only approach that may yield exact results. It allows to find all minima and maxima, including also the global minimum and maximum, if they exist. However, this approach can only be used if the problem is formulated in the form of a mathematical formula and not, for instance, as a set of sampled empirical data. Unfortunately, this is a rare situation, when real-world problems are concerned.

All the other types of methods for continuous optimization presented in the previous sections allow to find only approximate results. The first of them, derivative-based methods, allow to reasonably quickly find local optimum. Unlike the analytical approach, the derivative-based methods use numerical function optimization, which requires only to be able to calculate the value of the function in any point of the search space instead of requiring the symbolic formula of the function. However, the derivative-based methods require also information on the derivatives of the optimized function—i.e., it must be possible to also evaluate the derivatives for any of the points of the search space. Depending on the actual method, it may be either just the first derivative, or the first and second one.

Direct search methods are more robust than the previous two, that is, they are able to deal effectively with a wider range of continuous optimization problems. They only require that the function may be evaluated in any point of the search space. They do not require any additional information. Their drawback is however the fact that, similarly to derivative-based methods, they only find a local optimum, and not the global optimum.

Finally, there are metaheuristics. As in the case of combinatorial optimization problems, metaheuristics for continuous optimization do not provide any guarantees about the quality of the solutions found. However, they may be able to look not only for local optima, but also for the global optimum. The metaheuristics also have little requirements about the problem formulation. Similarly to the direct methods, they only need to be able to evaluate the function in any point of the search space.

The ability of the metaheuristics to look for global optimum is often coupled with the speed of finding the local optima by the direct methods or derivative-based methods. The algorithms used in practice to tackle real-world continuous optimization problems are often constructed as hybrids of a metaheuristic and a direct or derivative-based method. An example of such approach is presented in Chapter 5, where the continuous ACO algorithm is used with derivative-based methods to optimize weights of a neural network.

2.3 Mixed-Variable Optimization

Based on the models of combinatorial and continuous optimization problems, a model for a *mixed-variable optimization problem* (MVOP) may be formally defined as follows:

Definition 2.4 *A model* $R = (\mathbf{S}, \mathbf{\Omega}, f)$ *of a MVOP consists of:*

- *a search space* $\mathbf{S}$ *defined over a finite set of both discrete and continuous decision variables and a set* $\mathbf{\Omega}$ *of constraints among the variables;*
- *an objective function* $f : \mathbf{S} \rightarrow \mathbb{R}_0^+$ *to be minimized.*

The search space $\mathbf{S}$ *is defined as follows: Given is a set of* $n = p + q$ *variables* X_i, $i = 1, ..., n$, *of which* p *are discrete with values* $v_i^j \in \mathbf{D}_i = \{v_i^1, ..., v_i^{|\mathbf{D}_i|}\}$, *and* q *are continuous with possible values* $v_i \in \mathbf{D}_i \subseteq \mathbb{R}$. *A solution* $s \in \mathbf{S}$*—i.e., a complete assignment in which each decision variable has a value assigned—that satisfies all the constraints in the set* $\mathbf{\Omega}$, *is a feasible solution of the given MVOP. If the set* $\mathbf{\Omega}$ *is empty,* R *is called an unconstrained problem model, otherwise it is said to be constrained. A solution* $s^* \in \mathbf{S}$ *is called a global optimum if and only if:* $f(s^*) \leq f(s)\ \forall_{s \in \mathbf{S}}$. *The set of all globally optimal solutions is denoted by* $\mathbf{S}^* \subseteq \mathbf{S}$. *Solving a MVOP requires finding at least one* $s^* \in \mathbf{S}^*$.

Mixed-variable optimization problems are hence a combination of combinatorial and continuous optimization problems. They are particularly difficult to tackle, as they pose both type of difficulties—those of combinatorial problems (e.g., the necessity to check essentially all the solutions to be certain that the optimal one has been found), and those of continuous problems (e.g., the fact that the search space is infinite and may be unbounded).

For this reason, there are not many dedicated algorithms to tackle mixed-variable problems directly. Most of the approaches that are in use relax some of the constraints of the problem and then try to solve the transformed problem with the hope that the solution obtained may be easily *repaired* to become a good solution to the original problem. The most popular approach is to relax the requirement for the discrete variables. They are replaced by continuous variables and the problem is solved as a continuous optimization problem, but taking into account during the objective function evaluation only the allowed values for the discrete variables. This type of approach is often called *continuous relaxation approach.*

The relaxation is particularly easy if the discrete variables are numerical, or at least if a certain order among them may be established. For instance, such would be the case for a variable representing the diameter of pipes used in water distribution systems. While the number of available diameters is limited, they may be easily ordered from the smallest to the largest. The continuous optimization algorithm may hence easy move between slightly smaller and slightly larger sizes during the optimization process.

The situation is different in the case of categorical discrete variables. An example of such problem is the choice of material to be used for manufacturing certain parts. Each material may have several different characteristics (e.g., weight, thermal conductivity, strength, etc.), all of which may be important for the optimization problem at hand. In such cases, there is no implicit ordering of such categorical variables and relaxation of this problem to the continuous case requires from the algorithm designer to impose an ordering. This choice may then influence the performance of the algorithm.

Certainly, relaxation of the mixed-variable problem to a continuous problem is not the only possible approach, and there exist also methods performing full mixed-variable optimization (see Section 2.3.2). Also, such a method is presented as part of this work—an ant colony optimization algorithm for tackling mixed-variable optimization problems.

2.3.1 Applications

Applications of mixed-variable optimization problems concern mostly industrial applications. Many industrial process and problems contain parameters of which some are discrete and some continuous. Popular examples include the truss design problem [Sellar et al., 1994; Turkkan, 2003; Pandia Raj and Kalyanaraman, 2005; Schmidt and Thierauf, 2005], the coil spring design problem [Deb and Goyal, 1998; Lampinen and Zelinka, 1999c; Guo et al., 2004], designing a pressure vessel [Deb and Goyal, 1998; Guo et al., 2004; Schmidt and Thierauf, 2005], welded beam design [Deb and Goyal, 1998], or design of thermal insulation systems [Audet and Dennis Jr., 2001; Kokkolaras et al., 2001].

All these problems require choosing values for discrete and continuous variables. Some of them may be easily transformed into continuous optimization problems (this is often the approach used to solve them). Some—the design of thermal insulation systems is an example—may not be easily transformed in such a way. This is due to the fact that there is no obvious ordering that may be imposed on the categorical variables.

2.3.2 Methods

The methods proposed in the literature to tackle mixed-variable optimization problems may be divided into two groups. The first group consists of methods that modify or divide the problem, so that it can be tackled with available continuous and combinatorial algorithms. The second group of methods represent the true mixed-variable optimization algorithms, which can natively handle mixed-variable optimization problems. We will shortly present both groups.

Hybrid and Relaxation-based Methods

As mentioned earlier, often mixed-variable optimization problems are relaxed to become continuous optimization problems. In such cases, most of the methods presented in Section 2.2 may be used to tackle them. In these cases, the methods are usually enriched with additional repair mechanism that converts the continuous values chosen by the original methods for the discrete variables to the actual values allowed. Popular methods used for such approach include differential evolution (DE) [Lampinen and Zelinka, 1999b,c], genetic algorithms (GA) [Turkkan, 2003], and particle swarm optimization (PSO) [Guo et al., 2004]. It should be noted that often also another approach is used with genetic algorithms—all continuous variables are discretized with a certain resolution and then incorporated into a binary encoding [Pandia Raj and Kalyanaraman, 2005].

Another possible approach to solving a mixed-variable optimization problem is to couple two methods specific to respectively combinatorial and continuous optimization. Then, one method may be used as a form of *local search* for the other. For instance, for a given problem, a continuous optimization method may be the main one, and for each evaluated combination of continuous variables, a full local search may be launched for the optimal values of the discrete variables. The situation may be also inverted—the combinatorial optimization method may play the role of the main algorithm, and the continuous one the role of local search. Depending on the problem at hand, one of these configurations may perform better than the other.

The inherent drawback of this approach comes from the different philosophy represented by continuous and combinatorial optimization methods. In continuous optimization (true also for most mixed-variable optimization problems), one of the crucial aspects is to have a small number of objective function evaluations. Usually the effort required to evaluate the objective function by far exceeds any other actions performed by the algorithm. At the same time, combinatorial optimization algorithms are usually optimized for absolute speed (usually measured in terms of CPU time). They often use *delta function evaluations*, that is, the objective function is only partially reevaluated if the values of only some discrete variables change. Such an approach is usually not possible in case of mixed-variable optimization, as it would usually lead to an excessive number of objective function evaluations.

Ways to overcome this problem have been proposed by some researchers. Parharaj and Azarm [Praharaj and Azarm, 1992] run in turn a combinatorial and a continuous optimization algorithms starting from the optimal solution found by the other one. Another approach is taken by Stelmack and Batill [Stelmack and Batill, 1997] who use a concurrent subspace optimization algorithm. They propose to discretize all the continuous variables, and solve such defined problem first. Then, the best solution found is used to run a continuous optimization algorithm on continuous variables.

Native Mixed-Variable Optimization Algorithms

Finally, there have been methods proposed in the literature that are able to natively handle mixed-variable optimization problems. Only few such methods have been proposed, and we now shortly present them here.

Genetic Adaptive Search: As could be seen from the discussion above, genetic algorithms allow for quite a flexible approach to representation of the variables. There are versions of genetic algorithms that are applied to mixed-variable optimization problems through relaxing all the variables to be continuous [Turkkan, 2003], but there exist also others, where in fact the continuous variables are treated as discrete ones [Pandia Raj and Kalyanaraman, 2005]. Clearly, this approach may be further extended to natively handling both continuous and discrete variables. Such an approach is presented by the genetic adaptive search [Deb and Goyal, 1998].

In order to enable simultaneous handling of both discrete and continuous variables, a particular crossover and mutation operators are used. The crossover is performed variable-by-variable. If a variable is discrete, a standard binary crossover is performed. If a variable is continuous, the children are created using a two-modal probability distribution, whose peaks are aligned with the values of the given continuous variable of the parents. The mutation operator is also applied per variable, and differs slightly depending on its type. For discrete variables, the mutation means a change of one of the bits with certain small probability. For the continuous ones, the new value may be replaced with one generated using a polynomial probability distribution.

Pattern Search Method: It is an iterative method that generates a sequence of feasible solutions whose objective function value is non-increasing. At any given iteration, the objective function f is evaluated at a finite number of points on a mesh in order to try to find one that yields a decrease in the objective function value. Initially, the method has been proposed for continuous optimization problems only [Torczon, 1997], but later it has been transformed to be applicable directly to mixed-variable optimization problems [Audet and Dennis Jr., 2001].

An iteration of a pattern search method is initiated with the *initial solution* s, as well as with an enumerable subset $\mathcal{M}$ of the domain $\Omega \subset \mathbf{S}$. The resolution of the mesh $\mathcal{M}$ is parametrized by a positive real number Δ. The goal of each iteration is to obtain a new initial solution on the current mesh whose objective function value is strictly less (by any amount) than the old one.

Exploration of the mesh is conducted in one or two phases. First, a finite search, free of any other rules imposed by the algorithm, is performed anywhere on the mesh. It is called the *search step*. Any strategy can be used as long as it searches finitely many points (including none). This part of the algorithm has the advantage that the user can put in place any *ad-hoc* search he/she might favor for improving the initial solution with the knowledge that if this fails, the next phase will provide a fail-safe.

If the search does not succeed in improving the initial solution, the second phase is called. A potentially exhaustive (but always finite) search in a local mesh neighborhood around s and around promising points in its set of neighbors is performed. The second phase (called the *poll step*) follows stricter rules than the first one, and it guarantees theoretical convergence to a local minimum of a quality specified by the user. The set of points visited in this phase is referred to as the *poll set*.

If a point with a better objective value than s is found in either phase, then the iteration is declared successful. The better point becomes the new initial solution, and the next iteration is initiated with a (possibly) coarser (and different) mesh around the new initial solution. Otherwise, the iteration is declared unsuccessful. In that case, the next iteration is initiated at the same (old) initial solution, but with a finer mesh on the continuous variables, and a set of neighbors *closer* (if possible) to the initial solution. A key property of the mesh exploration is that if an iteration is unsuccessful, then the current objective function value is less than or equal to the objective function values evaluated at all points in the trial set consisting of all points considered in the search and poll steps.

For a detailed description of the method of creating and evaluating the mesh $\mathcal{M}$ and the poll set, we refer the reader to [Audet and Dennis Jr., 2001].

Mixed Bayesian Optimization Algorithm: The bayesian optimization algorithm has been proposed initially by Pelikan *et al.* [Pelikan et al., 2000]. Recently, Očenášek and Schwarz [Očenášek and Schwarz, 2002] have proposed an extension of that algorithm to also handle mixed-variable optimization problems. In their approach, they use decision trees. They have adopted the CART model [Chipman et al., 1998], that is, the classification and regression tree, which may be used for both categorical and continuous splits.

For each target random variable x_i they build one decision tree. The split nodes of the i-th decision tree are used to cut the domain of parent variables $Pa(x_i)$ into parts, where the variable x_i is more linear or predictable. They start building the decision trees from empty trees and recursively add the splitting nodes until no splitting is favorable. The leaf nodes define the elementary models for obtaining the target variable x_i. For continuous variables, a one-dimensional normal probability density function is estimated and used as the leaf.

2.3.3 Choosing the Suitable Method

Again, each of the methods for mixed-variable optimization has its advantages and disadvantages. These must be taken into consideration when choosing a method to tackle a given problem. In this section, we shortly compare their main advantages and disadvantages.

Since mixed-variable optimization is a combination of combinatorial and continuous optimization, the methods used consist of essentially a certain combination of methods used in combinatorial and continuous optimization. The two types of methods identified in the previous section differ mostly in the way these combinatorial and continuous parts are used together.

The first type of methods divides or transforms the problem in such a way that regular combinatorial or continuous algorithms may be used to tackle it. The second type encompasses the algorithms that try to perform combinatorial and continuous optimization at the same time.

Advantages and Disadvantages

Choosing, which approach to use is often difficult. Often, researchers use one of the approaches and obtain reasonable results, but they do not investigate the other approach, nor they provide convincing arguments why the approach they used is the best one.

Certainly, the choice of the approach depends on the particular problem at hand. For instance, problems with combinatorial decision variables that are independent of the continuous ones may easily be divided into two separate problems, one being solved with a combinatorial optimization technique, and the other with a continuous one. However, often it is not obvious whether this is the case. If the variables are dependent, and the problem is nevertheless divided into the combinatorial and continuous parts and solved separately, the risk is that a sub-optimal solution will be found.

As we show in Chapter 6 of this work, another possible way of classifying mixed-variable optimization problems is considering whether the discrete variables may be easily ordered, or not. We show that, if they are ordered, it brings good results to use a continuous-relaxation approach, that is, to relax the discrete variables and solve the entire problem as a continuous optimization one. If the discrete variables can not be easily ordered, often the native mixed-variable approach is more efficient.

2.4 Discussion

In this chapter, we have presented how and why optimization problems may be divided into combinatorial, continuous, and mixed-variable ones. We have given the formal definition of these different problem classes. Further, we have discussed the area of their applications, and listed the most popular methods used to tackle them.

Although there exist exact methods for solving combinatorial and continuous optimization problems, they have their disadvantages. They may not be always used either due to the problem formulation, or to the fact that they would need excessive time to solve the problems. This is why metaheuristics and other approximate methods are enjoying increasing attention. Ant colony optimization, which is the subject of this work, is one of the metaheuristics. While ant colony optimization has been proposed as a method for combinatorial optimization, in this work we show that it may also be extended to the other types problems presented in this chapter, namely continuous and mixed-variable problems.

Clearly these two classes of problems are often encountered in real-world situations, especially in mechanical engineering. Efficient algorithms, especially for mixed-variable optimization are still not very numerous, and research is needed in order to make them available.

Chapter 3

Ant Colony Optimization

This chapter presents an overview of ant colony optimization (ACO) – a metaheuristic inspired by the behavior of real ants. ACO was proposed by Dorigo and colleagues [Dorigo et al., 1991; Dorigo, 1992; Dorigo et al., 1996] as a method for solving hard combinatorial optimization problems.

ACO algorithms may be considered to be part of *swarm intelligence*, that is, the research field that studies algorithms inspired by the observation of the behavior of *swarms*. Swarm intelligence algorithms are made up of simple individuals that cooperate through self-organization, that is, without any form of central control over the swarm members. A detailed overview of the self-organization principles exploited by these algorithms, as well as examples from biology, can be found in [Camazine et al., 2003]. Many swarm intelligence algorithms have been proposed in the literature. For an overview of the field of swarm intelligence, we refer the interested reader to [Bonabeau et al., 1999].

This chapter, which is dedicated to present a concise overview of ACO, is organized as follows. Section 3.1 presents the biological phenomenon that provided the original inspiration. Section 3.2 presents a formal description of the ACO metaheuristic. Section 3.3 overviews the most popular variants of ACO and gives examples of their application. Section 3.4 shows current research directions, and Section 3.5 summarizes and concludes the chapter.

3.1 From Biology to Algorithms

Ant colony optimization was inspired by the observation of the behavior of real ants. In this section, we present a number of observations made in experiments with real ants, and then we show how these observations inspired the design of the ACO metaheuristic.

3.1.1 Ants

One of the first researchers to investigate the social behavior of insects was the French entomologist Pierre-Paul Grassé. In the forties and fifties of the 20-th century, he was observing the behavior of termites – in particular, the *Bellicositermes natalensis* and *Cubitermes* species. He discovered [Grassé, 1946] that these insects are capable to react to what he called “significant stimuli”, signals that activate a genetically encoded reaction. He observed [Grassé, 1959] that the effects of these reactions can act as new significant stimuli for both the insect that produced them and for the other insects in the colony. Grassé used the term *stigmergy* [Grassé, 1959] to describe this particular type of indirect communication in which the “workers are stimulated by the performance they have achieved”.

The two main characteristics of stigmergy that differentiate it from other means of communication are:

- the physical, non-symbolic nature of the information released by the communicating insects, which corresponds to a modification of physical environmental states visited by the insects; and
- the local nature of the released information, which can only be accessed by those insects that visit the place where it was released (or its immediate neighborhood).

Examples of stigmergy can be observed in colonies of ants. In many ant species, ants walking to, and from, a food source deposit on the ground a substance called *pheromone*. Other ants are able to smell this pheromone, and its presence influences the choice of their path—i.e., they tend to follow strong pheromone concentrations. The pheromone deposited on the ground forms a *pheromone trail*, which allows the ants to find good sources of food that have been previously identified by other ants.

Some researchers investigated experimentally this pheromone laying and following behavior in order to better understand it and to be able to quantify it. Deneubourg *et al.* [Deneubourg et al., 1990] set up an experiment called a “binary bridge experiment”.

They used *Linepithema humile* ants (also known as Argentine ants). The ants' nest was connected to a food source by two bridges of equal length. The ants could freely choose which bridge to use when searching for food and bringing it back to the nest. Their behavior was then observed over a period of time.

In this experiment, initially there is no pheromone on the two bridges. The ants start exploring the surroundings of the nest and eventually cross one of the bridges and reach the food source. When walking to the food source and back, the ants deposit pheromone on the bridge they use. Initially, each ant randomly chooses one of the bridges. However, due to random fluctuations, after some time there will be more pheromone deposited on one of the bridges than on the other. Because ants tend to prefer in probability to follow a stronger pheromone trail, the bridge that has more pheromone will attract more ants. This in turn makes the pheromone trail grow stronger, until the colony of ants converges towards the use of a same bridge.[1]

This colony level behavior, based on autocatalysis, that is, on the exploitation of positive feedback, can be exploited by ants to find the shortest path between a food source and their nest. This was demonstrated in another experiment conducted by Goss *et al.* [Goss et al., 1989], in which the two bridges were not of the same length: one was significantly longer than the other. In this case, the stochastic fluctuations in the initial choice of a bridge were much reduced as a second mechanism played an important role: those ants choosing by chance the shorter bridge were also the first to reach the nest and when returning to the nest they chose the shorter bridge with higher probability as it had a stronger pheromone trail. Therefore, the ants—thanks to the pheromone following and depositing mechanism—quickly converged to the use of the shorter bridge.

In the next section we explain how these experiments and findings were used to develop optimization algorithms.

3.1.2 Algorithms

Stimulated by the interesting results of the experiments described in the previous section, Goss *et al.* [Goss et al., 1989] developed a model to explain the behavior observed in the binary bridge experiment. Assuming that after t time units since the start of the experiment, m_1 ants had used the first bridge and m_2 the second one, the probability p_1 for the $(m+1)$-th ant to choose the first bridge can be given by:

[1]Deneubourg *et al.* conducted several experiments, and results show that each of the two bridges was used in about 50% of the cases.

$$p_{1(m+1)} = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} \ , \tag{3.1}$$

where parameters k and h are needed to fit the model to the experimental data. The probability that the same $(m+1)$-th ant chooses the second bridge is $p_{2(m+1)} = 1 - p_{1(m+1)}$. Monte Carlo simulations, run to test how the model corresponds to the real data [Pasteels et al., 1987], showed very good fit for $k \approx 20$ and $h \approx 2$.

This basic model, which explains the behavior of real ants, may be used as an inspiration to design artificial ants that solve optimization problems defined in a similar way. In the above described *ant foraging behavior* example, stigmergic communication happens via the pheromone that ants deposit on the ground. Analogously, artificial ants may simulate pheromone laying by modifying appropriate pheromone variables associated with problem states they visit while building solutions to the optimization problem. Also, according to the stigmergic communication model, the artificial ants would have only local access to these pheromone variables.

Therefore, the main characteristics of stigmergy mentioned in the previous section can be extended to artificial agents by:

- associating state variables with different problem states; and
- giving the agents only local access to these variables.

Another important aspect of real ants' foraging behavior that may be exploited by artificial ants is the coupling between the autocatalytic mechanism and the *implicit evaluation* of solutions. By implicit solution evaluation, we mean the fact that shorter paths (which correspond to lower cost solutions in the case of artificial ants) are completed earlier than longer ones, and therefore they receive pheromone reinforcement quicker. Implicit solution evaluation coupled with autocatalysis can be very effective: the shorter the path, the sooner the pheromone is deposited, and the more ants use the shorter path. If appropriately used, it can be a powerful mechanism in population-based optimization algorithms (e.g., in evolutionary algorithms [Holland, 1975; Fogel, 1995] autocatalysis is implemented by the selection/reproduction mechanism).

Stigmergy, together with implicit solution evaluation and autocatalytic behavior, gave rise to ACO. The basic idea of ACO follows very closely the biological inspiration. Therefore, there are many similarities between real and artificial ants. Both real and artificial ant colonies are composed of a population of individuals that work together to achieve a certain goal. A colony is a population of simple, independent, asynchronous agents that cooperate to find a good *solution* to the problem at hand. In the case of real ants, the problem is to find the food, while in the case of artificial ants, it is to find a good solution to a given optimization problem. A single ant (either a real or an artificial one) is able to find a solution to its problem, but only cooperation among many individuals through stigmergy enables them to find *good* solutions.

In the case of real ants, they deposit and react to a chemical substance called *pheromone.* Real ants simply deposit it on the ground while walking. Artificial ants live in a *virtual* world, hence they only modify numeric values (called for analogy *artificial pheromones*) associated with different problem states. A sequence of pheromone values associated with problem states is called *artificial pheromone trail.* In ACO, the artificial pheromone trails are the sole means of communication among the ants. A mechanism analogous to the evaporation of the physical pheromone in real ant colonies allows the artificial ants to *forget* the past history and focus on new promising search directions.

Just like real ants, artificial ants create their solutions sequentially by moving from one problem state to another. Real ants simply walk, choosing a direction based on local pheromone concentrations and a stochastic decision policy. Artificial ants also create solutions step-by-step, moving through available problem states and making stochastic decisions at each step.

There are however some important differences between real and artificial ants:

- Artificial ants live in a discrete world – they move sequentially through a finite set of problem states.

- The pheromone update (i.e., pheromone depositing and evaporation) is not accomplished in exactly the same way by artificial ants as by real ones. Sometimes the pheromone update is done only by some of the artificial ants, and often *only after* a solution has been constructed.

- Some implementations of artificial ants use additional mechanisms that do not exist in the case of real ants. Examples include look-ahead, local search, backtracking, etc.

3.2 The Ant Colony Optimization Metaheuristic

Ant colony optimization (ACO) has been formalized into a combinatorial optimization metaheuristic by Dorigo *et al.* [Dorigo and Di Caro, 1999; Dorigo et al., 1999; Dorigo and Stützle, 2004] and has since been used to tackle many combinatorial optimization problems.

Given a combinatorial optimization problem (COP), which has been already presented in Section 2.1, the first step for the application of ACO to its solution consists in defining an adequate model. This is then used to define the central component of ACO: the pheromone model.

First, an instantiated decision variable $X_i = v_i^j$ (i.e., a variable X_i with a value v_i^j assigned from its domain $\mathbf{D}_i$), is called a *solution component* and denoted by c_{ij}. The set of all possible solution components is denoted by $\mathbf{C}$. A pheromone trail parameter T_{ij} is then associated with each component c_{ij}. The set of all pheromone trail parameters is denoted by $\mathbf{T}$. The value of a pheromone trail parameter T_{ij} is denoted by τ_{ij} (and called pheromone value).[2] This pheromone value is then used and updated by the ACO algorithm during the search. It allows modeling the probability distribution of different components of the solution.

In ACO, artificial ants build a solution to a combinatorial optimization problem by traversing the so-called *construction graph*, $G_C(\mathbf{V}, \mathbf{E})$. The fully connected construction graph consists of a set of vertexes $\mathbf{V}$ and a set of edges $\mathbf{E}$. The set of components $\mathbf{C}$ may be associated either with the set of vertexes $\mathbf{V}$ of the graph G_C, or with the set of its edges $\mathbf{E}$. The ants move from vertex to vertex along the edges of the graph, incrementally building a *partial solution.* Additionally, the ants deposit a certain amount of pheromone on the components, that is, either on the vertexes or on the edges that they traverse. The amount $\Delta\tau$ of pheromone deposited may depend on the quality of the solution found. Subsequent ants utilize the pheromone information as a guide towards more promising regions of the search space.

The ACO metaheuristic is shown in Algorithm 1. It consists of an initialization step and a loop over three algorithmic components. A single iteration of the loop consists of constructing solutions by all ants, their (optional) improvement with the use of a local search algorithm, and an update of the pheromones. In the following, we explain these three algorithmic components in more detail.

[2]Note that pheromone values are in general a function of the algorithm's iteration $t : \tau_{ij} = \tau_{ij}(t)$.

Algorithm 1 Ant colony optimization metaheuristic

```
Set parameters, initialize pheromone trails
while termination conditions not met do
    ConstructAntSolutions
    ApplyLocalSearch        {optional}
    UpdatePheromones
end while
```

ConstructAntSolutions: A set of m artificial ants construct solutions from elements of a finite set of available solution components $\mathbf{C} = \{c_{ij}\},\ i = 1, ..., n,\ j = 1, ..., |\mathbf{D}_i|$. A solution construction starts with an empty partial solution $s^p = \emptyset$. Then, at each construction step, the current partial solution s^p is extended by adding a feasible solution component from the set of feasible neighbors $\mathbf{N}(s^p) \subseteq \mathbf{C}$. The process of constructing solutions can be regarded as a path on the construction graph $G_C = (\mathbf{V}, \mathbf{E})$. The allowed paths in G_C are hereby implicitly defined by the solution construction mechanism that defines the set $\mathbf{N}(s^p)$ with respect to a partial solution s^p.

The choice of a solution component from $\mathbf{N}(s^p)$ is done probabilistically at each construction step. The exact rules for the probabilistic choice of solution components vary across different ACO variants. The best known rule is the one of Ant System (AS) [Dorigo et al., 1996]:

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \cdot \eta(c_{ij})^{\beta}}{\sum_{c_{il} \in \mathbf{N}(s^p)} \tau_{il}^{\alpha} \cdot \eta(c_{il})^{\beta}}, \quad \forall c_{ij} \in \mathbf{N}(s^p), \tag{3.2}$$

where τ_{ij} is the pheromone value associated with the component c_{ij}, and $\eta(\cdot)$ is a function that assigns at each construction step a heuristic value to each feasible solution component $c_{ij} \in \mathbf{N}(s^p)$. The values that are given by this function are commonly called *heuristic information.* Furthermore, α and β are positive parameters, whose values determine the relative importance of pheromone versus heuristic information. Eq. 3.2 is a generalization of Eq. 3.1 presented in Sec. 3.1: ACO formalization follows closely the biological inspiration.

ApplyLocalSearch: Once solutions have been constructed, and before updating pheromones, often some optional actions may be required. These are often called *daemon actions*, and can be used to implement problem specific and/or centralized actions, which cannot be performed by single ants. The most used daemon action consists in the application of local search to the constructed solutions: the locally optimized solutions are then used to decide which pheromones to update.

UpdatePheromones: The aim of the pheromone update is to increase the pheromone values associated with good or promising solutions, and to decrease those that are associated with bad ones. Usually, this is achieved (i) by decreasing all the pheromone values through *pheromone evaporation*, and (ii) by increasing the pheromone levels associated with a chosen set of good solutions $\mathbf{S}_{upd}$:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \rho\cdot\sum_{s\in\mathbf{S}_{upd}|c_{ij}\in s} F(s)\ , \tag{3.3}$$

where $\mathbf{S}_{upd}$ is the set of solutions that are used for the update, $\rho \in (0,1]$ is a parameter called evaporation rate, and $F : \mathbf{S} \to \mathbb{R}_0^+$ is a function such that $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in \mathbf{S}$. $F(\cdot)$ is commonly called the *fitness function*.

Pheromone evaporation is needed to avoid a too rapid convergence of the algorithm. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Different ACO algorithms, such as for example Ant Colony System (ACS) [Dorigo and Gambardella, 1997] or $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [Stützle and Hoos, 2000] differ in the way they update the pheromone.

Instantiations of the update rule presented in Eq. 3.3 are obtained by different specifications of $\mathbf{S}_{upd}$, which in many cases is a subset of $\mathbf{S}_{iter} \cup \{s_{bs}\}$, where $\mathbf{S}_{iter}$ is the set of solutions that were constructed in the current iteration, and s_{bs} is the *best-so-far* solution, that is, the best solution found since the first algorithm iteration. A well-known example is the AS-update rule, that is, the update rule of Ant System [Dorigo et al., 1996], where:

$$\mathbf{S}_{upd} \leftarrow \mathbf{S}_{iter}\ . \tag{3.4}$$

An example of a pheromone update rule that is more often used in practice is the IB-update rule (where IB stands for *iteration-best*):

$$\mathbf{S}_{upd} \leftarrow \arg \max_{s \in \mathbf{S}_{iter}} F(s) \ . \tag{3.5}$$

The IB-update rule introduces a much stronger bias towards the good solutions found than the AS-update rule. Although this increases the speed with which good solutions are found, it also increases the probability of premature convergence. An even stronger bias is introduced by the BS-update rule, where BS refers to the use of the best-so-far solution s_{bs}. In this case, $\mathbf{S}_{upd}$ is set to $\{s_{sb}\}$. In practice, ACO algorithms that use variations of the IB-update or the BS-update rules and that additionally include mechanisms to avoid premature convergence, achieve better results than those that use the AS-update rule.

3.2.1 Example: The Traveling Salesman Problem

One of the most popular ways to illustrate how the ACO metaheuristic works, is via its application to the traveling salesman problem (TSP). The TSP consists of a set of locations (cities) and a travelling salesman that has to visit all the locations once and only once. The distances between the locations are given and the task is to find a Hamiltonian tour of minimal length. The problem has been proven to be NP-hard [Lawler et al., 1985].

The application of ACO to the TSP is straightforward. The moves between the locations become the solution components—i.e, the move from city i to city j becomes a solution component $c_{ij} \equiv c_{ji}$. The construction graph $G_C = (\mathbf{V}, \mathbf{E})$ is defined by associating the set of locations with the set $\mathbf{V}$ of vertices of the graph. Since, in principle, it is possible to move from any city to any other one, the construction graph is fully connected and the number of vertices is equal to the number of locations defined by the problem instance. Furthermore, the lengths of the edges between the vertices are proportional to the distances between the locations represented by these vertices. The pheromone is associated with the set $\mathbf{E}$ of edges of the graph. An example of the resulting construction graph G_C is presented in Fig. 3.1a.

The ants construct the solutions as follows. Each ant starts from a randomly selected location (vertex of the graph G_C). Then, at each construction step it moves along the edges of the graph. Each ant keeps a memory of its path through the graph, and in subsequent steps it chooses among the edges that do not lead to vertexes that it has already visited. An ant has constructed a solution once it has visited all the vertexes

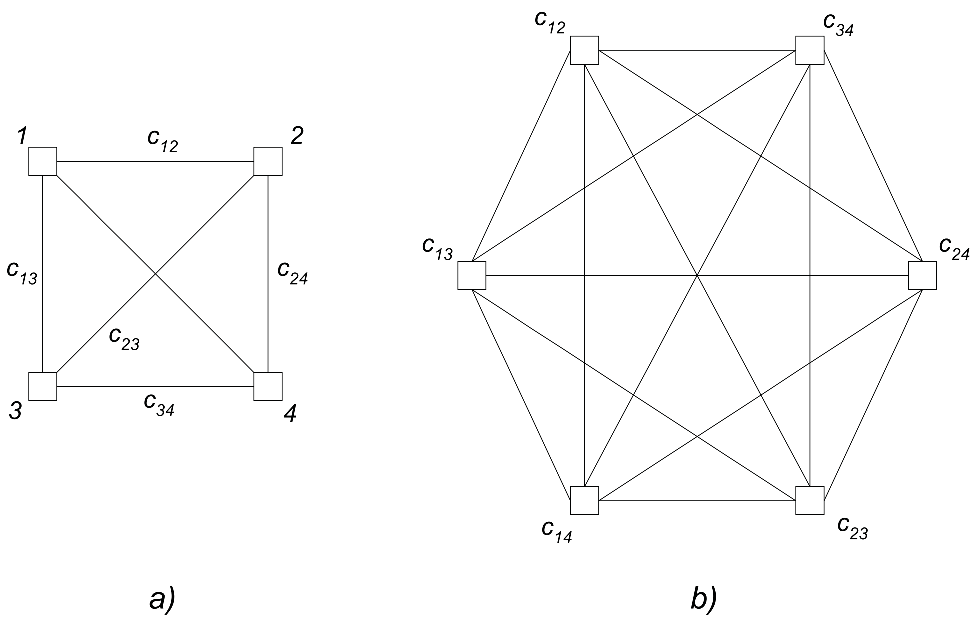


Figure 3.1: Example construction graphs for a 4-city TSP. a) - when components are associated with the edges of the graph, b) - when components are associated with the vertexes of the graph. Note that $c_{ij} \equiv c_{ji}$.

of the graph. At each construction step an ant chooses probabilistically the edge to follow among the available ones (those that lead to yet unvisited vertices). The exact rule depends on the implementation, an example being Eq. 3.2. Once all the ants have finished their tour, the pheromone on the edges is updated according to one of the possible implementations of Eq. 3.3. ACO has been shown to perform quite well on the TSP [Stützle and Dorigo, 1999].

It is worth noticing that it is also possible to associate the set of solution components of the TSP (or any other combinatorial optimization problem) with the set of vertices $\mathbf{V}$ rather than the set of edges $\mathbf{E}$ of the construction graph G_C. For the TSP, this would mean associating the moves between locations with the set $\mathbf{V}$ of vertices of the construction graph, and the locations with the set $\mathbf{E}$ of its edges. The corresponding example construction graph for a 4-city TSP is presented in Fig. 3.1b. When using this approach, the ants' solution construction process has to be also properly modified: the ants would have to move from vertex to vertex of the construction graph choosing thereby the *connections between the cities.*

It is important to note that both ways of defining the construction graph are correct and both may be used in practice. Depending on the problem at hand, one may be more intuitive than the other. For instance, for the University Course Timetabling Problem (UCTP) the second one seems better suited [Socha et al., 2002].

3.3 Main Variants of ACO

Several variants of ACO have been proposed in the literature. We present the main characteristics of the most successful ones together with a short list of their applications. We attempt to present them in chronological order as new variants are often based on ideas introduced earlier.

In the following sections we present Ant System—the first implementation of an ACO algorithm—followed by $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and Ant Colony System. We mention also some others that are less popular but still quite interesting, such as hyper-cube ACO or population-based ACO. In order to illustrate the differences between them clearly, we use the example of the traveling salesman problem, as described in Sec. 3.2.1.

3.3.1 Ant System

Ant System (AS) was the first ACO algorithm to be proposed in the literature [Dorigo et al., 1991; Dorigo, 1992; Dorigo et al., 1996]. Its main characteristic is that the pheromone values are updated by *all* the ants that have completed the tour. The pheromone update for τ_{ij}, that is, for edge joining cities i and j, is performed as follows:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \ , \tag{3.6}$$

where ρ is the evaporation rate, m is the number of ants, and $\Delta\tau_{ij}^{k}$ is the quantity of pheromone per unit length laid on edge (i,j) by the k-th ant:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if ant } k \text{ used edge } (i,j) \text{ in its tour,} \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

where Q is a constant, and L_k is the tour length of the k-th ant.

When constructing the solutions, the ants in AS traverse a construction graph and make probabilistic decision at each vertex. The transitional probability p_{ij}^k of the k-th ant moving from city i to city j is given by:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in \text{allowed}_k} \tau_{il}^\alpha \cdot \eta_{il}^\beta} & \text{if } j \in \text{allowed}_k, \\ 0 & \text{otherwise}, \end{cases} \tag{3.8}$$

where allowed_k is the list of cities not yet visited by the k-th ant, and α and β are parameters that control the relative importance of the pheromone versus the heuristic information η_{ij} given by:

$$\eta_{ij} = \frac{1}{d_{ij}} \,, \tag{3.9}$$

where d_{ij} is the length of edge (i, j).

Several implementations of the AS algorithm have been applied to different combinatorial optimization problems. The first and best known is the application to the TSP [Dorigo et al., 1991; Dorigo, 1992; Dorigo et al., 1996]. However, AS was also used successfully to tackle other combinatorial problems. The AS-QAP [Maniezzo et al., 1994; Maniezzo and Colorni, 1999] algorithm was used to tackle quadratic assignment problem (QAP), AS-JSP [Colorni et al., 1994] for the job-shop scheduling problem (JSP), AS-VRP [Bullnheimer et al., 1998, 1999] for the vehicle routing problem (VRP), and AS-SCS [Michel and Middendorf, 1998, 1999] for the shortest common supersequence (SCS) problem.

3.3.2 $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System

$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) is an improvement over the original Ant System idea. $\mathcal{MM}$AS was proposed by Stützle and Hoos [Stützle and Hoos, 2000] and introduces the following two changes:

- only the best ant can update the pheromone trails, and
- the minimum and maximum values of the pheromone are limited.

Equation 3.6 takes hence the following new form:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{\text{best}} \ , \tag{3.10}$$

where $\Delta\tau_{ij}^{\text{best}}$ is the pheromone update value defined by:

$$\Delta\tau_{ij}^{\text{best}} = \begin{cases} \frac{Q}{L_{\text{best}}} & \text{if the best ant used edge } (i,j) \text{ in its tour,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

L_{best} is the length of the tour of the best ant. This may be (subject to the algorithm designer decision) either the best tour found in the current iteration—*iteration-best*, L_{ib}—or the best solution found since the start of the algorithm—*best-so-far*, L_{bs}—or a combination of both.

Concerning the limits on the minimal and maximal pheromone values allowed, respectively τ_{min} and τ_{max}, Stützle and Hoos suggest that they should be chosen experimentally based on the problem at hand. The maximum value τ_{max} may be calculated analytically provided that the optimum ant tour length is known. In the case of the TSP, τ_{max} is given by:

$$\tau_{max} = \frac{1}{\rho}\cdot\frac{1}{L^*} \ , \tag{3.12}$$

where L^* is the length of the optimal tour. The minimum pheromone value τ_{min} should be chosen with caution as it has a rather strong influence on the algorithm performance. They present an analytical approach to finding this value based on the probability p_{best} that an ant constructs the best tour found so far. This is done as follows. First, it is assumed that at each construction step an ant has a constant number k of options available. Therefore, the probability that an ant makes the *right* decision (i.e., the decision that belongs to the sequence of decisions leading to the construction of the best tour found so far) at each of n steps is given by $p_{\text{dec}} = \sqrt[n-1]{p_{\text{best}}}$. The analytical formula they suggest for finding τ_{min} is:

$$\tau_{min} = \frac{\tau_{max} \cdot (1 - p_{dec})}{k \cdot p_{dec}} \ . \tag{3.13}$$

For more details on how to choose τ_{max} and τ_{min}, we refer to [Stützle and Hoos, 2000]. It is important to mention here that it has been also shown [Socha et al., 2002] that for some problems the choice of an appropriate τ_{min} value is more easily done experimentally than analytically.

The process of pheromone update in $\mathcal{MM}$AS is concluded by verifying that all pheromone values are within the imposed limits:

$$\tau_{ij} = \begin{cases} \tau_{max} & \text{if } \tau_{ij} > \tau_{max}, \\ \tau_{min} & \text{if } \tau_{ij} < \tau_{min}. \end{cases} \tag{3.14}$$

$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System provided a significant improvement over the basic Ant System performance. While the first implementations focused on the TSP [Stützle and Hoos, 2000], it has been later applied to many other combinatorial optimization problems such as the QAP [Stützle and Hoos, 1998] or the university course timetabling problem (UCTP) [Socha et al., 2002], the generalized assignment problem (GAP) [Ramalhinho-Lourenço and Serra, 1998], and the set covering problem (SCP) [Lessing et al., 2004].

3.3.3 Ant Colony System

Another improvement over the original Ant System was Ant Colony System (ACS) introduced by Gambardella and Dorigo [Gambardella and Dorigo, 1996; Dorigo and Gambardella, 1997]. The most interesting contribution of ACS is the introduction of a *local pheromone update* in addition to the pheromone update performed at the end of the construction process (called here *offline* pheromone update).

The local pheromone update is performed by all the ants after each construction step. Each ant applies it only to the last edge traversed:

$$\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0 \ , \tag{3.15}$$

where $\varphi \in (0, 1]$ is the pheromone decay coefficient, and τ_0 is the initial value of the pheromone.

The main goal of the local update is to diversify the search performed by subsequent ants during one iteration. In fact, decreasing the pheromone concentration on the edges as they are traversed during one iteration encourages subsequent ants to choose other edges and hence to produce different solutions. This makes less likely that several ants produce identical solutions during one iteration.

The offline pheromone update, similarly to $\mathcal{MM}$AS, is applied at the end of each iteration by only one ant (the one that found the best solution in the iteration). However, the update formula is slightly different:

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij} & \text{if edge } (i,j) \text{ belongs to } T_{\text{best}}, \\ \tau_{ij} & \text{otherwise,} \end{cases} \tag{3.16}$$

and in case of TSP, $\Delta\tau_{ij} = \frac{1}{L_{\text{best}}}$.

Another important difference between AS and ACS is in the decision rule used by the ants during the construction process. Ants in ACS use the so-called *pseudorandom proportional* rule: the probability for an ant to move from city i to city j depends on a random variable q uniformly distributed over $[0, 1]$, and a parameter q_0; if $q \leq q_0$, then $j = \text{argmax}_{l \in N(s^p)}\{\tau_{il}\eta_{il}^{\beta}\}$, otherwise Eq. 3.8 is used.

ACS has been initially developed for the travelling salesman problem [Gambardella and Dorigo, 1996; Dorigo and Gambardella, 1997], but it was later used to tackle various combinatorial optimization problems, including vehicle routing [Bianchi et al., 2004] and timetabling [Socha et al., 2003].

3.3.4 Others

In addition to the main variants of ACO just described, it is worth mentioning the hypercube ACO (HC-ACO) proposed by Blum [Blum and Dorigo, 2004], and population-based ACO (PB-ACO) proposed by Guntsch and Middendorf [Guntsch and Middendorf, 2002b].

The main idea introduced by HC-ACO is the normalization of pheromone values used in the pheromone table. According to HC-ACO, the pheromone values should always be normalized in the interval $[0,1]$. It has been shown [Blum and Dorigo, 2004] that this makes the HC-ACO algorithm behavior independent of the scaling of the objective function, an issue for previous ACO algorithms.

Population-based ACO introduces a novel mechanism for pheromone updates. As in regular ACO, some of the good solutions found are used to increase the pheromone values. However, pheromone evaporation is implemented differently. PB-ACO memorizes the solutions used to increase the pheromone values (the set of memorized solutions is called a "population", hence its name). Once the population has reached its maximum dimension (a parameter of the algorithm), the worst solutions in the population are removed to make room for the new ones. When a solution of the population is removed, the pheromone associated with it is also removed: this is obtained by applying a negative pheromone update.

3.4 Future Directions

Research in ant colony optimization is very active. It includes the application of ACO algorithms to new real-world optimization problems or new types of problems, such as dynamic optimization [Guntsch and Middendorf, 2002a], multiobjective optimization [Iredi et al., 2001], stochastic problems [Gutjahr, 2004], or continuous and mixed-variable optimization [Socha, 2004]. Also, with an increasing popularity of parallel hardware architectures (multi-core processors and the grid technology), a lot of research is being done on creating parallel implementations of ACO that will be able to take advantage of the available hardware. In this section we shortly present current research in these new areas.

3.4.1 Parallel ACO Implementations

Parallelization of algorithms becomes more and more an interesting and practical option for algorithm designers. ACO is particularly well suited for parallel implementations thanks to ants operating in an independent and asynchronous way. There have already been many attempts to propose parallel ACO algorithms. They are usually classified by their *parallel grain*, that is, the relationship between computation and communication. We can then distinguish between *coarse-grained* and *fine-grained* models. While the former are characterized by many ants using the same CPU and rare communication between the CPUs, in the latter only few ants use each CPU and there is a lot of communication going on. A review of the trends and strategies in designing parallel algorithms may be found in [Cung et al., 2001].

Randall and Lewis proposed a first reasonably complete classification of parallel ACO implementations [Randall and Lewis, 2002]. Although many parallel ACO implementations have been proposed in the literature [Merkle and Middendorf, 2002; Talbi et al., 1999; Gambardella et al., 1999; Rahoual et al., 2002; Bullnheimer et al., 1997; Stützle, 1998], the results are fragmented and difficult to compare. Experiments are usually of limited scale and concern different optimization problems. Also, not all parallel implementations proposed are compared with their sequential counterparts, which is an essential measure of their usefulness [Stützle, 1998]. All this implies that more research is necessary in the area of parallelization of the ACO metaheuristic.

3.4.2 Other Types of Problems

Dynamic Optimization

One of the new areas of application of ACO is dynamic optimization. This type of problems are characterized by the fact that the search space dynamically changes. While an algorithm searches for good solutions, the conditions of the search as well as the quality of the solutions already found may change. This poses a whole new set of issues for designing successful algorithms that can deal with such situations. It becomes crucial for an algorithm to be able to adjust the search direction, following the changes of the problem being solved. Initial attempts to apply ACO to dynamic optimization problems have been quite successful [Di Caro and Dorigo, 1998; Guntsch et al., 2001; Guntsch and Middendorf, 2002a] .

Multiobjective Optimization

Multiobjective optimization is another area of application for metaheuristics that has received increasing attention over the past years. A multiobjective optimization problem involves solving simultaneously several optimization problems with potentially conflicting objectives. For each of the objectives, a different objective function is used to assess the quality of the solutions found. Algorithms usually aim at finding the so called *Pareto* *set*—i.e., a set of non-dominated solutions—based on the defined objective functions. In the Pareto set, no solution is worse than any other in the set, when evaluated over all the objective functions. Some ACO algorithms designed to tackle multiobjective problems have been proposed in the literature [Iredi et al., 2001; Guntsch and Middendorf, 2003; Doerner et al., 2004].

3.4.3 Continuous and Mixed-Variable Optimization

Finally, another new direction of research related to ACO is its extension from the combinatorial domain, so that it may be applied also to continuous and mixed-variable optimization problems. As discussed already in Chapter 2, tackling these optimization problems presents different requirements to an algorithm. There have been already some attempts to apply ACO-inspired algorithms to continuous optimization problems. The best known include Continuous ACO (CACO) [Bilchev and Parmee, 1995], API algorithm [Monmarché et al., 2000], Continuous Interacting Ant Colony (CIAC) [Dréo and Siarry, 2002]. These approaches however do not follow the ACO metaheuristic very closely, and hence may not be really considered true extensions of ACO to continuos domain. Also, none of the algorithms proposed so far can tackle both continuous and mixed-variable optimization problems. Until now.

Indeed, this thesis focuses on an extension of the ACO metaheuristic to continuous and mixed-variable domains. The algorithms presented in the remaining chapters of this work not only perform well when tested on various continuous and mixed-variable benchmark problems, but also follow closely the ACO philosophy. Hence, they may be considered an extension of ACO to both continuous and mixed-variable domains.

3.5 Discussion

We have presented an introduction to ant colony optimization—a metaheuristic inspired by the foraging behavior of real ants. The central component of ACO is the pheromone model based on the underlying model of the problem being solved. The basic idea of ACO, which has been formalized into a metaheuristic framework, leaves many options and choices to the algorithm designer. Several variants of ACO have been already proposed, the most successful being $\mathcal{MM}$AS Ant System and Ant Colony System.

ACO is a relatively young metaheuristic, when compared to others such as evolutionary computation, tabu search, or simulated annealing. Yet, it has proven to be quite efficient and flexible. ACO algorithms are currently state-of-the-art for solving many combinatorial optimization problems including the sequential ordering problem (SOP) [Gambardella and Dorigo, 2000], the resource constraint project scheduling (RCPS) problem [Merkle et al., 2002], and the open shop scheduling (OSS) problem [Blum, 2005]. For an in-depth overview of ACO, including applications, the interested reader should refer to [Dorigo and Stützle, 2004].

Chapter 4

ACO for Continuous Domains

Combinatorial optimization—as the name suggests—deals with finding optimal *combinations* or *permutations* of available problem components. Hence, it is required that the problem is partitioned into a finite set of components, and the combinatorial optimization algorithm attempts to find their optimal combination or permutation. Many real-world optimization problems may be represented as COPs in a straightforward way. There exists however an important class of problems for which this is not the case: the class of optimization problems that require choosing values for continuous variables. Such problems may be tackled with a combinatorial optimization algorithm only once the continuous *ranges* of allowed values are converted into finite sets. This is not always convenient, especially if the initial possible range is wide, and the resolution required is very high. In these cases, algorithms that can natively handle continuous variables usually perform better.

Chapter 3 presented already in considerable detail the basics of ant colony optimization. This chapter presents a way to effectively apply ACO—an algorithm originally developed to tackle COPs—to continuous optimization problems.

From the early days of ACO as a combinatorial optimization tool, reasearchers have tried to use it for tackling also continuous problems. However, applying the ACO metaheuristic to continuous domains was not obvious. The different methods proposed often deviated from the original ACO formulation.

This work presents a way to extend ACO to continuous domain without the need to make any major conceptual change to its structure. To improve the clarity of this thesis, we denote this ACO extended to continuous domains by $\text{ACO}_{\mathbb{R}}$. We aim at presenting the core idea of applying $\text{ACO}_{\mathbb{R}}$ to continuous domains as well as an implementation that performs well on standard benchmark test problems. In order to have a proper perspective on the performance of $\text{ACO}_{\mathbb{R}}$, we compare it not only to other ant-related methods, but also to other metaheuristics used for continuous optimization.

The reminder of the chapter is organized as follows. Section 4.1 presents $\text{ACO}_{\mathbb{R}}$, our extension of ACO to tackle continuous optimization problems. Section 4.2 provides a discussion of the proposed approach with regard to other methods for continuous optimization. Section 4.3 presents the experimental setup and results obtained, and compares them to the results found in the literature. Finally, Section 4.4 presents the conclusions and future work plans.

Additionally, the material presented in this chapter is futher extended in several appnedixes at the end of the thesis. Appendix A discusses an important issue concerning variable correlation handling in $\text{ACO}_{\mathbb{R}}$. Appendix B presents an alternative way of describing the proposed $\text{ACO}_{\mathbb{R}}$ algorithm, one that was used in some initial publications [Socha and Dorigo, 2008; Socha and Blum, 2006, 2007]. Finally, Appendix C provides the source code of the algorithm presented in this chapter.

4.1 The Algorithm

The idea that is central to the way ACO works is the incremental construction of solutions based on the biased probabilistic choice of solution components. In ACO applied to combinatorial optimization problems, the set of available solution components is defined by the problem formulation. At each construction step, ants make a probabilistic choice of the solution component c_i from the set $N(s^p)$ of available components according to Equation 3.2. The probabilities associated with the elements of the set $N(s^p)$ make up a *discrete probability distribution* (Figure 4.1a) that an ant samples in order to choose a component to be added to the current partial solution s^p.

The fundamental idea underlying $\text{ACO}_{\mathbb{R}}$ is the shift from using a *discrete* probability distribution to using a *continuous* one, that is, a *probability density function* (PDF) (Figure 4.1b). In $\text{ACO}_{\mathbb{R}}$, instead of choosing a component $c_{ij} \in N(s^p)$ according to Equation 3.2, an ant samples a PDF.

The ACO metaheuristic finds approximate solutions to an optimization problem by iterating the following two steps:

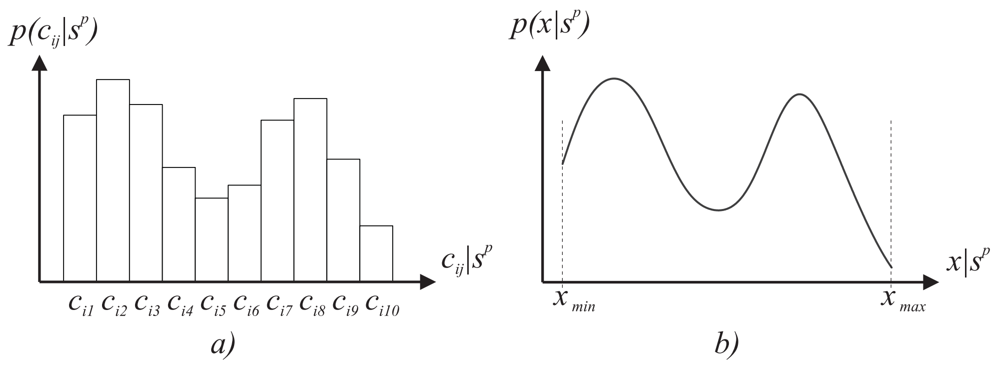


Figure 4.1: (a) Discrete probability distribution $P_d\left(c_{ij}|s^p\right)$ of a finite set $\{c_{i1},..,c_{i10}\} \in N(s^p)$ of available components. (b) Continuous probability density function $P_c\left(x|s^p\right)$ with possible range $x \in [x_{min}, x_{max}]$. The y-axis on both plots indicates the probability p. Note that $\sum_j p\left(c_{ij}|s^p\right) = \int_{x_{min}}^{x_{max}} p\left(x|s^p\right) dx = 1$.

1. Candidate solutions are constructed in a probabilistic way using a probability distribution over the search space;
2. The candidate solutions are used to modify the probability distribution in a way that is deemed to bias future sampling toward high quality solutions.

ACO algorithms for combinatorial optimization problems make use of a *pheromone model* in order to probabilistically construct solutions. A pheromone model is a set of so-called *pheromone trail parameters.* The numerical values of these pheromone trail parameters (that is, the pheromone values) reflect the search experience of the algorithm. They are used to bias the solution construction over time towards the regions of the search space containing high quality solutions.

In ACO for combinatorial problems, the pheromone values are associated with a finite set of discrete values related to the decisions that the ants make. This allows to represent the pheromone values in the form of a *pheromone table.* This is not possible in the continuous case, as the number of possible values is not finite. Hence, ACO$_\mathbb{R}$ uses rather a *solution archive* as way of describing the pheromone distribution over the search space. The solution archive contains a number of complete solutions to the problem. While a pheromone model in combinatorial optimization can be seen as an implicit memory of the search history, a solution archive is an explicit memory.[1]

The basic flow of the ACO$_\mathbb{R}$ algorithm is as follows. As a first step, the solution archive is initialized. Then, at each iteration, a number of solutions is probabilistically constructed

[1]A similar idea was proposed for by Guntsch and Middendorf [Guntsch and Middendorf, 2002b] for combinatorial optimization problems.

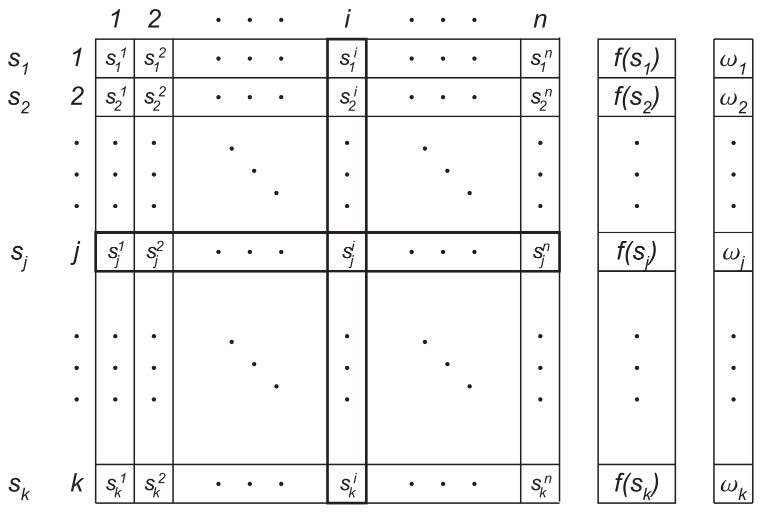


Figure 4.2: The structure of the solution archive. The solutions in the archive are sorted according to their quality (i.e., the value of the objective function $f(s)$), hence the position of a solution in the archive always corresponds to its rank.

by the ants. These solutions may be improved by any improvement mechanism (for example, local search or gradient techniques). Finally, the solution archive is updated with the generated solutions. In the following we outline the components of $\text{ACO}_{\mathbb{R}}$ in more details.

4.1.1 Archive structure, initialization, and update

$\text{ACO}_{\mathbb{R}}$ keeps a history of its search process by storing solutions in a *solution archive* T of dimension $|T| = k$. Given an n-dimensional continuous optimization problem and k solutions, $\text{ACO}_{\mathbb{R}}$ stores in T the values of the solutions' n variables and the value of their objective functions. The value of the i-th variable of the j-th solution is in the following denoted by s_j^i. Figure 4.2 shows the structure of the solution archive.

Before the start of the algorithm, the archive is initialized with k random solutions. At each algorithm iteration, first, a set of m solutions is generated by the ants and added to those in T. From this set of $k + m$ solutions, the m worst ones are removed. The remaining k solutions are sorted according to their quality (i.e., the value of the objective function) and stored in the new T. In this way, the search process is biased towards the best solutions found during the search. The solutions in the archive are always kept sorted based on their quality (i.e., the objective function values), so that the best solution is on top.

4.1.2 Probabilistic solution construction

Construction of new solutions by the ants is accomplished in an incremental manner, variable by variable. First, an ant chooses probabilistically one of the solutions in the archive. The probability of choosing solution j is given by:

$$p_j = \frac{\omega_j}{\sum_{r=1}^{k} \omega_r}, \tag{4.1}$$

where ω_j is a weight associated with solution j. The weight may be calculated using various formulas depending on the problem tackled. In the remainder of this paper we use a Gaussian function $g(\mu, \sigma) = g(1, qk)$, which was also used in our previous work [Socha and Dorigo, 2008]:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(j-1)^2}{2q^2k^2}}, \tag{4.2}$$

where, q is a parameter of the algorithm and k is the size of the archive. The mean of the Gaussian function is set to 1, so that the best solution has the highest weight.

The choice of the Gaussian function was motivated by its flexibility and non-linear characteristic. Thanks to its non-linearity, it allows for a flexible control over the weights. It is possible to give higher probability to a few leading solutions, while significantly reducing the probability of the remaining ones.

Once one of the solutions in the archive is chosen, an ant may start constructing a new solution. However, before the new solution is created, the algorithm attempts to de-correlate the variables of the solutions in the archive. It does so through a linear transformation of the soltuion archive while generating new solutions and reverting to original state afterwards. As this does not have an impact on the way the solutions are generated, we do not describe this process here. A detailed description of the de-correlation process implemented is provided in Appendix A.

The ant treats each problem variable $i = 1, ..., n$ separately. It takes the value s_j^i of the variable i of the chosen j-th solution and samples its neighborhood. This is done using a *probability density function* (PDF). Again, as in the case of choosing the weights, many different functions may be used. A PDF $P(x)$ must however satisfy the condition:

$$\int_{-\infty}^{\infty} P(x)dx = 1. \tag{4.3}$$

In this work, similarly to earlier publications presenting ACO$_{\mathbb{R}}$ [Socha and Dorigo, 2008], we use as PDF the Gaussian function:

$$P(x) = g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{4.4}$$

The function has two parameters that must be defined: μ, and σ. When considering variable i of solution j, we assign $\mu \leftarrow s_j^i$. Further, we assign σ:

$$\sigma \leftarrow \xi \sum_{r=1}^{k} \frac{|s_r^i - s_j^i|}{k-1} \ . \tag{4.5}$$

which is the average distance between the i-th variable of the solution s_j and the i-th variables of the other solutions in the archive, multiplied by a parameter ξ[2].

This whole process is repeated for each dimension $i = 1, ..., n$ in turn by each of the m ants.

An alternative description of the ACO$_{\mathbb{R}}$ aglorithm using the notion of Gaussian kernels is also possible. It has been used in some of our initial publications on the subject [Socha and Dorigo, 2008; Socha and Blum, 2006, 2007]. This alternative description is provided as Appendix B.

[2]Parameter ξ has an effect similar to that of the pheromone evaporation rate in ACO. The higher the value of ξ, the lower the convergence speed of the algorithm. While the pheromone evaporation rate in ACO influences the long term memory—i.e., the worst solutions are forgotten faster—ξ in ACO$_{\mathbb{R}}$ influences the way the long term memory is used—i.e., the new solutions are considered *closer* to known good solutions.

4.2 Positioning of $ACO_{\mathbb{R}}$

$ACO_{\mathbb{R}}$ is part of a rather large family of algorithms for continuous optimization. The description of the most of the important ones have been already provided in Chapter 2. In this section, we give a short summary and discuss, how $ACO_{\mathbb{R}}$ may be positioned in relation to the others.

For continuous optimization problems, a number of methods have been proposed in the literature. They include some ant-related methods [Bilchev and Parmee, 1995; Monmarché et al., 2000; Dréo and Siarry, 2002], as well as a more generic *swarm* inspired method such as Particle Swarm Optimization [Kennedy and Eberhart, 1995]. There are also many others: many metaheuristics have been originally developed for combinatorial optimization and later adapted to the continuous case. Examples include the Continuous Genetic Algorithm (CGA) [Chelouah and Siarry, 2000], Enhanced Simulated Annealing (ESA) [Siarry et al., 1997], or Enhanced Continuous Tabu Search (ECTS) [Chelouah and Siarry, 1999].

Additionally, there are also other methods that—similarly to ACO—explicitly use some notion of probability distribution estimation. Many of these algorithms have spawned from the general class of Evolutionary Algorithms (EAs). Examples include Evolutionary Strategies (ES) [Schwefel, 1981; Ostermeier et al., 1994; Hansen and Ostermeier, 2001], Iterated Density Estimation Algorithm (IDEA) [Bosman and Thierens, 2000], Mixed Bayesian Optimization Algorithm (MBOA) [Očenášek and Schwarz, 2002], or Population-Based Incremental Learning (PBIL) [Baluja and Caruana, 1995; Yuan and Gallagher, 2003]. Some of them, similarly to ACO, have been initially used for combinatorial optimization, and only later adapted to handle continuous domains.

In addition to all the algorithms mentioned so far, there are also many gradient-based algorithms for continuous optimization. They are fast, but they have some prerequisites. They are able to quickly find a local minimum, but they require the optimized function to be continuous and differentiable. Examples of such algorithms include the Newton method [Ralston and Rabinowitz, 1978], or the backpropagation algorithm [Rumelhart et al., 1986] routinely used for training artificial neural networks. The usefulness of gradient based algorithms is limited due to the prerequisites mentioned above. $ACO_{\mathbb{R}}$, as well as all other algorithms for continuous optimization mentioned earlier, do not have such limitations, which makes them much more general.

Last, but not least, there are the direct search methods, such as the simplex method [Nelder and Mead, 1965], or the Powell's method [Powell, 1964]. They are quick and efficient in most cases, but they only allow to find the nearest local optimum. They are however quite interesting as a method to complement $\text{ACO}_{\mathbb{R}}$ or other metaheuristics. Namely, they may be used as local search routines for the metaheuristics. In Section 4.3.3, we present some experiments involving the direct search methods, also used as local search for $\text{ACO}_{\mathbb{R}}$. Although the hybridized versions do not seem to have particular advantage in these experiments, the approach of improving performance of $\text{ACO}_{\mathbb{R}}$ with a local search is also used and evaluated more thoroughly in Chapter 5 with more encouraging results.

4.2.1 $\text{ACO}_{\mathbb{R}}$ and Other Swarm-Based Algorithms

The main type of swarm-based algorithms that we will refer to in this section are the ant-related algorithms. A single swarm-based algorithm that is not ant-related will be presented towards the end of this section.

There have been previous attempts to apply ant-based optimization algorithms to the continuous domain. Some attempts were more successful than others, but none of them was an extension of ACO to the continuous domain. Rather, they were new algorithms that also drew their initial inspiration from the behavior of ants. In the following paragraphs, we shortly present these algorithms and indicate how they differ from $\text{ACO}_{\mathbb{R}}$.

One of the first attempts to apply an ant-related algorithm to the continuous optimization problems was Continuous ACO (CACO) [Bilchev and Parmee, 1995]. In CACO the ants start from a point, called a *nest*, situated somewhere in the search space. The good solutions found are stored as a set of vectors, which originate in the nest. The ants at each iteration of the algorithm choose probabilistically one of the vectors. They then continue the search from the end-point of the chosen vector by making some random moves from there. The vectors are updated with the best results found. Although the authors of CACO claim that they draw inspiration from the original ACO formulation, there are important differences. They introduce the notion of *nest*, which does not exist in the ACO metaheuristic. Also, CACO does not perform an incremental construction of solutions, which is one of the main characteristics of the ACO metaheuristic. CACO does not qualify therefore to be an extension of ACO.

Another ant-related approach to continuous optimization is the API algorithm [Monmarché et al., 2000]. API does not claim to be based on the ACO metaheuristic. The ants perform their search independently, but starting from the same nest (the nest is moved periodically). The ants use only *tandem running*, a type of recruitment strategy. It is the only known algorithm among the ant-related algorithms published so far that allows to tackle both discrete and continuous optimization problems.

The third ant-based approach to continuous optimization is Continuous Interacting Ant Colony (CIAC) [Dréo and Siarry, 2002]. CIAC uses two types of communication between ants: stigmergic information (spots of pheromone deposited in the search space) and direct communication between ants. The ants move through the search space being attracted by pheromone laid in spots, and guided by some direct communication between individuals. Although also CIAC claims to draw its original inspiration from ACO, the differences are many: there is direct communication between ants and no incremental construction of solutions. As CACO, also CIAC does not qualify as an extension of ACO.

Finally, as mentioned at the beginning of this section, there is a well-known *swarm-based* algorithm for continuous optimization that is not ant-related. It is called Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995]. PSO works with a population of particles. These particles move in the search space with a certain velocity. The value and the direction of the velocity vector change according to the attractors in the search space. Each particle reacts to two such attractors. One is the best value found by the particle, and the other one is the best value found globally. PSO has been shown experimentally to perform well on many continuous optimization problems.

4.2.2 ACO$_\mathbb{R}$ and Evolutionary Algorithms

ACO for combinatorial optimization is similar to Evolutionary Algorithms (EAs) in many respects. Both ACO and EAs use some notion of probability distribution in order to find promising areas in the search space. This similarity is maintained when comparing ACO$_\mathbb{R}$ with EAs developed or adapted to continuous domains.

Kern *et al.* in [Kern et al., 2004] present an extensive comparison of several evolutionary algorithms dedicated to continuous optimization—from very simple ones to those that are quite advanced. We will now shortly present them.

The set of algorithms compared by Kern *et al.* contains three versions of Evolutionary Strategies (ES), and two other algorithms—the Mixed-Bayesian Optimization Algorithm (MBOA), and the Iterated Density Estimation Algorithm (IDEA). The simplest algorithm used in this comparison is (1+1) ES [Kern et al., 2004]. It is a simple ES with one parent generating one offspring per iteration. Only the individual representing the higher quality solution is kept. The next ES included in the comparison is Evolutionary Strategy with Cumulative Step Size Adaptation (CSA-ES) [Ostermeier et al., 1994; Kern et al., 2004]. It adapts the global step size by using the path traversed by the parent population over a number of generations. The third ES considered is CMA-ES—ES with Covariance Matrix Adaptation [Hansen and Ostermeier, 2001; Kern et al., 2004]. It is an extended version of CSA-ES, with de-randomized adaptation of the covariance matrix.

The first of the two algorithms that are not ES is IDEA, proposed by Bosman and Thierens [Bosman and Thierens, 2000]. It formalizes the Estimation of Distribution Algorithms (EDA) in continuous domains. To estimate the distribution of the parent population, IDEA exploits the fact that every multivariate joint probability distribution can be written as a *conditional factorization* of the following form: $P(x_i, ..., x_n) = \prod_{i=1}^{n} P(x_i|x_{i+1}, x_{i+2}, ..., x_n)$. The probabilistic model of the parent population is rebuilt in every generation.

The last algorithm considered in this comparison is MBOA [Očenášek and Schwarz, 2002]. It is a Bayesian network with local structures in the form of decision trees that captures the mutual dependencies among the parent individuals. The first EDA employing the Bayesian network model with decision trees was the hierarchical Bayesian Optimization Algorithm (hBOA) [Pelikan et al., 2000]. MBOA is an extension of hBOA from binary to continuous domains. In fact, MBOA is able to deal with discrete and continuous variables simultaneously, just like $\text{ACO}_\mathbb{R}$.

Each of these algorithms is different, but they all use some way of learning and modelling explicitly probability distributions. There are two ways these algorithms use the probability distributions. All versions of the ES incrementally update their probability distributions at each iteration. In contrast, IDEA and MBOA each time entirely rebuild them. $\text{ACO}_\mathbb{R}$ acts in this respect yet differently. Similarly to ACO for combinatorial optimization, in $\text{ACO}_\mathbb{R}$ ants use at each construction step a different, dynamically created probability distribution. This distribution depends on the previous construction steps and may be different for each ant. Hence, the $\text{ACO}_\mathbb{R}$ approach is closer to what IDEA and MBOA do, but it is even more fine-grained—several dynamically created probability distributions are used during one iteration.

All of the probability-learning algorithms compared by Kern *et al.* use some form of Gaussian function for modeling the probability distributions. (1+1)ES and CSA-ES use isotropic Gaussian distributions. IDEA uses one (or more—a mixture, if clustering is enabled) arbitrary Gaussian distribution. MBOA uses a concept somewhat similar to the one used by ACO$_{\mathbb{R}}$—Gaussian kernel distribution, but defined on partitions of the search space.

4.3 Experimental Setup and Results

In this section, we present the experimental setup for evaluating the performance of ACO$_{\mathbb{R}}$ and the results obtained. In order to have an overview of the performance of ACO$_{\mathbb{R}}$ in comparison to other methods for continuous optimization, we use the typical benchmark test functions that have been used in the literature for presenting the performance of different methods and algorithms.

Obviously, it is impractical to compare ACO$_{\mathbb{R}}$ to every method that has been used for continuous optimization in the past. Hence, we decided to limit our comparison to other metaheuristics used for this purpose. We then decided to divide those metaheuristics into three groups, based on their similarity to ACO$_{\mathbb{R}}$:

- Probability-learning methods—methods which explicitly model and sample probability distributions.
- Ant-related methods—methods that claim to draw inspiration from the behavior of ants.
- Other metaheuristics originally developed for combinatorial optimization and later adapted to continuous domains.

We have used a slightly different experimental methodology for each comparison in order to make the results obtained by ACO$_{\mathbb{R}}$ as comparable as possible to those obtained with the other methods.

It is important to emphasize that—unlike combinatorial optimization—the comparison of algorithms for continuous optimization is usually *not* done based on CPU time. In case of combinatorial optimization, usually each algorithm is given the same amount of CPU time and the results obtained within that time are compared. This makes the comparison of different algorithms complicated, as the CPU time depends significantly on the programming language used, the compiler, the skills of the programmer, and finally also on the machine(s) used for running the experiments. Hence, in case of combinatorial optimization it is strongly recommended to re-implement all the algorithms used in the comparison in order to make it fair. This still does not guarantee an entirely fair comparison, as it is difficult to assure the same amount of effort being put in optimization of the code of all the implemented algorithms.[3]

In contrast, the great majority of the papers on continuous optimization algorithms use as criterion of comparison the *number of function evaluations* needed to achieve a certain solution quality [Kern et al., 2004; Bilchev and Parmee, 1995; Monmarché et al., 2000; Dréo and Siarry, 2004]. Such an approach gives several key advantages: it solves the problem of the algorithms being implemented using different programming languages; it is insensitive to the code-optimization skills of the programmer (or to the compiler used); and it allows comparing easily the results obtained on different machines. The drawback of this approach is that it does not take into consideration the time-complexity of the algorithms compared. However, in view of the other numerous disadvantages of using the CPU time as a criterion, it is an acceptable methodology, and we adopt it in this paper. Also, in case of continuous optimization problems, usually the majority of the time of the execution of the algorithm is spent on evaluating the objective function, so in general the measuring the number of function evaluations is a good way to also approximate the time needed for the algorithm to run.

The use of the number of function evaluations as a criterion allows us to run the experiments only with $\text{ACO}_{\mathbb{R}}$ and compare the results obtained to those found in the literature. Additionally, in order to ensure a fair comparison, we replicate carefully the experimental setup (in particular: initialization interval, parameter tuning methodology, and termination condition) used by the competing algorithms.

4.3.1 $\text{ACO}_{\mathbb{R}}$ Compared with the Probability-Learning Methods

There are many metaheuristics that explicitly model and sample probability distributions. In this comparison we use the algorithms tested by Kern *et al.* [Kern et al., 2004], which have been already described in Sec. 4.2.2.

The test functions used for comparison range from very simple to quite complex, and were chosen by Kern *et al.* for their particular characteristics. In particular, they test the algorithms' robustness when applied to a linear transformation of the considered problem. For a fair comparison of $\text{ACO}_{\mathbb{R}}$'s performance, we have used the same test

[3]Automatic parameter tuning procedures such as F-Race [Birattari et al., 2002; Birattari, 2005] can help in this respect.

Table 4.1: Summary of the test functions used for comparing $\text{ACO}_{\mathbb{R}}$ with other methods explicitly estimating probability distributions. We used $n = 10$ dimensions for all the test functions listed here.

Function	Formula
Plane (PL) $\vec{x} \in [0.5, 1.5]^n,\ n = 10$	$f_{PL}(\vec{x}) = x_1$
Diagonal Plane (DP) $\vec{x} \in [0.5, 1.5]^n,\ n = 10$	$f_{DP}(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i$
Sphere (SP) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{SP}(\vec{x}) = \sum_{i=1}^{n} x_i^2$
Ellipsoid (EL) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{EL}(\vec{x}) = \sum_{i=1}^{n}(100^{\frac{i-1}{n-1}} x_i)^2$
Cigar (CG) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{CG}(\vec{x}) = x_1^2 + 10^4 \sum_{i=2}^{n} x_i^2$
Tablet (TB) $\vec{x} \in [-3, 7]^n,\ n = 10$	$f_{TB}(\vec{x}) = 10^4 x_1^2 + \sum_{i=2}^{n} x_i^2$
Rosenbrock (R_n) $\vec{x} \in [-5, 5]^n,\ n = 10$	$f_{Rn}(\vec{x}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2$

functions. They are listed in Tab. 4.1. *Plane* and *Diagonal Plane* functions are maximization problems (the goal is to reach $\epsilon_{max} = 10^{10}$), the others are minimization problems (required accuracy is $\epsilon_{min} = 10^{-10}$). In addition to the functions listed in Tab. 4.1, the comparison also uses randomly rotated versions of the *Ellipsoid*, *Cigar*, and *Tablet* functions. In all the experiments we used 10-dimensional versions of the functions. The performance was judged based on the number of function evaluations needed to reach the stopping condition. The stopping condition used was the required accuracy: $f > \epsilon_{max}$ for maximization problems, and $|f - f^*| < \epsilon_{min}$ for minimization problems, where f is the value of the best solution found by $\text{ACO}_{\mathbb{R}}$, and f^* is the (known *a priori*) optimal solution for the given test problem.

When tackling continuous optimization *test* problems (i.e., those for which the optimal solution is known *a priori*), one has to do the initialization with caution.[4] In fact, it has been shown in the literature that if the initialization is done close to the optimum, or it is *symmetric* around the optimum, it may introduce an undesired bias [Fogel and Bayer, 1995]. It has also been demonstrated that the results obtained by some algorithms differ significantly, when the symmetric and asymmetric initialization are used [Eiben and

[4]In the case of real-world problems, the optimal solutions are often not known, and hence the initialization intervals have to be chosen based on some other estimates or even at random.

Bäck, 1997; Chellapilla and Fogel, 1999].

Due to these issues, special care must be taken in order to use (when possible) initializations that do not introduce the bias. Such an approach is often called *skewed initialization* [Eiben and Bäck, 1997; Deb et al., 2002]. Also, any comparison of the continuous optimization algorithms should explicitly take into account the initialization used by each of the algorithms. In all our test runs we use initialization intervals identical to those used by the methods we compare to.

In order to compare the performance of $\text{ACO}_{\mathbb{R}}$ to that of the algorithms presented in [Kern et al., 2004], we have adopted the same methodology for conducting the experiments. We have done 20 independent runs on each of the test problems. Concerning parameters tuning, Kern *et al.* used the parameters as suggested by the authors, with the exception of the size of the population[5] used. The latter was chosen separately for each algorithm-problem pair—the smallest population from the set $p \in [10, 20, 50, 100, 200, 400, 800, 1600, 3200]$ was chosen, which still allowed to achieve the required accuracy in all the runs. The summary of the parameters we used for $\text{ACO}_{\mathbb{R}}$ is presented in Tab. 4.2. The archive size $k = 50$ was used for all test problems.

Table 4.2: Summary of the parameters used by $\text{ACO}_{\mathbb{R}}$. The solution archive size varied depending on the test function, as done by Kern *et al.*[Kern et al., 2004].

Parameter	Symbol	Value
no. of ants used in an iteration	m	2
speed of convergence	ξ	0.85
locality of the search process	q	10^{-4}
archive size	k	50

Table 4.3 presents the results obtained by $\text{ACO}_{\mathbb{R}}$ compared to those obtained by the algorithms tested by Kern *et al.*. For each test problem, the relative median performance for all the algorithms is reported, 1.0 being the best algorithm (lowest median number of function evaluations). Numbers for the other algorithms indicate *how many times* larger was their median number of function evaluations in relation to the best one on a given test function. For the best algorithm also the actual median number of function evaluations is supplied in curly brackets.

[5]In case of the evolutionary algorithms, there is the notion of *population* size. In case of $\text{ACO}_{\mathbb{R}}$, the respective parameter is the *archive* size.

Table 4.3: Results obtained by $ACO_{\mathbb{R}}$ compared to those obtained by other probability-learning algorithms—based on [Kern et al., 2004]. Reported is the relative median number of function evaluations. The actual median number of function evaluations is given in curly brackets only for the best performing algorithm on a given problem. Results marked with * indicate that the required accuracy was not reached in every run.

Function	$ACO_{\mathbb{R}}$	(1+1)ES	CSA-ES	CMA-ES	IDEA	MBOA
Plane	**1.0** (175)	4.5	7.2	6.3	∞	4970
Diag. Plane	**1.0** (170)	4.9	7.4	6.4	∞	241
Sphere	1.1	**1.0** (1370)	1.6	1.3	5.0	48
Ellipsoid	2.6	66	110	**1.0** (4450)	1.6	14
Cigar	1.4	610	800	**1.0** (3840)	4.6	12
Tablet	**1.0** (2567)	46	65	1.7	2.9	24
Rot. Ellipsoid	2.8	64	110	**1.0** (4490)	13	*1800
Rot. Cigar	1.4	600	800	**1.0** (3840)	38	*2100
Rot. Tablet	**1.0** (2508)	44	63	1.7	12	*1600
Rosenbrock	*1.1	*51	180	**1.0** (7190)	*210	*1100

The performance of $ACO_{\mathbb{R}}$ is quite good. It achieves the best result in four out of 10 test problems. Also, when it is not the best, it is only slightly worse than the best algorithm (CMA-ES). Unlike some of the competing algorithms, $ACO_{\mathbb{R}}$ is performing well in case of both maximization (Plane and Diagonal Plane) and minimization problems. It is able to adjust well to problems that are scaled differently in different directions (such as Ellipsoid, Cigar, and Tablet functions), and the rotation of the test function does not have any impact on the performance. In this respect only CMA-ES performs similarly well.

Due to unavailability of full result sets and missing results of some algorithms for some test functions, it is impossible to do any serious statistical significance analysis. In order to enable future researchers to perform more statistically sound comparisons, the full set of results obtained with $ACO_{\mathbb{R}}$ is available online.[6]

4.3.2 $ACO_{\mathbb{R}}$Compared with Other Ant-Related Approaches and Other Metaheuristics

As we have mentioned earlier, there were other ant-related methods proposed for continuous optimization in the past. The very first one—called Continuous ACO (CACO)—was proposed by Bilchev and Parmee [Bilchev and Parmee, 1995], and also used later [Wodrich and Bilchev, 1997; Mathur et al., 2000]. Others include the API algorithm by Monmarché [Monmarché et al., 2000], and Continuous Interacting Ant Colony (CIAC), proposed by Dréo and Siarry [Dréo and Siarry, 2002, 2004]. These algorithms have been already described in Sec. 4.2.1. They were tested by their authors on some classical test functions and compared with other metaheuristics that had been primarily developed for combinatorial optimization and later adapted to continuous domain. The continuous versions of these metaheuristics include in particular Continuous Genetic Algorithm (CGA) [Chelouah and Siarry, 2000], Enhanced Continuous Tabu Search (ECTS) [Chelouah and Siarry, 1999], Enhanced Simulated Annealing (ESA) [Siarry et al., 1997], and Differential Evolution (DE) [Storn and Price, 1995].

In these comparisons, the parameters chosen by the authors of the competing algorithms were essentially picked by a simple trial and error procedure. Hence, we have also refrained from doing an extensive parameter tuning. Instead, we used almost identical parameters to those used earlier for comparing $ACO_{\mathbb{R}}$ with probability-learning methods. The only exception was parameter q, which required a slightly larger value in order to increase the robustness of $ACO_{\mathbb{R}}$ on the multimodal functions used in this set of experiments. The value $q = 0.1$ was used. More analysis of the influence of parameter q on the performance of $ACO_{\mathbb{R}}$ is provided in Sec. 4.3.4.

To compare $ACO_{\mathbb{R}}$ with all these algorithms, we have run $ACO_{\mathbb{R}}$ on a number of test functions used by the other algorithms, and we followed the original experimental setup in terms of the initialization interval and required accuracy. The list of test functions on which we run $ACO_{\mathbb{R}}$, along with the number of dimensions used and the initialization interval, is presented in Tab. 4.4 & 4.5. A more detailed description of the test functions used may be found in the literature [Björkman and Holmström, 1999; Chelouah and

[6]http://iridia.ulb.ac.be/∼ksocha/aco_r.html

Table 4.4: First part of the test functions used for comparing ACO$_{\mathbb{R}}$ to other ant-related algorithms and other metaheuristics adapted for continuous optimization.

Function	Formula
Branin RCOS (RC) $\vec{x} \in [-5, 15]^n,\ n = 2$	$f_{RC}(\vec{x}) = \left(x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_i + 10$
B_2 $\vec{x} \in [-100, 100]^n,\ n = 2$	$f_{B2}(\vec{x}) = x_1^2 + 2x_2^2 - \frac{3}{10}\cos(3\pi x_1) - \frac{2}{5}\cos(4\pi x_2) + \frac{7}{10}$
Easom (ES) $\vec{x} \in [-100, 100]^n,\ n = 2$	$f_{ES}(\vec{x}) = -\cos(x_1)\cos(x_2)e^{-\left((x_1-\pi)^2+(x_2-\pi)^2\right)}$
Goldstein & Price (GP) $\vec{x} \in [-2, 2]^n,\ n = 2$	$f_{gp}(\vec{x}) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 - 48x_2 - 36x_1x_2 + 27x_2^2))$
Martin & Gaddy (MG) $\vec{x} \in [-20, 20]^n,\ n = 2$	$f_{MG}(\vec{x}) = (x_1 - x_2)^2 + \left(\frac{x_1+x_2-10}{3}\right)^2$
Rosenbrock (R_n) $\vec{x} \in [-5, 10]^n,\ n = 2, 5$	$f_{Rn}(\vec{x}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2$
Zakharov (Z_n) $\vec{x} \in [-5, 10]^n, n = 2, 5$	$f_{Zn}(\vec{x}) = \left(\sum_{j=1}^{n} x_j^2\right) + \left(\sum_{j=1}^{n} \frac{jx_j}{2}\right)^2 + \left(\sum_{j=1}^{n} \frac{jx_j}{2}\right)^4$
De Jong (DJ) $\vec{x} \in [-5.12, 5.12]^n,\ n = 3$	$f_{DJ}(\vec{x}) = x_1^2 + x_2^2 + x_3^2$
Griewank (GR_n) $\vec{x} \in [-5.12, 5.12]^n, n = 10$	$f_{GRn}(\vec{x}) = \left(\frac{1}{10} + \left(\sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\right)\right)^{-1}$
Sphere Model (SM) $\vec{x} \in [-5.12, 5.12]^n,\ n = 6$	$f_{SM}(\vec{x}) = \sum_{i=1}^{n} x_i^2$

Siarry, 2000; Dréo and Siarry, 2004]. Following the experimental setup described in the literature, we performed 100 independent runs, and we used the following stopping criterion (as used by the other algorithms in this comparison):

$$|f - f^*| < \epsilon_1 f + \epsilon_2\ , \tag{4.6}$$

where f is the value of the best solution found by ACO$_{\mathbb{R}}$, f^* is the (known *a priori*) optimal value for the given test problem, and ϵ_1 and ϵ_2 are respectively the relative and absolute errors. For all the test runs of ACO$_{\mathbb{R}}$, we used $\epsilon_1 = \epsilon_2 = 10^{-4}$, following the values reported in the literature.

Table 4.5: Second part of the test functions used for comparing ACO$_{\mathbb{R}}$ to other ant-related algorithms and other metaheuristics adapted for continuous optimization.

Function	Formula
Hartmann ($H_{3,4}$) $\vec{x} \in [0,1]^n$ $n = 4$	$f_{H3,4}(\vec{x}) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2}$ $a_{ij} = \left\{ \begin{array}{ccc} 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \\ 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \end{array} \right\}, c_i = \left\{ \begin{array}{c} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{array} \right\},$ $p_{ij} = \left\{ \begin{array}{ccc} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.0381 & 0.5743 & 0.8828 \end{array} \right\}$
Hartmann ($H_{6,4}$) $\vec{x} \in [0,1]^n$ $n = 6$	$f_{H6,4}(\vec{x}) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2}$ $a_{ij} = \left\{ \begin{array}{cccccc} 10.00 & 3.00 & 17.0 & 3.50 & 1.50 & 8.00 \\ 0.05 & 10.00 & 17.0 & 0.10 & 8.00 & 14.00 \\ 3.00 & 3.50 & 1.70 & 10.00 & 17.00 & 8.00 \\ 17.00 & 8.00 & 0.05 & 10.00 & 0.10 & 14.00 \end{array} \right\}, c_i = \left\{ \begin{array}{c} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{array} \right\},$ $p_{ij} = \left\{ \begin{array}{cccccc} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{array} \right\}$
Shekel ($S_{4,k}$) $k = 5, 7, 10$ $\vec{x} \in [0,10]^n$ $n = 4$	$f_{S4,k}(\vec{x}) = -\sum_{i=1}^{k} \left((\vec{x} - \vec{a_i})^T(\vec{x} - \vec{a_i}) + c_i\right)^{-1}$ $a_{ij} = \left\{ \begin{array}{cccc} 4.0 & 4.0 & 4.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \\ 8.0 & 8.0 & 8.0 & 8.0 \\ 6.0 & 6.0 & 6.0 & 6.0 \\ 3.0 & 7.0 & 3.0 & 7.0 \\ 2.0 & 9.0 & 2.0 & 9.0 \\ 5.0 & 5.0 & 3.0 & 3.0 \\ 8.0 & 1.0 & 8.0 & 1.0 \\ 6.0 & 2.0 & 6.0 & 2.0 \\ 7.0 & 3.6 & 7.0 & 3.6 \end{array} \right\}, c_i = \left\{ \begin{array}{c} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{array} \right\}$

Tables 4.6 & 4.7 present the results obtained by ACO$_{\mathbb{R}}$, and respectively other ant-related algorithms and other metaheuristics. Some of the test functions used in this section are multimodal—they have many local optima, where algorithms may get stuck. Hence, the results presented not only give an overview of the mean performance (i.e., number of function evaluations), but also give the success rate—percentage of successful runs, when the algorithm found the global optimum.

When compared with other ant-related methods (Tab. 4.6), ACO$_{\mathbb{R}}$ is a clear winner—all other algorithms in this category require many more function evaluations in order to reach the required accuracy. ACO$_{\mathbb{R}}$ is simply a much more effective approach. As mentioned earlier, serious statistical analysis of the results is not possible due to unavailability of the full data result sets for the other algorithms.

Table 4.6: Results obtained by ACO$_\mathbb{R}$ compared to those obtained by other ant-related algorithms. Reported is the relative mean number of function evaluations. The actual mean number of function evaluations is given in curly brackets only for the best performing algorithm on a given problem. Numbers in square brackets indicate the percentage of successful runs (i.e., when the algorithm did not get stuck in a local optimum). When the percentage is not given—all the runs were successful. Note that for some algorithms, the results on some test functions were not available.

Test Function	ACO$_\mathbb{R}$	CACO	API	CIAC
Rosenbrock (R_2)	**1.0** (820)	8.3	12	14
Sphere	**1.0** (781)	28	13	64
Griewank (GR_{10})	**1.0** [61%] (1390)	36	-	36 [52%]
Goldstein & Price	**1.0** (384)	14	-	61 [56%]
Martin & Gaddy	**1.0** (345)	5	-	34 [20%]
B_2	**1.0** (544)	-	-	22
Rosenbrock (R_5)	**1.0** [97%] (2487)	-	-	16 [90%]
Shekel ($S_{4,5}$)	**1.0** [57%] (787)	-	-	50 [5%]

When ACO$_\mathbb{R}$ is compared to other metaheuristics adapted for continuous domains, such as CGA, ECTS, or ESA (Tab. 4.7), it is not so clear anymore which of the algorithms is best. Almost each of the algorithms presented is best performing for at least some of the test problems (with the exception of ESA). While ACO$_\mathbb{R}$ is the winner for Easom, both Hartmann, Griewank, and one Zakharov problem, CGA is the leader for B_2 and all three Shekel problems. In turn ECTS is the best one on Branin RCOS, Goldstein & Price, one Zakharov, and both Rosenbrock problems. ESA is the only one that is not best on any of the problems, and results of DE—although promising—best for De Jong—are available for only very few problems.

The differences in performance between the metaheuristics are however rather small—they rarely exceed the factor of 2.0. It may be hence concluded that while each of these algorithms is different, they all perform similarly well, including ACO$_\mathbb{R}$ proposed in this paper. For particular real-world applications some of them may be better suited than

Table 4.7: Mean number of function evaluations obtained by $ACO_{\mathbb{R}}$ comprised to other metaheuristics adapted to continuous domains. The actual mean is given in curly brackets only for the best performing algorithm on a given problem. Numbers in square brackets indicate the percentage of successful runs (when not given—100% successful). Dash indicates that results were not available.

Test Function	$ACO_{\mathbb{R}}$	CGA	ECTS	ESA	DE
Branin RCOS	3.5	2.5	**1.0** (245)	-	-
B_2	1.3	**1.0** (430)	-	-	-
Easom	**1.0** [98%] (772)	1.9	-	-	-
Goldstein & Price	1.7	1.8	**1.0** (231)	3.4	-
Rosenbrock (R_2)	1.7	2.0	**1.0** (480)	1.7	1.3
Zakharov (Z_2)	1.5	3.2	**1.0** (195)	81	-
De Jong	1.0	1.9	-	-	**1.0** (392)
Hartmann ($H_{3,4}$)	**1.0** (342)	1.7	1.6	2.0	-
Shekel ($S_{4,5}$)	1.3 [57%]	**1.0** [76%] (610)	1.4 [75%]	1.9 [54%]	-
Shekel ($S_{4,7}$)	1.1 [79%]	**1.0** [83%] (680)	1.3 [80%]	1.8 [54%]	-
Shekel ($S_{4,10}$)	1.1 [81%]	**1.0** [83%] (650)	1.4 [80%]	1.8 [50%]	-
Rosenbrock (R_5)	1.2 [97%]	1.9	**1.0** (2142)	2.5	-
Zakharov (Z_5)	**1.0** (727)	1.9	3.1	96	-
Hartmann ($H_{6,4}$)	**1.0** (722)	1.3	2.1	3.7	-
Griewank (Gr_{10})	**1.0** [61%] (1390)	-	-	-	9.2

others. However, the differences are not large, and it is not trivial to say when any of these algorithms should be preferred over the others. Again, as mentioned earlier, serious statistical analysis of the results is not possible due to unavailability of the full result data sets for the other algorithms.

4.3.3 ACO$_{\mathbb{R}}$ compared with Direct Search Methods

As mentioned in Chapter 2, there exist also a set of direct search methods for continuous optimization. They include among others the Nelder-Mead (or Simplex) method the Powell method. These methods, similarly to metaheuristics, do not need any additional information about the problem being solved. Hence, they may be potentially applied to the same problems as our ACO$_{\mathbb{R}}$ proposed algorithm. The direct search methods are known to perform well on many real-world problems. Their main disadvantage is that they search for local minima rather than for a global one. Hence, they are often used either as local search methods for metaheuristics, or as random-restart local search algorithms, where multiple runs of a direct search method are used in order to increase the chances of finding the global optimum.

This section focuses on how our ACO$_{\mathbb{R}}$ algorithm compares to these direct search methods on different types of problems. In order to have a proper overview, we have chosen three continuous optimization test problems of different characteristics. We chose the Rosenbrock function ($n = 10$) as an example of a difficult unimodal test problem, the Shekel (4,7) function as an example of low dimensional ($n = 4$) multimodal test problem with few local optima, and finally an Ackley function ($n = 10$), as an example of a multimodal test function with many local optima. For all the test functions we used the earlier mentioned *skewed initialization* in order to avoid bias of population based methods towards the center of the search space.

As examples of the direct search methods, we used the Simplex (S) method and the Powell (P) method. In order to have a proper perspective, we have tested several different configurations, including: ACO$_{\mathbb{R}}$ algorithm alone, random-restart versions of Simplex (RRS) and Powell (RRP), as well as hybrids of ACO$_{\mathbb{R}}$ with respectively Simplex (ACO$_{\mathbb{R}}$-S) or Powell (ACO$_{\mathbb{R}}$-P) algorithm as a local search. We have implemented the RRS algorithm using the `optim` function in R, which provides the implementation of the Simplex algorithm. The RRP was implemented with the use of package `powell`, which provides the implementation of the Powell algorithm, also in R.

For each of the test problems, we have done a separate set of experiments including tuning of the algorithms' parameters. In the following subsections, we describe the results of each of these experiments.

Rosenbrock Function

The first problem that we tackled was the Rosenbrock test function with $n = 10$ dimensions. This function has already been used in Section 4.3.1. ACO$_\mathbb{R}$, ACO$_\mathbb{R}$-S, and ACO$_\mathbb{R}$-P required choosing appropriate parameters. We have accomplished this using the F-Race method mentioned earlier. Table 4.8 presents the set of parameters chosen for these algorithms.

Table 4.8: Summary of the parameters used by ACO$_\mathbb{R}$, ACO$_\mathbb{R}$-S, and ACO$_\mathbb{R}$-P for the Rosenbrock problem.

Symbol	ACO$_\mathbb{R}$	ACO$_\mathbb{R}$-S	ACO$_\mathbb{R}$-P
m	2	2	2
ξ	1	0.8	0.8
q	0.01	10^{-4}	10^{-4}
k	200	20	10

The experiment aimed at establishing how many function evaluations would each of the algorithms need to optimize the 10-dimensional Rosenbrock function with accuracy of at least $e = 10^{-4}$. We have done 100 independent runs of each algorithm. Table 4.9 presents the values of the minimum, median, and maximum number of objective function evaluations needed to achieve the required accuracy by each of the algorithms.

Table 4.9: Summary of the results obtained by ACO$_\mathbb{R}$, its hybrids with direct search methods, and the random-restart versions of the direct search methods on the Rosenbrock problem. The dashes indicate when the solution could not be found with required accuracy even with one million of function evaluations.

Alg.	Min	Median	Max
ACO$_\mathbb{R}$	16726	18050	20284
ACO$_\mathbb{R}$-S	86394	159044	218460
ACO$_\mathbb{R}$-P	2093	2840	3409
RRS	-	-	-
RRP	790	1439	4808

After analyzing the results obtained, it is clear that the Powell method is the most efficient for solving this problem. Hybridizing the Powell method with ACO$_\mathbb{R}$ does not improve the results. Most likely, $\mathrm{ACO}_{\mathbb{R}}$ adds an unnecessary overhead, which increases the number of evaluations needed to solve the problem. Contrary to the Powell method, the Simplex method does not seem to be able to deal very well with the Rosenbrock problem. The random-restart version of the Simplex algorithm could not find the optimum solution even with one million of function evaluations. A hybrid of $\mathrm{ACO}_{\mathbb{R}}$ with the Simplex method as a local search did reach the optimum, but it took on average 100 times more function evaluations than the Powell method, and about 10 times more than $\mathrm{ACO}_{\mathbb{R}}$ alone. Note that $\mathrm{ACO}_{\mathbb{R}}$ was run here with different parameters than in Section 4.3.1. While there $\mathrm{ACO}_{\mathbb{R}}$ was converging faster, there were some unsuccessful runs. Here, more robust parameter settings were used, which allowed to obtain 100% success rate, that is, all $\mathrm{ACO}_{\mathbb{R}}$ runs resulted in finding the optimum solution with required accuracy.

Shekel (4,7) Function

The second function that we used to compare the performance of $\mathrm{ACO}_{\mathbb{R}}$ with direct search methods, was Shekel (4,7). It was used before in Section 4.3.2. It is a multimodal function with $n = 4$ dimensions, and few local optima. We have chosen the parameters for $\mathrm{ACO}_{\mathbb{R}}$, $\mathrm{ACO}_{\mathbb{R}}$-S, and $\mathrm{ACO}_{\mathbb{R}}$-P using the F-Race method mentioned earlier. Table 4.10 presents these parameters.

Table 4.10: Summary of the parameters used by $\mathrm{ACO}_{\mathbb{R}}$, $\mathrm{ACO}_{\mathbb{R}}$-S, and $\mathrm{ACO}_{\mathbb{R}}$-P algorithms for the Shekel (4,7) problem.

Symbol	$\mathrm{ACO}_{\mathbb{R}}$	$\mathrm{ACO}_{\mathbb{R}}$-S	$\mathrm{ACO}_{\mathbb{R}}$-P
m	2	2	2
ξ	1.1	0.2	1
q	0.4	0.7	0.6
k	50	100	20

Similarly to the Rosenbrock function, we measured the number of function evaluations required to reach the accuracy of $e = 10^{-4}$. We have done 100 independent runs of each algorithm. Table 4.11 presents the values of the minimum, median, and maximum number of the objective function evaluations needed to achieve the required accuracy by each of the algorithms.

Table 4.11: Summary of the results obtained by $ACO_{\mathbb{R}}$, its hybrids with direct search methods, and the random-restart versions of the direct search methods on the Shekel (4,7) problem.

Alg.	Min	Median	Max
$ACO_{\mathbb{R}}$	1376	1756	2698
$ACO_{\mathbb{R}}$-S	704	726	2156
$ACO_{\mathbb{R}}$-P	144	263	667
RRS	171	833	2515
RRP	57	109	526

Clearly, for the Shekel (4,7) problem, both direct search methods are much more efficient than $ACO_{\mathbb{R}}$. Powell is also much better than the Simplex method. Hybridizing $ACO_{\mathbb{R}}$ with the direct search methods improves the $ACO_{\mathbb{R}}$ performance, but it is still worse than the random-restart version of the respective direct search method alone.

The results show that a random-restart direct search method is quite efficient for a problem of few dimensions and not too many local optima. In the worst case only few iterations allow to find the global optimum. While for the direct search methods it is sufficient to start with an initial point that is the right *valley* to be certain to find the optimum, this is more complicated for $ACO_{\mathbb{R}}$. In the Shekel (4,7) problems, the valleys of all optima are rather large and the quality of local optima does not differ significantly. Hence, $ACO_{\mathbb{R}}$ tends to sample many of them at the same time, and it takes some tome before it converges to the correct one. Therefore, this experiment shows that random-restart direct search heuristics significantly outperform $ACO_{\mathbb{R}}$ for this type of problems.

Ackley Function

The third and final test function that we chose for comparing $ACO_{\mathbb{R}}$ to direct search methods, is the Ackley function [Ackley, 1987]. The Ackley function is a multimodal function with a large number of local optima:

$$f_{ac} = -20\exp\left[-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right] - \exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos\left(2\pi x_i\right)\right] + 20 + e \tag{4.7}$$

For our experimentation, we used the Ackley function with $n = 10$ dimensions. Usually in the literature, the domain of the Ackley function is defined as $X \in (-32, 32)^n$. This however leads to a bias performance of population-based methods, as the optimum is found exactly in the center of the search space, $X^* = \{0, 0, ..., 0\}$. In order to avoid this, we used the *skewed initialization*, that is, we used the domain of the same size, but shifted off the center, $X \in (-16, 48)^n$.

As before, we have chosen the parameters for ACO$_\mathbb{R}$, ACO$_\mathbb{R}$-S, and ACO$_\mathbb{R}$-P using the F-Race method. Table 4.12 presents these parameters.

Table 4.12: Summary of the parameters used by ACO$_\mathbb{R}$, ACO$_\mathbb{R}$-S, and ACO$_\mathbb{R}$-P algorithms for the Ackley problem.

Symbol	ACO$_\mathbb{R}$	ACO$_\mathbb{R}$-S	ACO$_\mathbb{R}$-P
m	2	2	2
ξ	0.9	0.5	0.5
q	0.06	10^{-4}	10^{-3}
k	50	50	50

The performance of the algorithms was compared using the number of function evaluations needed to reach the accuracy of $e = 10^{-4}$. We have done 100 independent runs of each algorithm. Table 4.13 presents the values of the minimum, median, and maximum number of the objective function evaluations needed to achieve the required accuracy by each of the algorithms.

Table 4.13: Summary of the results obtained by ACO$_\mathbb{R}$, its hybrids with direct search methods, and the random-restart versions of the direct search methods on the Ackley problem. The dashes indicate, when the solution could not be found with required accuracy even with one million of function evaluations.

Alg.	Min	Median	Max
ACO$_\mathbb{R}$	2206	2512	2794
ACO$_\mathbb{R}$-S	503046	604632	665586
ACO$_\mathbb{R}$-P	20422	27708	39962
RRS	-	-	-
RRP	-	-	-

The results are quite different than on the other two test problems. Both random-restart versions of the direct search heuristics failed to find the optimum of the Ackley function even when allowed one million of function evaluations. This is most likely due to the fact that the Ackley function has so many local optima that finding a starting point that would allow the direct search method to reach the global optimum is very difficult when doing it at random. A higher level strategy is probably needed to guide the search process towards the promising regions of the search space. This is supported by the fact that the hybrids of $\mathrm{ACO}_{\mathbb{R}}$ with either of the direct search methods managed to find the optimum.

However, the most interesting result is the one obtained by the $\mathrm{ACO}_{\mathbb{R}}$ algorithm alone. It performs on average 10 times better than $\mathrm{ACO}_{\mathbb{R}}$ and Powell hybrid and 200 times better than $\mathrm{ACO}_{\mathbb{R}}$ and Simplex hybrid. The poor performance of the hybridized versions of $\mathrm{ACO}_{\mathbb{R}}$ may be explained. While $\mathrm{ACO}_{\mathbb{R}}$ is able to guide the search process towards the global optimum, the direct search methods used as local search use many function evaluations in order to find each local optimum on the way. However, this is not really necessary to converge to the global optimum. The $\mathrm{ACO}_{\mathbb{R}}$ algorithm when used alone can therefore quickly converge towards the global optimum, and only then intensify the search to reach the required accuracy. Hence, for problems with many local optima and a general structure that may be exploited by a higher level strategy, $\mathrm{ACO}_{\mathbb{R}}$ appears to be much more efficient than any of the two direct search methods investigated.

Conclusions

The results of the experiments presented in this section lead to quite interesting observations. The first, and the most obvious one is that depending on the problem at hand, $\mathrm{ACO}_{\mathbb{R}}$ may outperform direct search methods. This is particularly the case for complex multimodal problems with many local optima.

Another observation is that for all the problems investigated, the performance of the Simplex method is never better than the Powell method. Hence, if the direct search methods are considered, the Powell method rather than Simplex should be used.

The third, and probably the most interesting observation is the fact that for none of the problems investigated, a hybridized version of $\mathrm{ACO}_{\mathbb{R}}$ was the overall winner. For any of the problems, either the random-restart Powell method was the best performing one (Rosenbrock and Shekel (4,7)), or $\mathrm{ACO}_{\mathbb{R}}$ alone was the best one (Ackley). Hence, it appears that there is no added value in hybridizing $\mathrm{ACO}_{\mathbb{R}}$ with a direct search method as a local search. Although adding such local search could improve $\mathrm{ACO}_{\mathbb{R}}$ performance in some cases, when it happens, probably the random-restart version of this direct search method could perform better. On the other hand, if $\mathrm{ACO}_{\mathbb{R}}$ performs well—as for the Ackley problem—adding a local search does not help.

4.3.4 Diversification vs. Intensification

When tackling multimodal test functions, it is important that the algorithm is able to avoid being stuck in one of the local optima. An algorithm must use some strategy to *diversify* the search in such a way that it does not get stuck in a local optimum, and yet it is able to converge in the global optimum.

These in fact are two contradictory goals. On one hand, an algorithm is expected to converge as fast as possible, while on the other hand, it is expected *not to converge* entirely to a local optimum. The fundamental problem is that an algorithm *does not know* if a given promising region contains a local or a global optimum. Hence, an algorithm has to make *an intelligent guess*, whether to focus on diversification (higher robustness), or intensification (higher convergence speed—higher efficiency).

Algorithms proposed for continuous optimization deal with this problem in various ways. Some just ignore it (like the simple (1+1)ES), but this usually does not give good results. Others explicitly divide the operation of the algorithm into the *diversification* and *intensification* phases—e.g., CGA, ECTS, or CIAC. Finally, some algorithms use one or more parameters in order to define the balance between diversification and intensification. Such an approach is used for instance by CSA-ES, CMA-ES, IDEA, and $\mathrm{ACO}_{\mathbb{R}}$.

Usually, parameters such as learning rate and population size are those that most influence the robustness of the algorithm. In the case of $\mathrm{ACO}_{\mathbb{R}}$, they also play some role—i.e., the slower the learning rate and the larger the solution archive size, the more robust is the algorithm, but the slower is the convergence speed. In $\mathrm{ACO}_{\mathbb{R}}$, there is also another parameter specifically designed to control the diversification of the search process—parameter q.

When q approaches 0, it means that only the Gaussian function associated with *the best solution found so far* is used for generating further solutions by the ants. Following Equations. B.5 and B.6, for a given parameter q and size of the solution archive k, the probability p_{qk} of choosing one of the $q \cdot k$ highest ranking solutions as the base for the PDF is: $p_{qk} \approx 0.68$ (and respectively $p_{2qk} \approx 0.95$). This is due to the characteristic of the

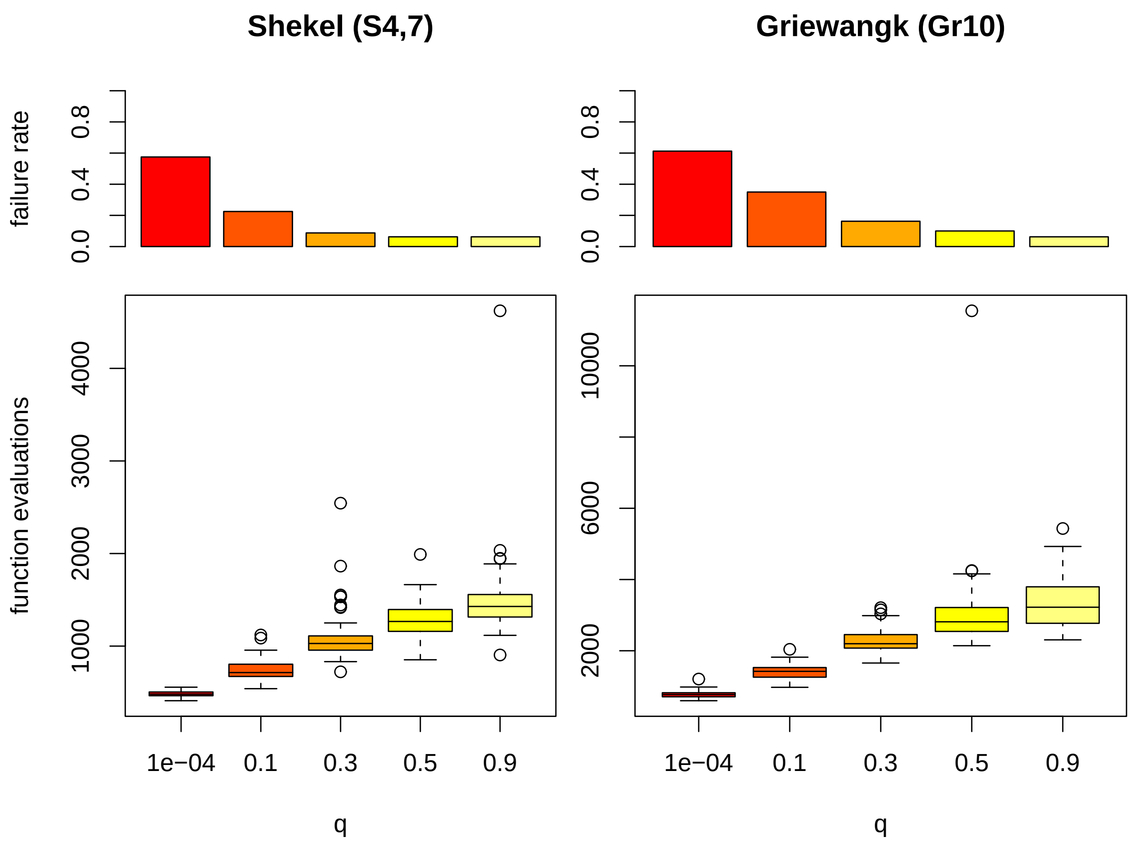


Figure 4.3: Relationship between the robustness of the ACO$_\mathbb{R}$ algorithm and its efficiency. For each of the two test functions, and for each value of the parameter q tested, the failure rate (upper part) and the distribution of the number of function evaluations needed to reach the required accuracy (lower part) are given. The size of the solution archive was fixed at $k = 100$, and $m = 2$ ants were used in all runs.

normal distribution: around 68% of the samples fall inside the interval $(-\sigma, \sigma)$ around the mean and respectively 95% in the interval $(-2\sigma, 2\sigma)$. For instance, for $q = 0.1$ and $k = 50$ (as used in experiments in Sec. 4.3.2), one of the 5 highest ranking solutions will be used with probability 0.68, and one of the 10 highest ranking solutions with probability 0.95.

When using larger q, the algorithm samples the search space based on a larger number of reasonably good solutions, rather than only on the best one found so far. The search is hence more diversified and the algorithm performs more robustly. Unfortunately, as already said, higher robustness usually means lower efficiency—slower convergence speed. This is illustrated in Fig. 4.3, which shows the failure rate (upper part) and the distribution of number of function evaluations for different values of the parameter q (lower part), for two typical multimodal test functions—Shekel ($S_{4,7}$) and Griewank (GR_{10}). The distribution of the number of function evaluations is given only for the successful runs. It is presented in the form of boxplots—the box is drawn between the first and the third quartile of the distribution, with median and outliers indicated.

4.4 Discussion

We have presented in this chapter a straightforward way of extending Ant Colony Optimization to continuous domains. We have discussed the idea and shown its implementation. ACO$_{\mathbb{R}}$ is a direct extension of ACO, and it is the first ant-based algorithm for continuous optimization which fits in the ACO framework.

We have discussed how ACO$_{\mathbb{R}}$ is situated within the (rather large) family of algorithms for continuous optimization. We have tested the performance of ACO$_{\mathbb{R}}$ against a substantial number of other algorithms and approaches. The results obtained show that ACO$_{\mathbb{R}}$ may be considered a competitive approach. Additionally, the performance of ACO$_{\mathbb{R}}$ may be adapted according to the needs to either show more robustness or higher efficiency.

ACO$_{\mathbb{R}}$, when compared to other probability-learning methods, proved to be the best on four out of 10 test problems. On the others, the quality of the solutions found was not significantly worse than the state-of-the-art. Also, ACO$_{\mathbb{R}}$ is a clear winner when compared to other ant-related algorithms for continuous optimization that were proposed in the past. When compared to these methods, ACO$_{\mathbb{R}}$ was better by almost two orders of magnitude. When compared to other metaheuristics adapted to continuous optimization, ACO$_{\mathbb{R}}$ was the winner in one-third of the test problems and performed not much worse on the others. Finally, while for some problems ACO$_{\mathbb{R}}$ is clearly outperformed by direct search methods, such as Powell, for problems with large number of local optima ACO$_{\mathbb{R}}$ performs much better than the direct search methods.

Chapter 5

ACO$_\mathbb{R}$ for Training Neural Networks

While in Chapter 4, we have presented the ACO$_\mathbb{R}$ algorithm and shown how it performs of typical continuous benchmark test problems, this chapter presents its practical application.

In this chapter, we present how the ACO$_\mathbb{R}$ algorithm may be successfully applied to the problem of training neural networks. Also, in order to provide a further insight into the performance of the ACO$_\mathbb{R}$ algorithm, we use not only a stand-alone ACO$_\mathbb{R}$, but also we introduce a hybrid versions of ACO$_\mathbb{R}$ with typical derivative-based methods used for neural network training. We compare the results obtained with those reported in the literature.

We have chosen a pattern classification problem from the medical field as a test problem. Artificial neural network (NN) is used as a classifier, but first it must be trained. We employ the ACO$_\mathbb{R}$ algorithm as the method to train this network.

Pattern classification is an important real-world problem. In the medical field, for example, pattern classification problems arise when physicians are interested in reliable classifiers for diseases based on a number of measurements. Feed-forward neural networks are commonly used systems for the task of pattern classification [Bishop, 2005], but require prior configuration. Generally, the configuration problem consists of two parts:

- First, the structure of the feed-forward NN has to be determined.
- Second, the numerical weights of the neuron connections have to be determined such that the resulting classifier is as correct as possible.

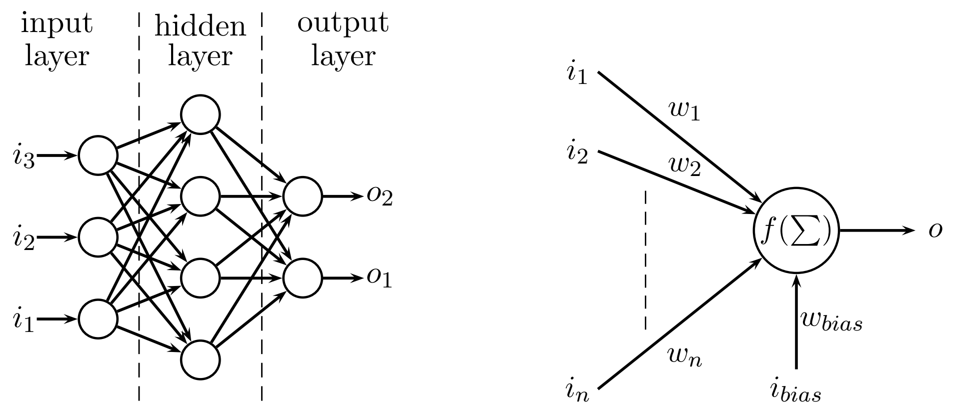


Figure 5.1: A feed-forward NN with one hidden layer of neurons is presented on the left. Note that each neuron of a certain layer is connected to each neuron of the next layer. On the right, one single neuron (from either the hidden layer, or the output layer) is presented. The neuron receives inputs (i.e., signals i_l, weighted by weights w_l) from each neuron of the previous layer. Additionally, it receives a so-called bias input i_{bias} with weight w_{bias}. The transfer function $f(\sum)$ of a neuron transforms the sum of all the weighted inputs into an output signal, which serves as input for all the neurons of the following layer. Input signals, output signals, biases and weights are real values.

In this work, we focus only on the second part, namely the optimization of the connection weights. We adopt the NN structures from earlier works on the same subject.

The outline of this chapter is as follows. In Section 5.1, we shortly present the structure of feed-forward NNs for the purpose of pattern classification. In Section 5.2, we present the experimental setup proposed. Then, in Section 5.3 we compare the ACO$_{\mathbb{R}}$ algorithm to methods specialized for feed-forward NN training, as well as to a genetic algorithm. Finally, in Section 5.4 we offer a conclusion and a glimpse of future work.

5.1 Feed-Forward Neural Networks for Pattern Classification

A dataset for pattern classification consists of a number of patterns together with their correct classification. Each pattern consists of a number of measurements (i.e., numerical values). The goal consists in generating a classifier that takes the measurements of a pattern as input, and provides its correct classification as output. A popular type of classifier are feed-forward neural networks (NNs).

A feed-forward NN consists of an input layer of neurons, an arbitrary number of hidden layers, and an output layer (for an example, see Figure 5.1). Feed-forward NNs for pattern classification purposes consist of as many input neurons as the patterns of the data set have measurements, i.e., for each measurement there exists exactly one input neuron. The output layer consists of as many neurons as the data set has classes, e.g., if the patterns of a medical data set belong to either the class normal or to the class pathological, the output layer consists of two neurons. Given the weights of all the neuron connections, in order to classify a pattern, one provides its measurements as input to the input neurons, propagates the output signals from layer to layer until the output signals of the output neurons are obtained. Each output neuron is identified with one of the possible classes. The output neuron that produces the highest output signal classifies the respective pattern (winner takes all).

5.1.1 Definition of the Problem

Due to their practical importance, we chose to evaluate the performance of ACO$_{\mathbb{R}}$ on classification problems arising in the medical field. More specifically, we chose three problems from the well-known PROBEN1 benchmark set [Prechelt, 1994], namely Cancer1, Diabetes1, and Heart1. Each of these problems consists of a number of patterns together with their correct classification, that is, Cancer1 consists of 699 patterns from a breast cancer database, Diabetes1 consists of 768 patterns concerning diabetes patients, and Heart1 is the biggest of the three data sets, consisting of 920 patterns describing a heart condition. Each pattern of the three problems is either classified as pathological, or as normal. Furthermore, each pattern consists of a number of measurements (i.e., numerical values): 9 measurements in the case of Cancer1, 8 in the case of Diabetes1, and 35 in the case of Heart1. The goal consists in generating a classifier that takes the measurements of a pattern as input, and provides its correct classification as output.

Concerning the hidden neuron layers of the feed-forward NNs that we used, we took inspiration from the literature. More specifically we used the same structure of hidden layers that were used in [Alba and Chicano, 2004]. For an overview of the feed-forward NNs that we used see Tab. 5.1. The number of weights to be optimized is (for each of the three data sets) given by the following formula:

$$n_h(n_i + 1) + n_o(n_h + 1) \ , \tag{5.1}$$

Table 5.1: Summary of the NN structures that we use for the three data sets. In the last table column is given the number of weights to be optimized for each tackled problem. Note that for the calculation of this number, the bias inputs of the neurons have also to be taken into account.

Data Set	Input Layer	Hidden Layer	Output Layer	# of weights
Cancer1	9	6	2	74
Diabetes1	8	6	2	68
Heart1	35	6	2	230

where n_i, n_h, and n_o are respectively the numbers of input, hidden, and output neurons. Note that the additional input for each neuron of the hidden layer and the output layer represents the bias inputs. The last column of Tab. 5.1 provides the number of weights to be optimized.

The process of generating a NN classifier consists of determining the weights of the connections between the neurons such that the NN classifier shows a high performance. Since the weights are real-valued, this is a continuous optimization problem of the following form: Given are n decision variables $\{X_1, \ldots, X_n\}$ with continuos domains. These domains are not restricted, i.e., each real number is feasible. Furthermore, the problem is unconstrained, which means that the variable settings do not depend on each other. Sought is a solution that minimizes the objective function called *square error percentage (SEP)*:

$$SEP = 100 \frac{o_{\max} - o_{\min}}{n_0 n_p} \sum_{p=1}^{n_p} \sum_{i=1}^{n_0} (t_i^p - o_i^p)^2 \quad , \tag{5.2}$$

where $o_{\max}$ and $o_{\min}$ are respectively the maximum and minimum values of the output signals of the output neurons, n_p represents the number of patterns, n_0 is the number of output neurons, and t_i^p and o_i^p represent respectively the expected and actual values of output neuron i for pattern p.

Finally, in order to assess the quality of the final solution found by a given algorithm, we used the Classification Error Percentage (CEP) as the performance measure. CEP represents the percentage of incorrectly classified patterns from the test set.

5.2 Experimental Setup

5.2.1 Algorithms Used for Comparison

The goal of our experimentation was to evaluate whether ACO$_{\mathbb{R}}$ may be used for training feed-forward NNs, and if so, we were interested in how it would compare to other algorithms. In order to be able to draw any meaningful conclusions, it is required to have some reference algorithm to which to compare the performance of ACO$_{\mathbb{R}}$. In order to ensure a fair comparison, we have re-implemented some algorithms traditionally used for training NNs—namely the back-propagation (BP) algorithm and the Levenberg-Marquardt (LM) algorithm. We used the R programming language (a free alternative to S+) for implementing these algorithms.

Backpropagation

Backpropagation is a gradient-descent algorithm traditionally used for training NNs [Rumelhart et al., 1986]. The term *backpropagation* is an abbreviation which stands for *backwards propagation of errors*. It is a first-order minimization algorithm—i.e., it is based on first-order derivatives (i.e., the gradient). It uses the estimation of the gradient of the instantaneous sum-squared error for each network layer:

$$\Delta\vec{w} = -\eta \bigtriangledown E(\vec{w}) \ , \tag{5.3}$$

where $\vec{w}$ is the vector of all weights, η is the learning rate, and E is the error. The technique may be informally described as follows:

1. Present a training pattern to the neural network
2. Compare the network's output to the desired output. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.

5. Assign *blame* for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat the steps above on the neurons at the previous level, using each one's *blame* as its error.

As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes. So technically speaking, backpropagation is used to calculate the first-order gradient of the error of the network with respect to the network's modifiable weights. This gradient is then used in a simple stochastic gradient descent algorithm to find weights that minimize the error. Often the term *backpropagation* is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Backpropagation usually allows quick convergence on satisfactory local minima. The BP algorithm has been rediscovered a number of times, and is a special case of a more general technique called automatic differentiation in the reverse accumulation mode.

Levenberg-Marquardt

Levenberg-Marquardt is a variation of Newton's method that was initially designed for minimizing functions that are either sums of squares, or, in general, other non-linear functions [Hagan and Menhaj, 1994]. In Newton's method, minimization is based on utilizing the second order derivatives as well as on the use of a batch training mode rather than the pattern mode (which is used, for example, in back-propagation). The batch training mode is based on derivatives of instantaneous errors. The LM algorithm uses an approximation of the Hessian matrix by adding a small constant μ multiplied by the identity matrix I to the product of the transposed Jacobian matrix J^T and the Jacobian matrix J:

$$\Delta\vec{w} = -\sum_{1}^{P} \triangledown E(w) \left[J(w)^T J(w) + \mu I\right]^{-1} \ . \tag{5.4}$$

Both algorithms (i.e., BP and LM) require gradient information. Hence, they require the neuron transfer function to be differentiable. Consequently, these algorithms may not be used in the case, when the neuron transfer function is not differentiable or is unknown. In contrast, ACO$_{\mathbb{R}}$ is a general heuristic optimization that can be applied when the neuron transfer function is non-differentiable. On the other side, in case, when the neuron transfer function is differentiable, the drawback of general optimization algorithms such as ACO$_{\mathbb{R}}$ is that they do not exploit available additional information as, for example, gradient information.

In order to see how the additional gradient information influences the performance of ACO$_{\mathbb{R}}$, we have also implemented hybridized versions of ACO$_{\mathbb{R}}$, namely ACO$_{\mathbb{R}}$-BP and ACO$_{\mathbb{R}}$-LM, which are hybrids of the ACO$_{\mathbb{R}}$ algorithm and respectively the BP and LM algorithms. In these hybrids, each solution generated by the ACO$_{\mathbb{R}}$ algorithm is improved by running a single improving iteration of either BP or LM, respectively.

Finally, we wanted to study how all the algorithms tested compare to a simple random restart search method. In order to accomplish that, we have implemented random search (RS)—i.e., an algorithm that randomly generates a set of values for the weights and then evaluates these solutions. As we used a sigmoid function as neuron transfer function, it was sufficient to limit the range of weight values to values close to 0. Hence, we arbitrarily chose a range of $[-5, 5]$.

5.2.2 Parameter Tuning

All our algorithms (with the exception of RS) require certain parameter values to be determined before they can be applied. While algorithms such as BP or LM have very few parameters, ACO$_{\mathbb{R}}$ (as well as its hybridized versions) have more. In general, in order to ensure a fair comparison of algorithms, an equal amount of effort is required in the parameter tuning process for each of the algorithms. Also, it has been shown in the literature that the stopping condition for the parameter tuning runs should be identical to the one used in the actual experiments (be that time, number of iterations, etc.), as otherwise the danger of choosing suboptimal parameter values increases [Socha, 2003]. We have hence used a common parameter tuning methodology for all our algorithms, with the same stopping condition that we planned to use for the final experiments.

The methodology that we used is known as F-RACE methodology [Birattari et al., 2002; Birattari, 2005]. In particular we used the RACE package for R. It allows running a race of different configurations of algorithms against each other on a set of test instances. After each round, the non-parametric Friedman test is used to compare the performance of different configurations. Configurations are being dropped from the race as soon as

Table 5.2: Summary of the number of patterns used for training and testing both for parameter tuning and the final performance evaluation. The parameters used for parameter tuning (learning and testing) were randomly chosen from the training set that we used later in the performance evaluation.

Problem	Total patterns	Parameter Tuning: Training set for tuning	Parameter Tuning: Test set for tuning	Performance Evaluation: Training set for testing	Performance Evaluation: Test set for testing
Cancer1	699	350	175	525	174
Diabetes1	768	384	192	576	192
Heart1	920	460	230	690	230

sufficient statistical evidence has been gathered against them. For more information on the F-RACE methodology, we refer the interested reader to [Birattari, 2005]. Since for the problems we investigated we did not have several instances available (i.e., we wanted to tune the algorithms for each of the three considered data sets separately), we have created a set of instances for each race by dividing randomly (several times) the training set of each problem instance into a training set for tuning (two thirds of the training set) and a test set for tuning (one third of the training set). Tab. 5.2 provides details on the number of patterns used respectively for learning and validation during the parameter tuning runs, as well as for training and testing the chosen configurations.

For the tuning, we determined 10 different configurations of parameter settings for each of our algorithms. Then, we applied the F-RACE to each instance set (i.e., per algorithm, per problem), allowing not more than 100 experiments in the race. Each of the parameter tuning races returned one configuration that performed best[1]. The final parameter value settings that we used for our final experiments are summarized in Tab. 5.3

5.3 Results

The evaluation of a classifier is generally performed as follows. Given a problem instance, the set of pattern is divided into a training set and a test set. The pattern from the training set are used for the training of the classifier, while the pattern from the test set

[1]Due to the limited resources for tuning, the chosen configuration for each race is not necessarily significantly better than all the others. The limit of 100 experiments per race did sometimes not allow reaching that level of assurance. However, the chosen configuration was definitely not significantly worse than any of the others.

Table 5.3: Summary of the final parameter values that we chose for our algorithms. Not included in the table are the parameters common to all $ACO_{\mathbb{R}}$ versions, namely q and m. For these parameters we used the settings $q = 0.01$, and $m = 2$ (the number of ants used in each iteration). Note that η is the step-size parameter of BP, and β is the adaptation-step parameter of LM.

	Cancer1				Diabetes1				Heart1			
Alg.	k	ξ	η	β	k	ξ	η	β	k	ξ	η	β
$ACO_{\mathbb{R}}$	148	0.95	-	-	136	0.8	-	-	230	0.6	-	-
$ACO_{\mathbb{R}}$-BP	148	0.98	0.3	-	136	0.7	0.1	-	230	0.98	0.4	-
$ACO_{\mathbb{R}}$-LM	148	0.9	-	10	136	0.1	-	10	230	0.1	-	10
BP	-	-	0.002	-	-	-	0.01	-	-	-	0.001	-
LM	-	-	-	50	-	-	-	5	-	-	-	1.5

are used for the performance evaluation of the trained classifier, that is, for calculating the CEP value. This method is called the *holdout method*. For the first set of experiments that are presented in Section 5.3, we chose the first 75% of the pattern as training set, and the remaining 25% as test set. An overview of the resulting number of patterns in training and test set is given in the last two columns of Table 5.2. We chose this method for the first set of experiments, as it made it possible to compare the results we obtained with those found in literature.

However, the holdout method may depend (sometimes heavily) on the chosen division of the pattern into training set and test set. In fact, the classifier evaluation may be significantly different depending on how this division is made. Therefore, we performed in a second set of experiments a so-called *k-fold cross-validation*. Hereby, the set of pattern is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the remaining $k - 1$ subsets are joined to form the training set. Then the average CEP value across all k trials may be computed. The aim of k-fold cross-validation is to average out the effects of the training/test set division. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation time to make an evaluation. In Section 5.3.2 we present the results of a 4-fold cross-validation.

5.3.1 Initial Results

In order to compare the performance of the algorithms, we applied each algorithm 50 times to each of the three test problems. As stopping condition we used the number of fitness function evaluations. Following the work of Alba and Chicano [Alba and Chicano, 2004], we used 1000 function evaluations as the limit in order to be able to compare our results to those obtained by them.

Figures 5.2, 5.3, and 5.4 present respectively the results obtained for the cancer, diabetes, and heart test problems in the form of box-plots. Each figure presents the distributions of the actual classification error percentage (CEP) values obtained by the algorithms (over 50 independent runs); the right one presents the distributions of rankings achieved by the algorithms. Any solution generated by any of the algorithms is ranked. Having 6 algorithms and running 50 trials each, the possible rankings vary from 1 to 300. The distribution of those rankings is then plotted per algorithm—this allows for a clear identification of those better performing ones, regardless of how small the difference may be in terms of objective function value. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95% confidence interval for a given distribution [McGill et al., 1978]. In other words, this means that if the notches of two distributions do not overlap, they are significantly different with 95

Cancer1 (see Figure 5.2) appears to be the easiest data set among the three that we tackled. All algorithms obtained reasonably good results, including the RS method. However, the best performing algorithm is BP. From the fact that the results obtained by RS do not differ significantly from the results obtained by other–more complex algorithms, it may be concluded that the problem is relatively easy, and that there are a lot of reasonably good solutions scattered over the search space. None of the algorithms was able to classify all the test patterns correctly. This may be due to the limited size of the training set, i.e. there might have been not enough information in the training set to generalize perfectly.

Diabetes1 (see Figure 5.3) is a problem that is more difficult than Cancer1. All our algorithms clearly outperform RS. However, the overall performance of the algorithms in terms of the CEP value is not very good. The best performing is again BP. The less good overall performance of the algorithms may again indicate that the training set does not represent fully all the possible patterns.

The Heart1 problem (see Figure 5.4) is—with 230 weights—the largest problem that we

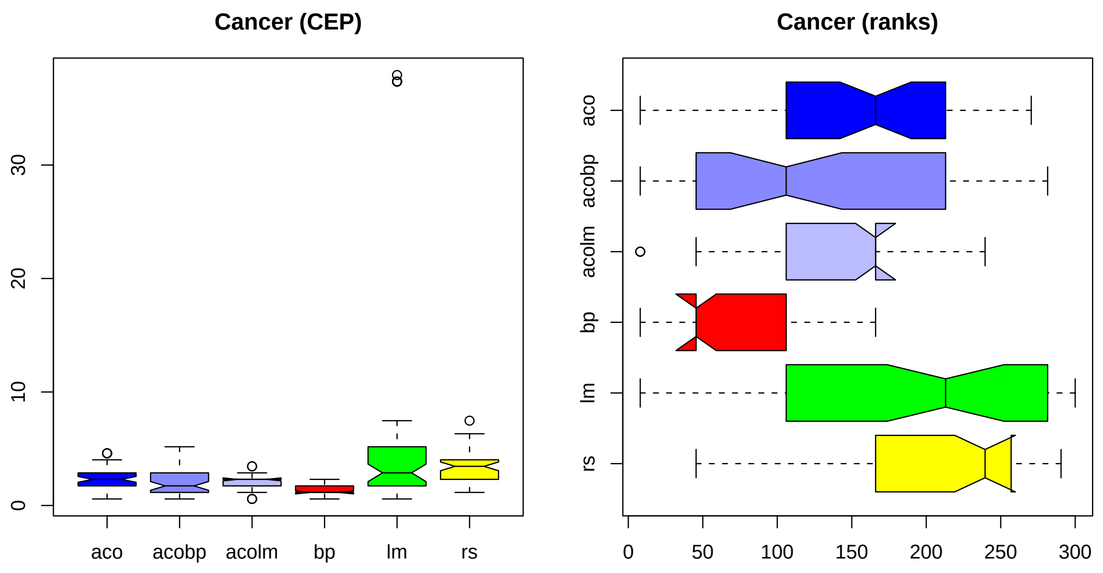


Figure 5.2: Box-plots for Cancer1. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95 % confidence interval.

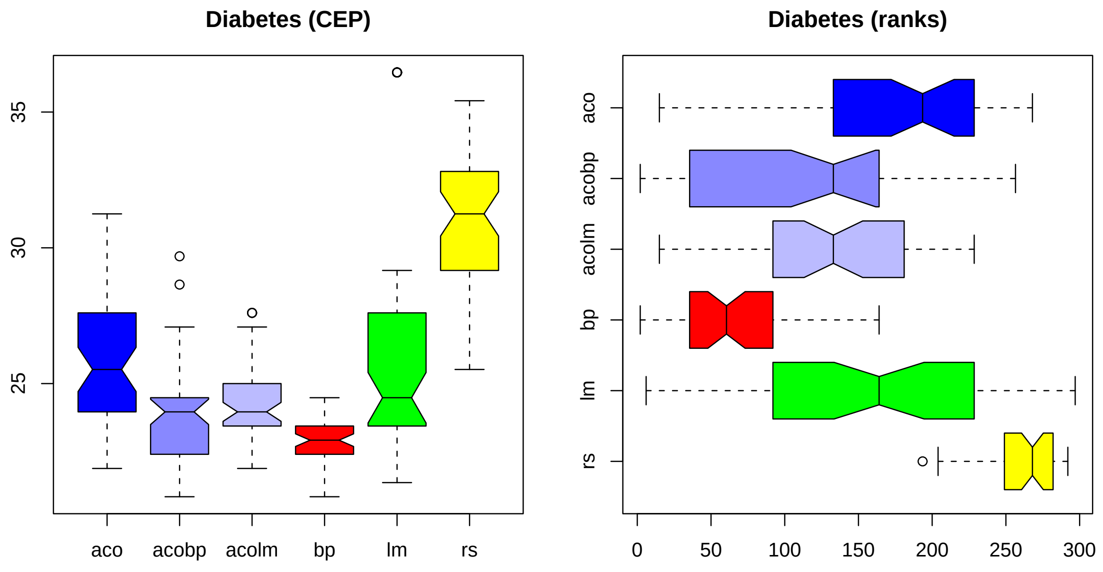


Figure 5.3: Box-plots for Diabetes1. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95 % confidence interval.

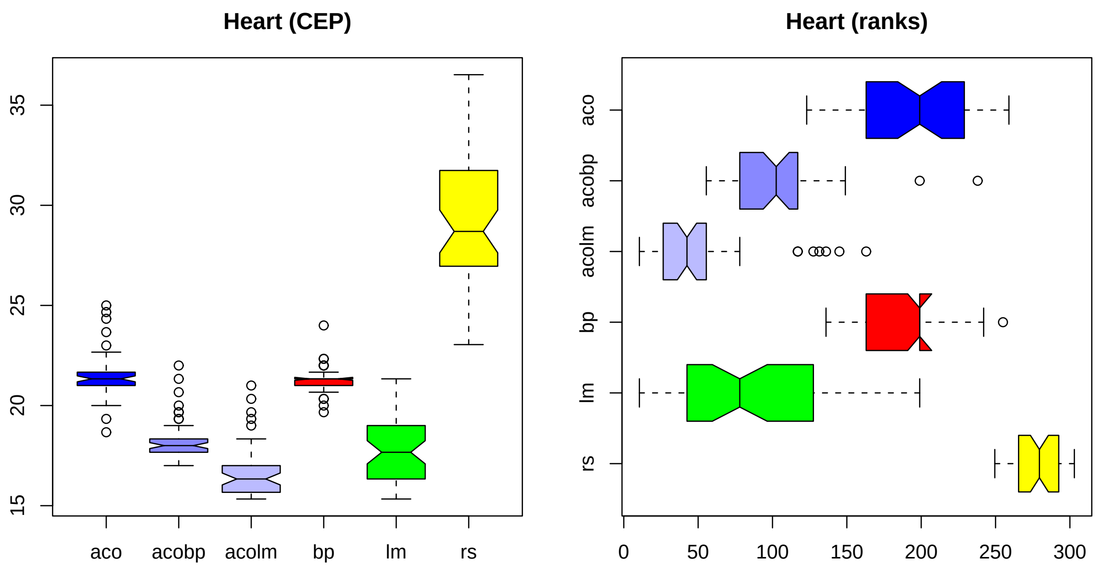


Figure 5.4: Box-plots for Heart1. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95 % confidence interval.

tackled. It is also the one on which the performance of the algorithms differed mostly. All tested algorithms clearly outperform RS, but there are also significant differences among the more complex algorithms. BP, which was performing quite well on the other two test problems, did not do so well on Heart1. ACO$_{\mathbb{R}}$ achieves results similar to BP. In turn, LM which was not performing so well on the first two problems, obtains quite good results. Very interesting is the performance of the hybridized versions of ACO$_{\mathbb{R}}$, namely ACO$_{\mathbb{R}}$-BP and ACO$_{\mathbb{R}}$-LM. The ACO$_{\mathbb{R}}$-BP hybrid clearly outperforms both ACO$_{\mathbb{R}}$ and BP. ACO$_{\mathbb{R}}$-LM outperforms respectively ACO$_{\mathbb{R}}$ and LM. Additionally, ACO$_{\mathbb{R}}$-LM performs best overall.

Summarizing, we note that the performance of ACO$_{\mathbb{R}}$ alone does often not quite reach the performance of the derivative based algorithms and the ACO$_{\mathbb{R}}$ hybrids. Its performance is, however, not much worse. Furthermore, the results show that hybridizing ACO$_{\mathbb{R}}$ with BP or LM helps to improve the results of the pure ACO$_{\mathbb{R}}$ algorithm. This was especially the case for Heart, where ACO$_{\mathbb{R}}$-LM was the overall winner. We want to remind at this point that ACO$_{\mathbb{R}}$ is much more general than for example BP and LM, because it does not require derivative information. Hence, it may be applied when the neuron transfer function of a NN is non-differentiable or unknown, while algorithms such as BP or LM could not be used in this case.

Table 5.4: Pair-wise comparison of the results (CEP) of ACO$_{\mathbb{R}}$ with recent results obtained by a set of GA (see [Alba and Chicano, 2004]). The results can be compared due to the fact that 1000 evaluations as stopping criterion were used for all the algorithms. For each problem-algorithm pair we give the mean (over 50 independent runs), and the standard deviation (in brackets). The best result of each comparison is indicated in bold.

	GA	ACO$_{\mathbb{R}}$
Cancer1	16.76 (6.15)	**2.39 (1.15)**
Diabetes1	36.46 (0.00)	**25.82 (2.59)**
Heart1	41.50 (14.68)	**21.59 (1.14)**

Table 5.5: Pair-wise comparison of the results (CEP) of the ACO$_{\mathbb{R}}$ based hybrid algorithms with recent results obtained by a set of GA based hybrid algorithms (see [Alba and Chicano, 2004]). The results can be compared due to the fact that 1000 evaluations as stopping criterion were used for all the algorithms. For each problem-algorithm pair we give the mean (over 50 independent runs), and the standard deviation (in brackets). The best result of each comparison is indicated in bold.

	GA-BP	ACO$_{\mathbb{R}}$-BP	GA-LM	ACO$_{\mathbb{R}}$-LM
Cancer1	**1.43 (4.87)**	2.14 (1.09)	**0.02 (0.11)**	2.08 (0.68)
Diabetes1	36.36 (0.00)	**23.80 (1.73)**	28.29 (1.15)	**24.26 (1.40)**
Heart1	54.30 (20.03)	**18.29 (1.00)**	22.66 (0.82)	**16.53 (1.37)**

Finally, it is interesting to compare the performance of the ACO$_{\mathbb{R}}$ based algorithms to some other general optimization algorithms. Alba and Chicano [Alba and Chicano, 2004] have published the results of a genetic algorithm (GA) used for tackling exactly the same three problems as we did. They have tested not only a stand-alone GA, but also its hybridized versions: GA-BP and GA-LM.

Tables 5.4 and 5.5 summarizes the results obtained by the ACO$_{\mathbb{R}}$ and GA based algorithms. Clearly the stand-alone ACO$_{\mathbb{R}}$ performs better than the stand-alone GA for all the test problems. ACO$_{\mathbb{R}}$-BP and ACO$_{\mathbb{R}}$-LM perform respectively better than GA-BP and GA-LM on both of the more difficult problems Diabetes1 and Heart1 and worse on Cancer1. For the Heart1 problem the mean performance of any ACO$_{\mathbb{R}}$ based algorithm is significantly better than the best GA based algorithm (which was reported as the state-of-the-art for this problem in 2004).

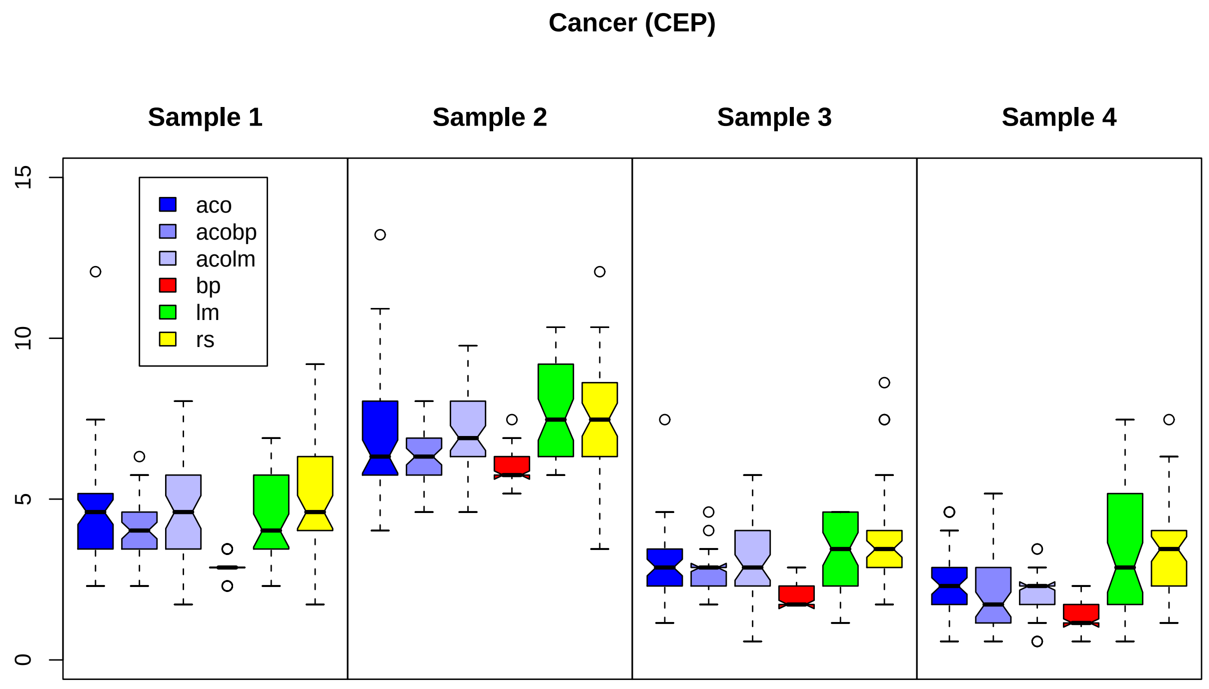


Figure 5.5: Box-plots concerning Cancer1. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95 % confidence interval. The results are presented for each sample of the 4-fold cross-validation.

5.3.2 Cross-Validation Experiments

In order to study the influence of the training/test set division, we have designed a 4-fold cross-validation. We have divided the set of pattern of each problem instance into four equal parts. Then we chose each part in turn as the test set, leaving the other three parts as training set. Each of the four pairs of training and test set is henceforth called a *sample*. Note that the 4-th sample is thus identical to the one which was used in the set of experiments described in Section 5.3.

We have not repeated the parameter tuning for the three new samples. Instead, we simply used the same set of parameter values as reported in Table 5.3. We applied each of our algorithms 50 times to each of the three new samples. Figures 5.5, 5.6, and 5.7 present the obtained results. Note that the order (from left to right) in which the algorithm results are presented is the same in each box-plot: ACO$_\mathbb{R}$, ACO$_\mathbb{R}$-BP, ACO$_\mathbb{R}$-LM, BP, LM, RS. Instead of presenting the averages over the 4 cross-validation experiments, we rather show the results for each of them separately. Several interesting observations may be made based on the performance of the algorithms on different

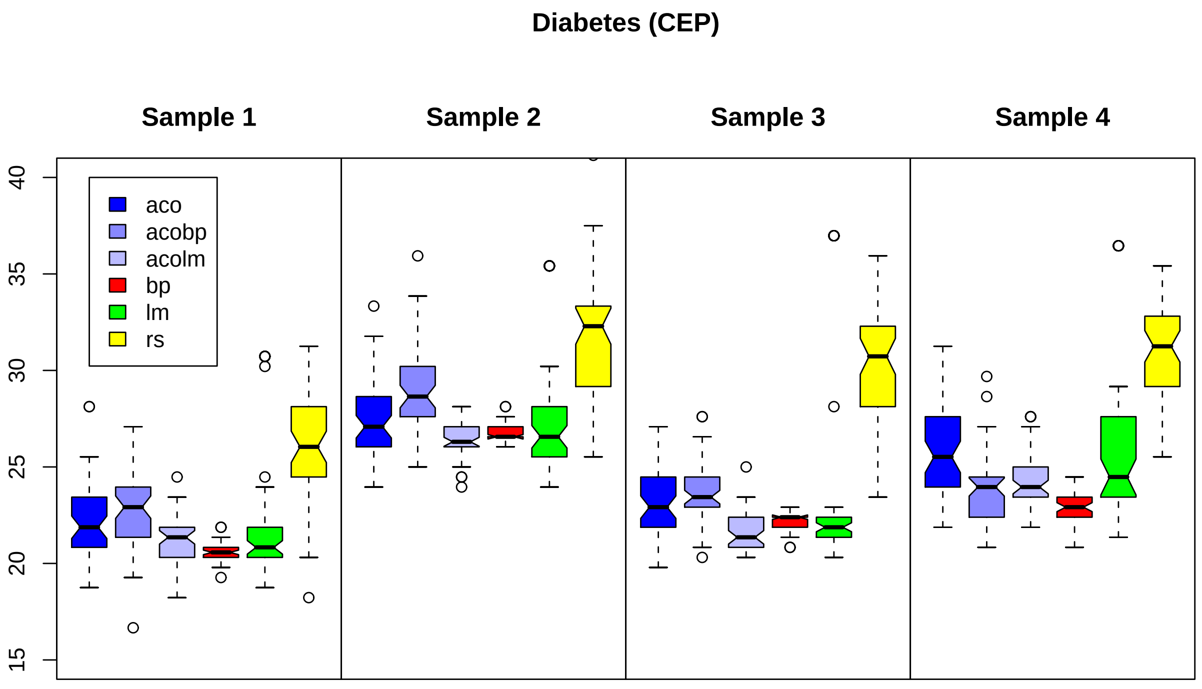


Figure 5.6: Box-plots concerning Diabetes1. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95 % confidence interval. The results are presented for each sample of the 4-fold cross-validation.

samples.

The results concerning Cancer1 allow us to make the following observation. While there are not many differences between the algorithms when applied to the same sample, the differences between the applications of the same algorithm to different samples is generally very high. The former is consistent with our observation concerning sample 4 in the previous section. However, the latter means that the difficulty levels represented by the different samples vary significantly. Apart from that, it can be observed that BP is consistently the best algorithm for all samples, although this is not always statistically significant. The fact that random search (RS) performs similarly to the other algorithms over all samples, strengthens our initial hypothesis that this problem instance is reasonably easy. The fact that there are big differences of the same algorithm when applied to different samples, might lead to the conclusion that the samples are neither homogenous nor entirely separable. If they were reasonably homogenous, the differences between the samples would be smaller. If they were separable, most likely some algorithms would perform much better than others, and in particular much better than random search. This might be caused by the fact that the number of patterns available in Cancer1 is

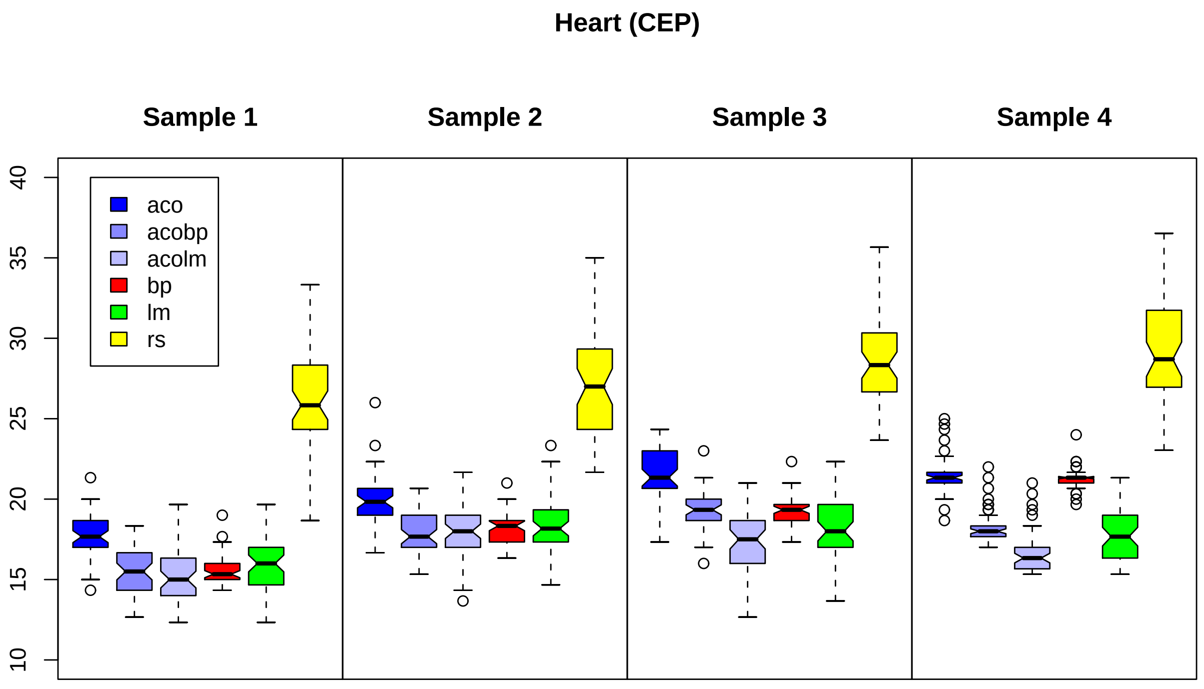


Figure 5.7: Box-plots concerning Heart1. The boxes are drawn between the first and the third quartile of the distribution, while the indentations in the box-plots (or notches) indicate the 95 % confidence interval. The results are presented for each sample of the 4-fold cross-validation.

relatively small. If the number of patterns was substantially larger, the chance of the uniform distribution of the two classes in the data set could be higher. Also, this might make the two classes better separable.

The situation for the Diabetes1 problem instance is similar to that of Cancer1. This is with the exception of random search, which performs significantly worse than all the other algorithms. Moreover, no longer a single algorithm may be identified that performs best across all the samples. While in samples 1 and 4 the best algorithm appears to be BP, in samples 2 and 3 it is rather ACO$_{\mathbb{R}}$-LM that performs best. Again, the differences of the same algorithm over the samples are quite big. This suggests a similar explanation concerning the homogeneity of the samples and number of pattern as in the case of Cancer1. Indeed, the number of pattern available for Diabetes1 is only slightly larger than for Cancer1. Finally, for all samples it holds that the separation of the different classes is far from perfect—hardly for any sample the CEPs obtained by any of the algorithms drop below 20%.

Finally, let us deal with the last problem instance tackled—the Heart1 problem. The situation in this case is slightly different than in the previous cases. First of all, over all samples the random search algorithm performs worst. Further, the performance differences between the other algorithms regarding a single sample are more pronounced, which holds for all samples. At the same time, the differences in performance of the same algorithm across the samples are no longer so pronounced—they are still there, but smaller. Moreover, $\text{ACO}_{\mathbb{R}}$-LM algorithm seems to be the best algorithm for all the samples. The higher homogeneity of the samples may be explained by the fact that the Heart1 problem instance has the highest number of patterns available in the data set: over 30% more than in the case of Cancer1. Heart1 is the most difficult problem instance of the three tackled—while the naive method of random search performs quite poorly, the remaining algorithms perform relatively well.

5.4 Discussion

We have presented an ant colony optimization algorithm (i.e., $\text{ACO}_{\mathbb{R}}$) for the training of feed-forward neural networks for pattern classification. The performance of the algorithm was evaluated on real-world test problems and compared to specialized algorithms for feed-forward neural network training (backpropagation and LevenbergMarquardt). In addition we compared our algorithms to another general optimizer, namely a genetic algorithm. Further, we have performed a 4-fold cross-validation analysis of the performance of the different algorithms. The results of this analysis show that there is probably insufficient number of training patterns in the cases of the two smaller problem instances, which leads to the fact that the classes are not very well separable. More training patterns would perhaps allow for more homogenous results. Of course, the non-separability may result from the problem definition itself. Perhaps, the chosen measurements do not provide enough information to always properly classify the case.

The performance of the stand-alone $\text{ACO}_{\mathbb{R}}$ was comparable (or at least it was not much worse) than the performance of specialized algorithms for neural network training. This result is particularly interesting as $\text{ACO}_{\mathbb{R}}$—being a much more generic approach—allows also the training of networks in which the neuron transfer function is either not differentiable or unknown (i.e., black-box type of neurons). The hybrid between $\text{ACO}_{\mathbb{R}}$ and the Levenberg-Marquardt algorithm (i.e., $\text{ACO}_{\mathbb{R}}$-LM) was in some cases able to outperform the backpropagation and the Levenberg-Marquardt algorithms that are traditionally used for neural network training. Finally, when compared to other general-purpose algorithms, namely genetic algorithm based algorithms from the literature, our results showed that the ant colony optimization based algorithms may provide superior performance for some of the test problems. Further research is needed to see how our algorithm performs on more complex problems, but the initial results are promising.

Chapter 6

Mixed-Variable Optimization with ACO—ACO_{MV}

As mentioned in the Chapter 2, mixed-variable optimization problems are a combination of discrete and continuous optimization. ACO is well known (as presented in Chapter 3) for being able to successfully tackle the discrete optimization problems. Also, as shown in Chapters 4 and 5, $\text{ACO}_{\mathbb{R}}$ may be successfully used for tackling continuous optimization problems. Hence, similarly to other continuos optimization algorithms, it may be used also for tackling mixed-variable optimization problems through relaxation of the discrete constraints.

However, this conclusion relies on the assumption that a certain *ordering* may be defined on the set of discrete variables' values. While this assumption is fulfilled in many real world problems,[1] it is not necessarily always the case. In particular, this is not true for *categorical variables*, that is, variables that may assume values associated with elements of an unordered set. We can intuitively expect that if the correct ordering is not known, or does not exist (as in case of categorical variables), the continuous optimization algorithms may perform poorly.

On the other hand, algorithms that natively handle mixed-variable optimization problems are indifferent to the ordering of the discrete variables, as they do not make any particular assumptions about it. Hence—again intuitively—we would expect that these algorithms would perform better on problems containing categorical variables, and that they would not be sensitive to the particular ordering chosen.

[1]Consider for instance pipe diameters: the sizes could be $\frac{1}{4}$", $\frac{1}{2}$", 1", $1\frac{1}{2}$", etc., and the ordering of these values is obvious.

Accordingly, in this chapter we test the hypothesis that:

Hypothesis 6.1 *Native mixed-variable optimization algorithms are more efficient than continuous optimization algorithms with relaxed discrete constraints in the case of mixed-variable problems containing* categorical *variables, for which no obvious ordering exists.*

In order to test this hypothesis, we propose a modified version of ACO$_{\mathbb{R}}$—ACO$_{\mathbf{MV}}$—an algorithm that is able to handle natively also mixed-variable optimization problems. We present the modifications made, and and give example of the implementation.

Additionally, a proper benchmark problem is needed. Mixed-variable benchmark problems may be easily found in the literature. They often originate from the mechanical engineering field. Examples include the coil spring design problem [Deb and Goyal, 1998; Lampinen and Zelinka, 1999c; Guo et al., 2004], the problem of designing a pressure vessel [Deb and Goyal, 1998; Guo et al., 2004; Schmidt and Thierauf, 2005], or the thermal insulation systems design [Audet and Dennis Jr., 2001; Kokkolaras et al., 2001]. None of these problems, however, can be easily parametrized for the purpose of comparing the performance of a continuous and a mixed-variable algorithm. Because of this, we propose a new simple yet flexible benchmark problem based on the Ellipsoid function. We use this benchmark problem for analyzing the performance of two versions our algorithm—continuous ACO$_{\mathbb{R}}$ and native mixed-variable ACO$_{\mathbf{MV}}$. Later, we evaluate the performance of both the algorithms also on other typical benchmark problems from the literature.

The remainder of this chapter is organized as follows. Section 6.1 presents the modified ACO$_{\mathbb{R}}$ that is able to handle both continuous and mixed-variable problems—ACO$_{\mathbf{MV}}$. In Section 6.2, our proposed benchmark is defined, which allows to easily compare the performance of ACO$_{\mathbb{R}}$ with that of ACO$_{\mathbf{MV}}$. The results of this comparison are presented and analyzed. In Section 6.3, ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ are tested on benchmark problems derived from real-world problems. The results obtained are also compared to those found in the literature. Finally, Section 6.4 summarizes the results obtained, offers conclusions, and outlines the plans for future work.

6.1 ACO and Mixed Variables

While ACO$_{\mathbb{R}}$ has been designed to handle only continuous variables, the ability to handle both continuous and discrete variables can be easily introduced. It requires only an additional routine responsible for constructing the discrete part of the solutions for mixed-variable optimization problems.

ACO$_{\mathbf{MV}}$ extends ACO$_{\mathbb{R}}$ allowing to declare each variable of the considered problem as continuous, ordered discrete, or categorical discrete. Continuous variables are treated as in the original ACO$_{\mathbb{R}}$, while discrete variables are treated differently. The pheromone representation (i.e., the *solution archive*) as well as the general flow of the algorithm do not change. Hence, we focus here on presenting how the discrete variables are handled.

If there are any *ordered discrete variables* defined, ACO$_{\mathbf{MV}}$ uses a continuous-relaxation approach. The natural ordering of the values for these variables may have little to do with their actual numerical values (and they may even not have numerical values, e.g., $x \in \{\text{small}, \text{big}, \text{huge}\}$). Hence, instead of operating on the actual values of the ordered discrete variables, ACO$_{\mathbf{MV}}$ operates on their indexes. The values of the indexes for the new solutions are generated by the algorithm as real numbers, as it is the case for the continuous variables. However, before the objective function is evaluated, the continuous values are rounded to the nearest valid index, and the value at that index is then used for the objective function evaluation.

Clearly, at the algorithm level, ACO$_{\mathbf{MV}}\equiv$ACO$_{\mathbb{R}}$ in this case. However, things change when the problem includes *categorical discrete variables*, as for this type of variables there is no pre-defined ordering. This means that the information about the ordering of the values in the domain may not be taken into consideration. The values for these variables need to be generated with a different method—one that is closer to the regular combinatorial ACO. We present the method used by ACO$_{\mathbf{MV}}$ in the following section.

Solution Construction for Categorical Variables

In standard ACO (see [Dorigo and Stützle, 2004]), solutions are constructed from solution components using a probabilistic rule based on the pheromone values. Differently, in ACO$_{\mathbf{MV}}$ there are no static pheromone values, but a *solution archive.* As in standard ACO, in ACO$_{\mathbf{MV}}$ the construction of solutions for discrete variables is done by choosing the components, that is, the values for each of the discrete decision variables. However,

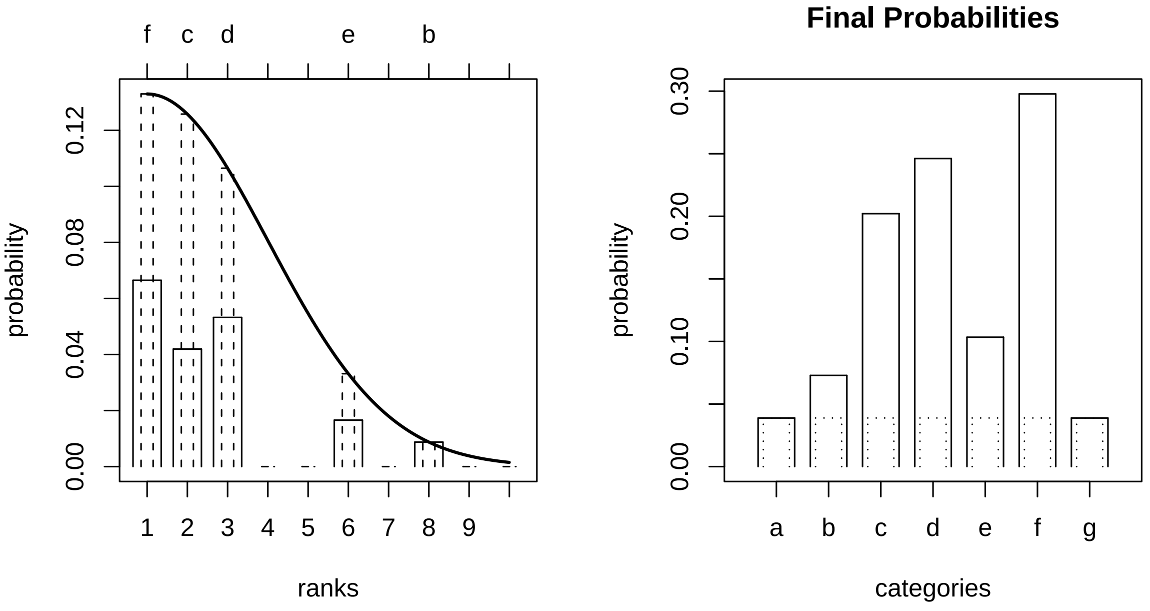


Figure 6.1: Calculating probabilities of choosing different categorical values for a given decision variable. First, the initial probabilities are generated using a normal distribution and based on the best ranked solution that uses given value (left plot, dashed bars). Then, they are divided by the number of solutions using this value (left plot, solid bars), and finally a fixed value is added (right plot, dotted bars) in order to increase the probability of choosing those values, which are currently not used. The final probabilities are presented on the right plot, as solid bars.

since the static pheromone values of standard ACO are replaced by the solution archive, the actual probabilistic rule used has to be modified.

Similarly to the case of continuous variables, each ant constructs the discrete part of the solution incrementally. For each $i = 1, ..., n$ discrete variable, each ant chooses probabilistically one of c^i available values $v_l^i \in \mathbf{D}_i = \{v_1^i, ..., v_{c^i}^i\}$. The probability of choosing the l-th value is given by:

$$o_l^i = \frac{w_l}{\sum_{r=1}^{c} w_r}, \tag{6.1}$$

where w_l is the weight associated with the l-th available value. It is calculated based on the weights ω and some additional parameters:

$$w_l = \frac{\omega_{j_l}}{u_l^i} + \frac{q}{\eta}. \tag{6.2}$$

The final weight w_l is hence a sum of two components. The weight ω_{j_l} is calculated according to Equation 4.2, where the j_l is the index of the highest quality solution that uses value v_l^i for the i-th variable. In turn, u_l^i is the number of solutions using value v_l^i for the i-th variable in the archive. Therefore, the more *popular* the value v_l^i is, the lower is its final weight.

The second component is a fixed value (i.e., it does not depend on the value v_l^i chosen): η is the number of values v_l^i from the c^i available ones that are unused by the solutions in the archive, and q is the same parameter of the algorithm that was used in Equation 4.2.

The graphical representation of how the first component $\frac{\omega_{j_l}}{u_l^i}$ is calculated is presented on the left plot of Figure 6.1. The dashed bars indicate the values of the weights ω_{j_l} obtained for the best solutions using the available values.[2] The solid bars represent the weights ω_{j_l} divided by the respective number of solutions u_l^i that use values v_l^i. It is shown for the available set of categorical values used, $v_l^i \in \{\mathrm{a, b, c, d, e, f, g}\}$ in this example.

Some of the available categorical values v_l may be unused for a given i-th decision variable in all the solutions in the archive. Hence, their initial weight is zero. In order to enhance exploration and to prevent premature convergence, in such a case, the final weights w are further modified by adding to all of them the second component. Its value depends on the parameter q and on the number of unused categorical values η_i, as shown in Equation 6.2.

The right plot in Figure 6.1 presents the normalized final probabilities for an example in which the solution archive has size $k = 10$, and where the set of categorical values is $\{\mathrm{a, b, c, d, e, f, g}\}$, with values $\{\mathrm{a}\}$ and $\{\mathrm{g}\}$ unused by the current decision variable. The dotted bars show the value of q/η added to all the solutions, and the solid bars show the final resulting probabilities associated with each of the available categories. These probabilities are then used to generate the value of the i-th decision variable for the new solutions.

[2]If a given value is not used, the associated index is indefinite, and thus its initial weight is zero.

6.2 Simple Benchmark Problem

Benchmark problems used in the literature to evaluate the performance of mixed-variable optimization algorithms are typically real-world mechanical engineering problems. They include among others: pressure vessel design, coil spring design, or thermal insulation system design problems. Although we also use these problems in Section 6.3, they are not very well suited for the detailed investigation of the performance of an algorithm. This is because their search space is not clearly defined and easy to analyze. Additionally, the objective functions of these problems cannot be manipulated easily in order to check the sensitivity of the algorithm to particular conditions. Hence, such test problems do not represent a sufficiently *controlled environment* for the investigation of the performance of an algorithm.

Such well defined test problems providing a controlled environment are often used to test the performance of continuous optimization algorithms. Although simple, they allow to compare different algorithms on particular, well defined difficulties. A good example is a set of problems used by Kern *et al.* [Kern et al., 2004].

In order to be able to flexibly compare ACO$_{\mathbb{R}}$ using the continuous relaxation approach with the native mixed-variable ACO$_{\mathbf{MV}}$ algorithm, we have designed a new mixed-variable benchmark problem. We have based it on a randomly rotated Ellipsoid function:

$$f_{EL}(\vec{x}) = \sum_{i=1}^{n} (\beta^{\frac{i-1}{n-1}} z_i)^2, \quad \begin{cases} \vec{x} \in (-3,7)^n, \\ \vec{z} = \mathbf{A}\vec{x}, \end{cases} \tag{6.3}$$

where β is the coefficient defining the scaling of each dimension of the ellipsoid, n is number of dimensions and $\mathbf{A}$ is a random normalized n-dimensional rotation matrix. In order to make it easier to analyze and visualize the results, we have limited ourselves to the two dimensional case ($n = 2$).[3]

In order to transform this continuous optimization problem into a mixed-variable one, we have divided the continuous domain of variable $x_1 \in (-3, 7)$ into a set of discrete values, $\mathbf{T} = \{\theta_1, \theta_2, ..., \theta_t\} : \theta_i \in (-3, 7)$. This results in the following mixed-variable test function:

[3]Higher dimensional problems are discussed in Section 6.3.

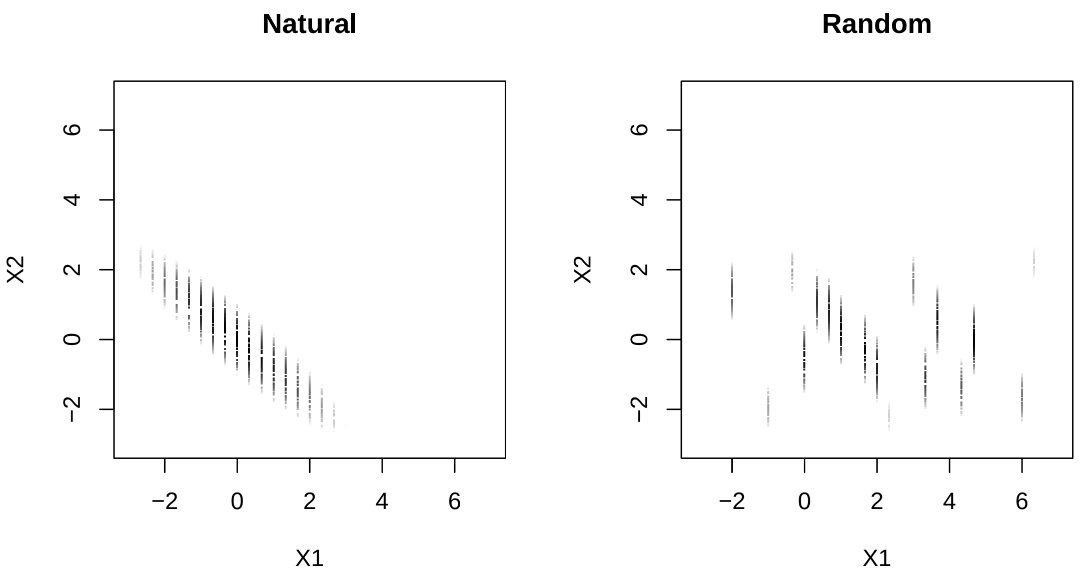


Figure 6.2: Randomly rotated ellipsoid function ($\beta = 5$) with discrete variable $x_1 \in \mathbf{T}, |\mathbf{T}| = t = 30$. The left plot presents the case in which the natural ordering of the intervals is used, while the right one presents the case in which a random ordering is used.

$$f_{EL_{MV}}(\vec{x}) = z_1^2 + \beta \cdot z_2^2, \quad \begin{cases} x_1 \in \mathbf{T}, \\ x_2 \in (-3, 7), \\ \vec{z} = \mathbf{A}\vec{x}. \end{cases} \tag{6.4}$$

The set $\mathbf{T}$ is created by choosing t uniformly spaced values from the original domain $(-3, 7)$ in such a way that $\exists_{i=1,\ldots,t}\ \theta_i = 0$. This way, it is always possible to find the optimum $f_{EL_{MV}}(0, 0) = 0$, regardless of the chosen value for t.

6.2.1 Experimental Setup

We have compared ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on two experimental setups of our test problem. We used $\beta = 100$, as this is the value most often reported in the literature for the ellipsoid function. The two setups simulate two types of mixed-variable optimization problems: (i) with ordered discrete variables, and (ii) with categorical variables.

In the first setup, the discrete intervals for variable x_1 are naturally ordered. Such a setup simulates a problem where the ordering of the discrete variables may be easily defined. The left plot in Figure 6.2 shows how the algorithm sees such a naturally ordered rotated ellipsoid function, with discrete x_1 variable.[4] The test function is presented as the ACO$_{\mathbb{R}}$ algorithm sees it—as a set of points representing different solutions found by the ants and stored in the solution archive. The darker the point, the higher the quality of the solution.

In the second setup, the intervals are ordered randomly, that is, for each run of the algorithm a different ordering was generated. This setup allows to investigate how the algorithm performs when the optimum ordering of the intervals is not well defined or unknown. The right plot of Figure 6.2 shows how the algorithm sees such modified problem for a given single random ordering. Clearly, compared to the natural ordering, the problem appears to be quite different.

For both setups, we have run both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$. We have tested the two algorithms on both test setups for a different number t of intervals for variable x_1. For fewer number of intervals, the problem is less continuous, but the probability of choosing the wrong interval is relatively small. At the same time, since the domain is always the same, the size of the intervals is larger, and ACO$_{\mathbb{R}}$ may get stuck more easily. The more intervals are used, the more the problem resembles a continuos problem, up to $t \to \infty$, when it would become a true continuos optimization problem.

6.2.2 Parameter Tuning

In order to ensure a fair comparison of the two algorithms, we have applied an identical parameter tuning procedure to both: the F-RACE method [Birattari et al., 2002; Birattari, 2005].

We have defined 168 candidate configurations of parameters for each of the algorithms. Then, we have run them on 150 instances of the test problem, which differed in terms of number of intervals used ($t \in \{10, 20, 50\}$), and of their ordering (we used always random ordering for parameter tuning). We used 50 instances for each chosen number of intervals.

The summary of the parameters chosen is given in Table 6.1.

[4]Please note that Figure 6.2 uses the value of $\beta = 5$, as it is clearer for visualization. This simply means that the ellipsoid is less flat and more circle-like.

Table 6.1: Summary of the parameters used by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$.

Parameter	Symbol	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
number of ants	m	2	2
speed of convergence	ξ	0.3	0.7
locality of the search	q	0.1	0.8
archive size	k	200	50

6.2.3 Results

For each version of the benchmark problem, we have evaluated the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ for different numbers of intervals $t \in \{2, 4, 8, 11,$
$13, 15, 16, 18, 20, 22, 32, 38, 50\}$. We have done 200 independent runs of each algorithm for each version of the benchmark problem and for each number of intervals tested.

Table 6.2 presents the means and standard deviations for both algorithms on both versions of the benchmark problem. Also, Table 6.3 presents the respective median values. However, such data is difficult to read, analyze, and draw conclusions. Hence, we also report the results separately for each version of the benchmark problem in a graphical form. This allows focusing on differences in performance of the two algorithms.

Figure 6.3 presents the performance of ACO$_{\mathbb{R}}$ (thick solid line) and ACO$_{\mathbf{MV}}$ (thinner solid line) on the benchmark problem with naturally ordered discrete intervals. The left plot presents the distributions boxplots of the results for a representative sample ($t = 8, 20, 50$) of the number of intervals used. The boxplots marked as *c.xx* were produced from results obtained with ACO$_{\mathbb{R}}$, and the boxplots marked as *m.xx* were produced from results obtained with ACO$_{\mathbf{MV}}$. The *xx* is replaced by the actual number of intervals used. Note that the y-axis is scaled to the range $[0, 1]$ for readability reasons (this causes some outliers to be not visible). The right plot presents an approximation (using smooth splines with five degrees of freedom) of the mean performance, as well as the actual mean values measured for various numbers of intervals. Additionally, we indicate the standard error of the mean (also using smooth splines).

Figure 6.4 presents in turn the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on the benchmark problem with randomly ordered intervals. It is organized similarly to Figure 6.3.

A comparison of Figures 6.3 and 6.4 reveals that, while the performance of ACO$_{\mathbf{MV}}$ does

Table 6.2: Summary of the results obtained by both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on both versions of the benchmark problem for different number of intervals. Reported are the mean and standard deviation values calculated over 200 independent runs.

	ACO$_{\mathbb{R}}$		ACO$_{\mathbf{MV}}$	
t	Nat. Order	Rnd. Order	Nat. Order	Rnd. Order
2	0.1355 (1.9164)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)
4	0.1233 (1.3465)	0.4786 (2.6095)	0.0013 (0.0116)	0.0049 (0.0334)
8	0.1124 (0.9990)	0.4133 (1.1191)	0.0177 (0.0456)	0.0206 (0.0601)
11	0.1212 (0.8242)	0.8658 (2.6419)	0.0290 (0.0804)	0.0523 (0.1762)
13	0.0989 (0.6261)	0.5537 (1.2790)	0.0467 (0.1594)	0.0274 (0.0838)
15	0.0144 (0.2042)	0.6656 (1.4998)	0.0400 (0.1120)	0.0384 (0.0939)
16	0.0192 (0.1965)	0.5929 (1.3251)	0.0350 (0.0928)	0.0264 (0.0619)
18	0.0927 (0.6050)	0.6230 (1.5268)	0.0436 (0.1173)	0.0389 (0.1089)
20	0.0287 (0.2902)	0.6917 (2.5739)	0.0361 (0.0890)	0.0324 (0.0933)
22	0.0050 (0.0709)	0.5526 (1.4248)	0.0279 (0.0677)	0.0317 (0.0837)
32	0.0053 (0.0537)	0.6055 (1.4865)	0.0263 (0.0560)	0.0674 (0.3180)
38	0.0049 (0.0699)	0.5321 (1.5796)	0.0352 (0.1176)	0.0555 (0.2378)
50	0.0008 (0.0115)	0.5163 (1.6856)	0.0659 (0.4775)	0.1161 (0.4398)

not change significantly, ACO$_{\mathbb{R}}$ performs much better in the case of natural ordering of the intervals. In this case, the mean performance of ACO$_{\mathbb{R}}$ is inferior to the one of ACO$_{\mathbf{MV}}$ when $t < 18$. As the number of intervals used increases, the performance of ACO$_{\mathbb{R}}$ improves, and eventually (for $t > 20$) it becomes better than ACO$_{\mathbf{MV}}$.

Clearly, the performance of ACO$_{\mathbb{R}}$ is different for a small or for a large number of intervals. For a small number of intervals, the mean performance of ACO$_{\mathbb{R}}$ is influenced by the large size of the intervals. If the algorithm gets stuck in the wrong interval, the best value it can find there is much worse than the optimum one. Hence, although this happens quite rarely, this causes worse mean performance. The more intervals are used, the closer the problem resembles a continuous one. The size of intervals becomes smaller, and ACO$_{\mathbb{R}}$ is less likely to get stuck in a wrong one. Hence, for larger number of intervals used, ACO$_{\mathbb{R}}$'s performance improves.

The mean performance of ACO$_{\mathbb{R}}$ on the version of the benchmark problem with randomly ordered intervals follows a similar pattern, but is generally worse. For a small number of intervals, the mean performance is penalized by their large size. With increasing number

Table 6.3: Median values (over 200 independent runs) obtained by both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on both versions of the test problem for different number of intervals.

	ACO$_{\mathbb{R}}$		ACO$_{\mathbf{MV}}$	
t	Nat. Order	Rnd. Order	Nat. Order	Rnd. Order
2	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.0000
8	0.0000	0.0000	0.0003	0.0005
11	0.0000	0.0000	0.0016	0.0033
13	0.0000	0.0000	0.0021	0.0023
15	0.0000	0.0000	0.0034	0.0034
16	0.0000	0.0000	0.0035	0.0032
18	0.0000	0.0000	0.0043	0.0025
20	0.0000	0.0000	0.0053	0.0015
22	0.0000	0.0000	0.0021	0.0022
32	0.0000	0.1112	0.0026	0.0023
38	0.0000	0.0694	0.0024	0.0019
50	0.0000	0.0541	0.0019	0.0031

of intervals, the performance further degrades due to the lack of natural ordering—the search space contains discrete traps, with which ACO$_{\mathbb{R}}$ does not deal very well. For $t > 20$, the mean performance begins to improve again, as there is a higher probability of finding a discrete interval reasonably close in quality to the optimal one. It is important to notice that, while in the case of natural ordering the median performance is systematically very good for any number of intervals, in the case of random ordering the median performance of ACO$_{\mathbb{R}}$ decreases with the increase of the number of intervals. Also, the mean performance of ACO$_{\mathbb{R}}$ remains inferior to ACO$_{\mathbf{MV}}$ for any number of intervals tested.

Contrary to ACO$_{\mathbb{R}}$, the mean performance of ACO$_{\mathbf{MV}}$ does not differ for the two versions of the benchmark problem. This is consistent with the initial hypothesis, since the ordering of the intervals should not matter for a native mixed-variable algorithm. ACO$_{\mathbf{MV}}$ is always able to find an interval that is reasonably close in quality to the optimal one. Its efficiency depends only on the number of intervals available. The more there are intervals, the more difficult it becomes to find the optimal one. This is why the mean performance of ACO$_{\mathbf{MV}}$ decreases with the increase of the number of intervals.

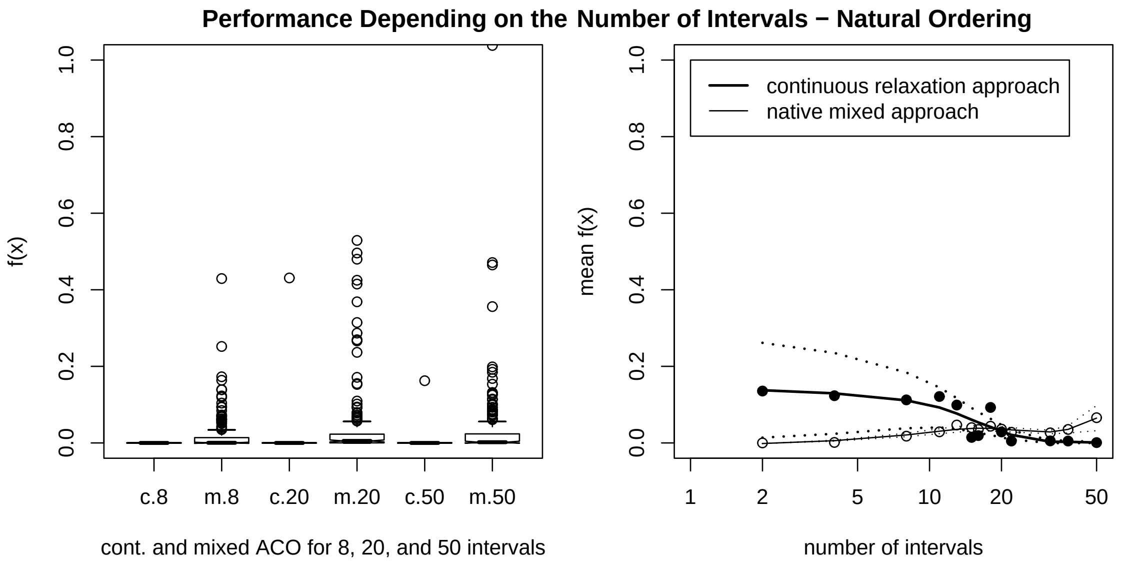


Figure 6.3: Performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on a discretized randomly rotated Ellipsoid function ($n = 2$) with *natural* ordering of the intervals. The results distribution of ACO$_{\mathbb{R}}$ (c) and ACO$_{\mathbf{MV}}$ (m) for $t = \{8, 20, 50\}$ are on the left plot. The right plot shows the mean performance for different number of intervals. The dotted lines indicate the standard error of the mean, and the circles are the measured means for different numbers of intervals.

6.2.4 Discussion

Our initial hypothesis that the ordering of the discrete intervals should not have impact on the performance of the native mixed-variable optimization algorithm appears to hold. The mean performance of ACO$_{\mathbf{MV}}$ is better than ACO$_{\mathbb{R}}$, when the ordering is not natural. These results suggest that ACO$_{\mathbf{MV}}$ should perform well also on other real-world and benchmark problems containing *categorical* variables.

In order to further asses the performance of ACO$_{\mathbf{MV}}$ on various mixed-variable optimization problems, we tackle them using both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ in the next section. We also compare the results obtained with those reported in the literature, so that the results obtained by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ may be put in perspective.

6.3 Results on Various Benchmark Problems

In many industrial processes and problems some parameters are discrete and other continuous. This is why problems from the area of mechanical design are often used as

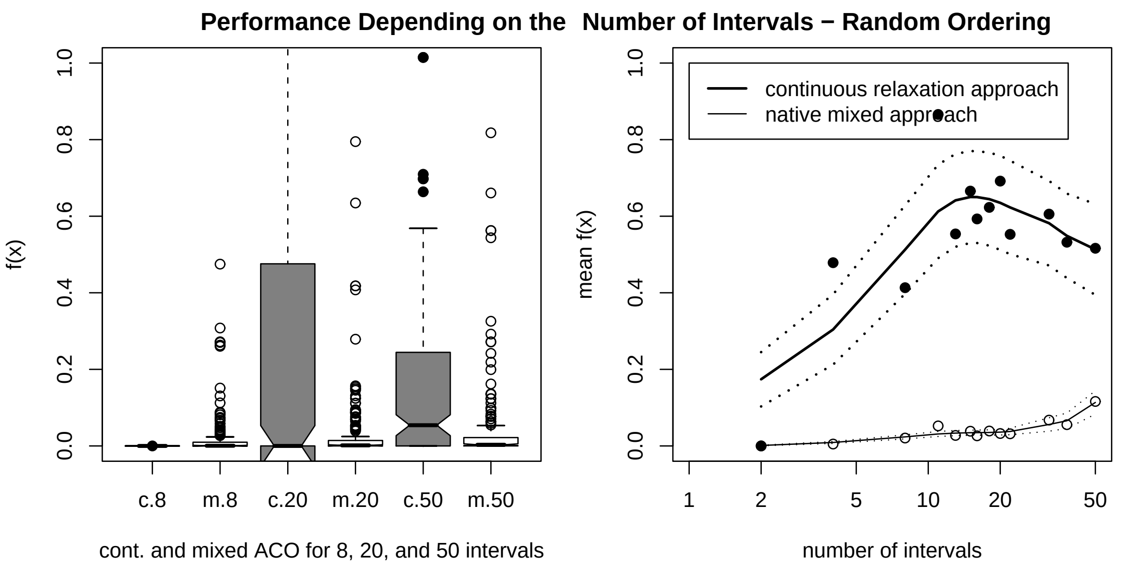


Figure 6.4: Performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on a discretized randomly rotated Ellipsoid function ($n = 2$) with *random* ordering of the intervals. The results distribution of ACO$_{\mathbb{R}}$ (c) and ACO$_{\mathbf{MV}}$ (m) for $t = \{8, 20, 50\}$ are on the left plot. The right plot shows the mean performance for different number of intervals. The dotted lines indicate the standard error of the mean, and the circles the measured means for different numbers of intervals.

benchmarks for mixed-variable optimization algorithms. Popular examples include truss design [Sellar et al., 1994; Turkkan, 2003; Pandia Raj and Kalyanaraman, 2005; Schmidt and Thierauf, 2005], coil spring design [Deb and Goyal, 1998; Lampinen and Zelinka, 1999c; Guo et al., 2004], pressure vessel design [Deb and Goyal, 1998; Guo et al., 2004; Schmidt and Thierauf, 2005], welded beam design [Deb and Goyal, 1998], and thermal insulation systems design [Audet and Dennis Jr., 2001; Kokkolaras et al., 2001].

In order to illustrate the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on real-world mixed-variable problems, we use a subset of these problems. In particular, in the following sections, we present the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on the pressure vessel design problem, the coil spring design problem, and the thermal insulation system design problem. Also, we compare the results obtained with those reported in the literature.

6.3.1 Pressure Vessel Design Problem

The first engineering benchmark problem that we tackle is the problem of designing a pressure vessel. The pressure vessel design (PVD) problem has been used numerous

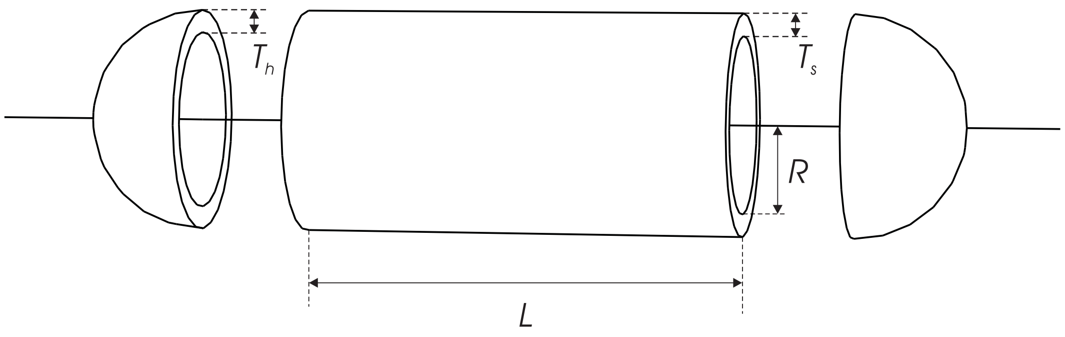


Figure 6.5: Schematic of the pressure vessel to be designed.

times as a benchmark for mixed-variable optimization algorithms [Sandgren, 1990; Deb and Goyal, 1998; Lampinen and Zelinka, 1999a; Guo et al., 2004; Schmidt and Thierauf, 2005].

Problem Definition

The problem requires designing a pressure vessel consisting of a cylindrical body and two hemispherical heads such that the cost of its manufacturing is minimized subject to certain constraints. The schematic picture of the vessel is presented in Figure 6.5. There are four variables for which values must be chosen: the thickness of the main cylinder T_s, the thickness of the heads T_h, the inner radius of the main cylinder R, and the length of the main cylinder L. While variables R and L are continuous, the thickness for variables T_s and T_h may be chosen only from a set of allowed values, these being the integer multiplies of 0.0625 inch.

There are numerous constraints for the choice of the values of the variables. In fact, there are three distinctive cases (A, B, and C) defined in the literature. These cases differ by the constraints posed on the thickness of the steel used for the heads and the main cylinder. Since each case results in a different optimal solution, we present here all three of them. Table 6.4 presents the required constrains for all three cases.

The objective function represents the manufacturing cost of the pressure vessel. It is a combination of material cost, welding cost, and forming cost. Using rolled steel plate, the main cylinder is to be made in two halves that are joined by two longitudinal welds. Each head is forged and then welded to the main cylinder. The objective function is the following:

Table 6.4: Constraints for the three cases (A, B, and C) of the pressure vessel design problem.

No	Case A	Case B	Case C
g_1	$T_s - 0.0193\,R \geq 0$		
g_2	$T_h - 0.00954\,R \geq 0$		
g_3	$\pi\,R^2L + \frac{4}{3}\pi\,R^3 - 750 \cdot 1728 \geq 0$		
g_4	$240 - L \geq 0$		
g_5	$1.1 \leq T_s \leq 12.5$	$1.125 \leq T_s \leq 12.5$	$1 \leq T_s \leq 12.5$
g_6	$0.6 \leq T_h \leq 12.5$	$0.625 \leq T_h \leq 12.5$	
g_7	$0.0 \leq R \leq 240.0$		
g_8	$0.0 \leq L \leq 240.0$		

$$f(T_s, T_h, R, L) = 0.6224\,T_sRL + 1.7781\,T_hR^2 + 3.1611\,T_s^2L + 19.84\,T_s^2R. \tag{6.5}$$

The coefficients used in the objective function, as well as the constraints, come from conversion of units from imperial to metric ones. The original problem defined the requirements in terms of imperial units, that is, the working pressure of 3000 psi and the minimum volume of 750 ft^3. For more details on the initial project formulation, as well as on how the manufacturing cost of the pressure vessel is calculated, we refer the interested reader to [Sandgren, 1990].

Experimental Setup

Most benchmarks found in the literature set to 10, 000 the number of function evaluations on which the algorithms are evaluated. Accordingly, we use 10, 000 function evaluations as stopping criterion for both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$.

The constraints defined in the PVD problem were handheld in a rather simple manner. The objective function was defined in such a way that if any of the constraints was violated, the function returns an infinite value. In this way feasible solutions are always better than infeasible ones.

For each of the cases A, B, and C, we have performed 100 independent runs for ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ in order to asses the robustness of the performance.

Parameter Tuning

In order to ensure a fair comparison of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$, we have used the F-RACE method for tuning the parameters. The parameters selected by F-RACE are listed in Table 6.5. Note that the same parameters were selected for all the three cases of the PVD problem.

Table 6.5: Summary of the parameters chosen for ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ for the PVD problem.

Parameter	Symbol	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
number of ants	m	2	2
speed of convergence	ξ	0.9	0.8
locality of the search	q	0.05	0.3
archive size	k	50	50

Results

The pressure vessel design problem has been tackled by numerous algorithms in the past. The results of the following algorithms are available in the literature:

- nonlinear integer and discrete programming (NLIDP) [Sandgren, 1990] (cases A and B),
- mixed integer-discrete-continuous programming (MIDCP) [Fu et al., 1991] (cases A and B),
- sequential linearization approach (SLA) [Loh and Papalambros, 1991] (case B),
- nonlinear mixed discrete programing (NLMDP) [Li and Chou, 1994] (case C),
- genetic algorithm (GA) [Wu and Chow, 1995] (case B),
- evolutionary programming (EP) [Cao and Wu, 1997] (case C),
- evolution strategy (ES) [Thierauf and Cai, 2000] (case C),
- differential evolution (DE) [Lampinen and Zelinka, 1999a] (cases A, B, and C),
- combined heuristic optimization approach (CHOPA) [Schmidt and Thierauf, 2005] (case C),

- hybrid swarm intelligence approach (HSIA) [Guo et al., 2004] (case B).

For the sake of completeness, we have run our algorithms on all the three cases of the PVD problem. Tables 6.6, 6.7, and 6.8 summarize the results found in the literature and those obtained by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$. Each table provides the best value found (rounded to three digits after the decimal point), the success rate—that is, the percentage of the runs, in which at least the reported best value was found, and the number of function evaluations allowed. We have performed 100 independent runs for both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$.

Table 6.6: Results for Case A of the pressure vessel design problem. For each algorithm are given the best value, the success rate (i.e., how often the best value was reached), and the number of function evaluations allowed. Note that, in some cases, the number of evaluations allowed was not indicated in the literature. Also, for the ACO algorithms, the mean number of evaluations of the successful runs is given in parentheses.

	NLIDP	MIDCP	DE	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
f^*	7867.0	7790.588	7019.031	7019.031	7019.031
success rate	100%	99%	89.2%	100%	28%
# of function eval.	-	-	10000	10000	10000
				(3037)	(6935)

Table 6.7: Results for Case B of the pressure vessel design problem. For each algorithm are given the best value, the success rate (i.e., how often the best value was reached), and the number of function evaluations allowed. Note that, in some cases, the number of evaluations allowed was not indicated in the literature. Also, for the ACO algorithms, the mean number of evaluations of the successful runs is given in parentheses.

	NLIDP	SLA	GA	DE	HSIA	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
f^*	7982.5	7197.734	7207.497	7197.729	7197.9	7197.729	7197.729
success rate	100%	90.2%	90.3%	90.2%	-	100%	35%
# of function eval.	-	-	-	10000	-	10000	10000
						(3124)	(6998)

Based on the results obtained, it may be concluded that both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ are able to find the best currently known value for all three cases of the PVD problem. Additionally, ACO$_{\mathbb{R}}$ is able to do so in just over 3,000 objective function evaluations on average, while maintaining 100% success rate. On the other hand, ACO$_{\mathbf{MV}}$, although

Table 6.8: Results for Case C of the pressure vessel design problem. For each algorithm are given the best value, the success rate (i.e., how often the best value was reached), and the number of function evaluations allowed. Note that, in some cases, the number of evaluations allowed was not indicated in the literature. Also, for the ACO algorithms, the mean number of evaluations of the successful runs is given in parentheses.

	NLMDP	EP	ES	DE	CHOPA	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
f^*	7127.3	7108.616	7006.9	7006.358	7006.51	7006.358	7006.358
success rate	100%	99.7%	98.3%	98.3%	-	100%	14%
# of function eval.	-	-	4800	10000	10000	10000 (3140)	10000 (6927)

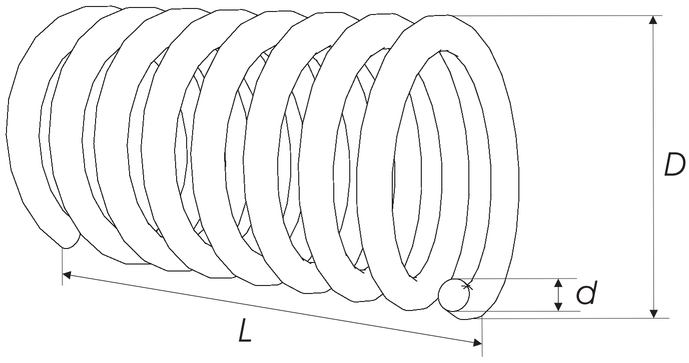


Figure 6.6: Schematic of the coil spring to be designed.

also able to find the best known solution for all the three cases of the PVD problem, it does so only in about 30% of the runs within the 10,000 evaluations of the objective function. Hence, on the PVD problem ACO$_{\mathbf{MV}}$ performs significantly worse than ACO$_{\mathbb{R}}$, which is consistent with the initial hypothesis that ACO$_{\mathbb{R}}$ performs better than ACO$_{\mathbf{MV}}$ when the optimization problem includes no categorical variables.

6.3.2 Coil Spring Design Problem

The second benchmark problem that we considered is the coil spring design (CSD) problem [Sandgren, 1990; Deb and Goyal, 1998; Lampinen and Zelinka, 1999c; Guo et al., 2004]. This is another popular benchmark used for comparing mixed-variable optimization algorithms.

Problem Definition

The problem consists in designing a helical compression spring that will hold an axial and constant load. The objective is to minimize the volume of the spring wire used to manufacture the spring. A schematic of the coil spring to be designed is shown in Figure 6.6. The decision variables are the number of spring coils N, the outside diameter of the spring D, and the spring wire diameter d. The number of coils N is an integer variable, the outside diameter of the spring D is a continuous one, and finally, the spring wire diameter is a discrete variable, whose possible values are given in Table 6.9.

Table 6.9: Standard wire diameters available for the spring coil.

Allowed wire diameters [inch]					
0.0090	0.0095	0.0104	0.0118	0.0128	0.0132
0.0140	0.0150	0.0162	0.0173	0.0180	0.0200
0.0230	0.0250	0.0280	0.0320	0.0350	0.0410
0.0470	0.0540	0.0630	0.0720	0.0800	0.0920
0.1050	0.1200	0.1350	0.1480	0.1620	0.1770
0.1920	0.2070	0.2250	0.2440	0.2630	0.2830
0.3070	0.3310	0.3620	0.3940	0.4375	0.5000

The original problem definition [Sandgren, 1990] used imperial units. In order to have comparable results, all subsequent studies that used this problem [Deb and Goyal, 1998; Lampinen and Zelinka, 1999c; Guo et al., 2004] continued to use the imperial units; so did we.

The spring to be designed is subject to a number of design constraints, which are defined as follows:

- The maximum working load, $F_{\max} = 1000.0\,\text{lb}$.
- The allowable maximum shear stress, $S = 189000.0\,\text{psi}$.
- The maximum free length, $l_{\max} = 14.0\,\text{in}$.
- The minimum wire diameter, $d_{\min} = 0.2\,\text{in}$.
- The maximum outside diameter of the spring, $D_{\max} = 3.0\,\text{in}$.
- The pre-load compression force, $F_p = 300.0\,\text{lb}$.

- The allowable maximum deflection under pre-load, $\sigma_{pm} = 6.0\,\text{in}$.
- The deflection from pre-load position to maximum load position, $\sigma_w = 1.25\,\text{in}$.
- The combined deflections must be consistent with the length, that is, the spring coils should not touch each other under the maximum load at which the maximum spring deflection occurs.
- The shear modulus of the material, $G = 11.5 \cdot 10^6$.
- The spring is guided, so the buckling constraint is bypassed.
- The outside diameter of the spring, D, should be at least three times greater than the wire diameter, d, to avoid lightly wound coils.

These design constraints may be formulated into a set of explicit constraints, listed in Table 6.10. The following symbols are used in the constraints definition:

$$\begin{aligned} C_f &= \frac{4\frac{D}{d}-1}{4\frac{D}{d}-4} + \frac{0.615\,d}{D} \\ K &= \frac{Gd^4}{8\,ND^3} \\ \sigma_p &= \frac{F_p}{K} \\ l_f &= \frac{F_{\max}}{K} + 1.05(N+2)d \end{aligned} \tag{6.6}$$

Table 6.10: Constraints for the coil spring design problem.

No	Constraint
g_1	$\frac{8\,C_f F_{\max} D}{\pi\,d} - S \leq 0$
g_2	$l_f - l_{\max} \leq 0$
g_3	$d_{\min} - d \leq 0$
g_4	$D - D_{\max} \leq 0$
g_5	$3.0 - \frac{D}{d} \leq 0$
g_6	$\sigma_p - \sigma_{pm} \leq 0$
g_7	$\sigma_p + \frac{F_{\max}-F_p}{K} + 1.05(N+2)d - l_f \leq 0$
g_8	$\sigma_w - \frac{F_{\max}-F_p}{K} \leq 0$

The cost function to be minimized computes the volume of the steel wire as a function of the design variables:

$$f_c(N, D, d) = \frac{\pi\, D d^2 (N+2)}{4} \tag{6.7}$$

Experimental Setup

Most of the research on the CSD problem reported in the literature focused on finding the best solution. Only the recent work by Lampinen and Zelinka [Lampinen and Zelinka, 1999a] gave some attention to the number of function evaluations used to reach the best solution. They used 8,000 function evaluations. In order to obtain results that could be compared, this was used also used for both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$.

The constraints defined in the CSD problem were handled with the use of a penalty function, similarly to the way it was done by Lampinen and Zelinka [Lampinen and Zelinka, 1999a]. The objective function was defined as follows:

$$f = f_c \prod_{i=1}^{8} c_i^3, \tag{6.8}$$

where:

$$c_i = \begin{cases} 1 + s_i g_i & \text{if } g_i > 0, \\ 1 & \text{otherwise,} \end{cases} \tag{6.9}$$
$$s_1 = 10^{-5},\ s_2 = s_4 = s_6 = 1,\ s_3 = s_5 = s_7 = s_8 = 10^2.$$

We have performed 100 independent runs for both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ in order to asses the robustness of the algorithms' performance.

Parameter Tuning

We have used the F-RACE method for tuning the parameters. The parameters chosen this way for both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ are summarized in Table 6.11.

Table 6.11: Summary of the parameters chosen for ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ for the CSD problem.

Parameter	Symbol	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
number of ants	m	2	2
speed of convergence	ξ	0.8	0.2
locality of the search	q	0.06	0.2
archive size	k	120	120

Results

In the CSD problem the discrete variables can be easily ordered. Therefore, similarly to the PVD problem, we expect that ACO$_{\mathbb{R}}$ will perform better than ACO$_{\mathbf{MV}}$. Table 6.12 presents the results found by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$, as well as those found in the literature.

Table 6.12: Results for the coil spring design problem. For each algorithm are given the best value, the success rate (i.e., how often the best value was reached), and the number of function evaluations allowed. Note that, in some cases, the number of evaluations allowed was not indicated in the literature.

	NLIDP	GA	DE	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
f^*	2.7995	2.6681	2.65856	2.65856	2.65856
success rate	100%	95.3%	95.0%	82%	39%
# of function eval.	-	-	8000	8000	8000

Both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ were able to find the current best known solution of the CSD problem. ACO$_{\mathbb{R}}$ again performed better than ACO$_{\mathbf{MV}}$. Both were performing better (in terms of quality of the solutions found) than many methods reported in literature. ACO$_{\mathbb{R}}$ was a bit less robust (lower success rate) than the differential evolution (DE) used by Lampinen and Zelinka [Lampinen and Zelinka, 1999a].

The results obtained by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ for the coil spring design problem are consistent with the findings for pressure vessel design. The variables of the coil spring design problem may be easily ordered and ACO$_{\mathbb{R}}$ performs better than ACO$_{\mathbf{MV}}$, as expected. ACO$_{\mathbb{R}}$ was run with the same number of function evaluations as the DE used by Lampinen and Zelinka. The best result is the same, but while in the PVD problem ACO$_{\mathbb{R}}$ had a higher success rate than DE [Lampinen and Zelinka, 1999a] (as well as a faster convergence), in the case of the CSD problem, the success rate is slightly lower than that of DE.

Generally, again—the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ is comparable to the best results reported in the literature. ACO$_{\mathbb{R}}$ is able to find the known optimum in a comparable amount of time. As expected, ACO$_{\mathbb{R}}$ is performing better than ACO$_{\mathbf{MV}}$ for problems, where the discrete variables may be easily ordered.

6.3.3 Designing Thermal Insulation System

The third engineering problem that we used to test our algorithms was the thermal insulation system design (TISD) problem. The choice was due to the fact that this is one of the few benchmark problems used in the literature that deals with *categorical* variables—that is, variables which have no natural ordering.

Various types of thermal insulation systems have been tackled and discussed in the literature. Hilal and Boom [Hilal and Boom, 1977] considered cryogenic engineering applications in which mechanical struts are necessary in the design of solenoids for superconducting magnetic energy storage systems. In this case, vacuum is ruled out as an insulator because the presence of a material is always necessary between the hot and the cold surfaces in order to support mechanical loads. Hilal and Boom used only few intercepts in their studies (up to three), and they considered only one single material for all the layers between the intercepts.

More recently, cryogenic systems of space borne magnets have been studied. The insulation efficiency of a space borne system ensures that the available liquid helium used for cooling the intercepts evaporates with minimum rate during the mission. Some studies [Musicki et al., 1989] focused on optimizing the inlet temperatures and flow rates of the liquid helium for a predefined number of intercepts and insulator layers. Others [Yamaguchi et al., 1991] studied the effect of the number of intercepts and the types of insulator on the temperature distribution and insulation efficiency. Yet others [Li et al., 1989] considered different substances such as liquid nitrogen or neon for cooling the intercepts and compared different types of insulators.

In all the studies mentioned so far, the categorical variables describing the type of insulators used in different layers were not considered as optimization variables, but rather as parameters. This is due to the fundamental property of categorical variables—there is no particular ordering defined on the set of available materials, and hence they may not be *relaxed* to be handheld as regular continuous optimization variables. The

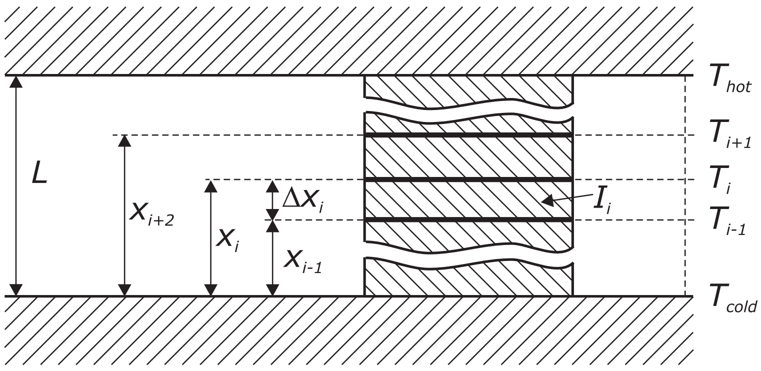


Figure 6.7: Schematic of the thermal insulation system.

algorithms used by the before mentioned studies did not allow to handle such categorical variables. Only the more recent work of Kokkolaras *et al.* [Kokkolaras et al., 2001] propose a mixed-variable programming (MVP) algorithm, which is able to handle such categorical variables properly.

In this section, we show that, thanks to the fact that ACO$_{\mathbf{MV}}$ can also handle natively categorical variables, it performs comparably to MVP on the thermal insulation system design problem, and outperforms significantly ACO$_{\mathbb{R}}$, which further confirms our initial Hypothesis 6.1.

Problem Definition

In our work, we use the definition of thermal insulation system as proposed by Hilal and Boom [Hilal and Boom, 1977], and later also used by Kokkolaras *et al.* [Kokkolaras et al., 2001]. Thermal insulation systems use heat intercepts to minimize the heat flow from a hot to a cold surface. The cooling temperature T_i is a control imposed at the $i = 1, 2, ..., n$ locations x_i to *intercept* the heat. The design configuration of such a multi-intercept thermal insulation system is defined by the number of intercepts, their locations, temperatures, and types of insulators placed between each pair of adjacent intercepts. Figure 6.7 presents the schematic of a thermal insulation system.

The optimization of a thermal insulation system consists of minimizing the total refrigeration power P required by the system, which is a sum of the refrigeration power P_i needed at all n intercepts:

$$
\begin{gathered}
f(\mathbf{x},\mathbf{T}) = \sum_{i=1}^{n} P_i = AC_i \left(\frac{T_{\text{hot}}}{T_{\text{cold}}} - 1\right)\left(\frac{1}{\Delta x_i}\int_{T_i}^{T^i+1} kdT - \frac{1}{\Delta x_{i-1}}\int_{T_{i-1}}^{T^i} kdT\right), \\
i = 1, 2, ..., n, \quad \sum_{i=1}^{n} \Delta x_i = L,
\end{gathered}
\tag{6.10}
$$

where C_i is a thermodynamic cycle efficiency coefficient at the i-th intercept, A is a constant cross-section area, k is the effective thermal conductivity of the insulator, and L is the total thickness of the insulation.

Kokkolaras *et al.* define the problem based on a general mathematical model of the thermal insulation system:

$$
\min_{n,\mathbf{I},\Delta\mathbf{x},\mathbf{T}} f(n,\mathbf{I},\Delta\mathbf{x},\mathbf{T}), \quad \text{subject to} \quad g(n,\mathbf{I},\Delta\mathbf{x},\mathbf{T}), \tag{6.11}
$$

where $n \in \mathbb{N}$ is the number of intercepts used, $\mathbf{I} = \{I_1, ..., I_n\}$ is a vector of insulators, $\Delta\mathbf{x} \in \mathbb{R}^n$ is a vector of thicknesses of the insulators, $\mathbf{T} \in \mathbb{R}^n_+$ is the vector of temperatures at each intercept, and $g(\cdot)$ is a set of constraints.

The applicable constraints come directly from the way the problem is defined. They are presented in Table 6.13.

Table 6.13: Constraints for the thermal insulation system design problem.

No	Constraint
g_1	$\Delta x_i \geq 0,\ i = 1, ..., n+1$
g_2	$T_{\text{cold}} \leq T_1 \leq T_2 \leq ... \leq T_{n-1} \leq T_n \leq T_{\text{hot}}$
g_3	$\sum_{i=1}^{n+1} \Delta x_i = L$

Kokkolaras *et al.* [Kokkolaras et al., 2001] have shown that the minimal refrigeration power needed decreases with the increase in the number of intercepts used. However, the more intercepts are used, the more complex and expensive becomes the task of manufacturing the thermal insulation system. Hence, due to practical reasons, the number of intercepts is usually limited to a value function of the manufacturing capabilities, and it may be chosen in advance.

Considering that the number of intercepts n is defined in advance, and based on the model presented, we may define the following problem variables:

- $I_i \in \mathbf{M},\ i = 1, ..., n{+}1$ — the material used for the insulation between the $(i{-}1)$-th and the i-th intercepts (from a set of $\mathbf{M}$ materials).
- $\Delta x_i \in \mathbb{R}_+,\ i = 1, ..., n + 1$ — the thickness of the insulation between the $(i - 1)$-th and the i-th intercepts.
- $\Delta T_i \in \mathbb{R}_+,\ i = 1, ..., n + 1$ — the temperature difference of the insulation between the $(i - 1)$-th and the i-th intercepts.

This way, for a TISD problem using n intercepts, there are $3(n + 1)$ problem variables. Of these, there are $n{+}1$ categorical variables chosen form a set $\mathbf{M}$ of available materials. The remaining $2n + 2$ variables are continuous—positive real values.

In order to be able to evaluate the objective function for a given TISD problem according to Equation 6.10, it is necessary to define several additional parameters. These are: the set of available materials, the thermodynamic cycle efficiency coefficient at i-th intercept C_i, the effective thermal conductivity of the insulator k for the available materials, the cross-section A, and the total thickness L of the insulation.

Since both the cross-section and the total thickness have only linear influence on the value of the objective function, we use normalized values $A = 1$ and $L = 1$ for simplicity. The thermodynamic cycle efficiency coefficient is a function of the temperature, as follows:

$$C = \begin{cases} 2.5 & \text{if} \quad T \geq 71\,\mathrm{K} \\ 4 & \text{if} \quad 71\,\mathrm{K} > T > 4.2\,\mathrm{K} \\ 5 & \text{if} \quad T \leq 4.2\,\mathrm{K} \end{cases} \tag{6.12}$$

The set of materials defined initially by Hilal and Boom [Hilal and Boom, 1977], and later also used by Kokkolaras *et al.* [Kokkolaras et al., 2001] includes: teflon (T), nylon (N), epoxy-fiberglass (in plane cloth) (F), epoxy-fiberglass (in normal cloth) (E), stainless steel (S), aluminum (A), and low-carbon steel (L):

$$\mathbf{M} = \{\mathrm{T, N, F, E, S, A, L}\} \tag{6.13}$$

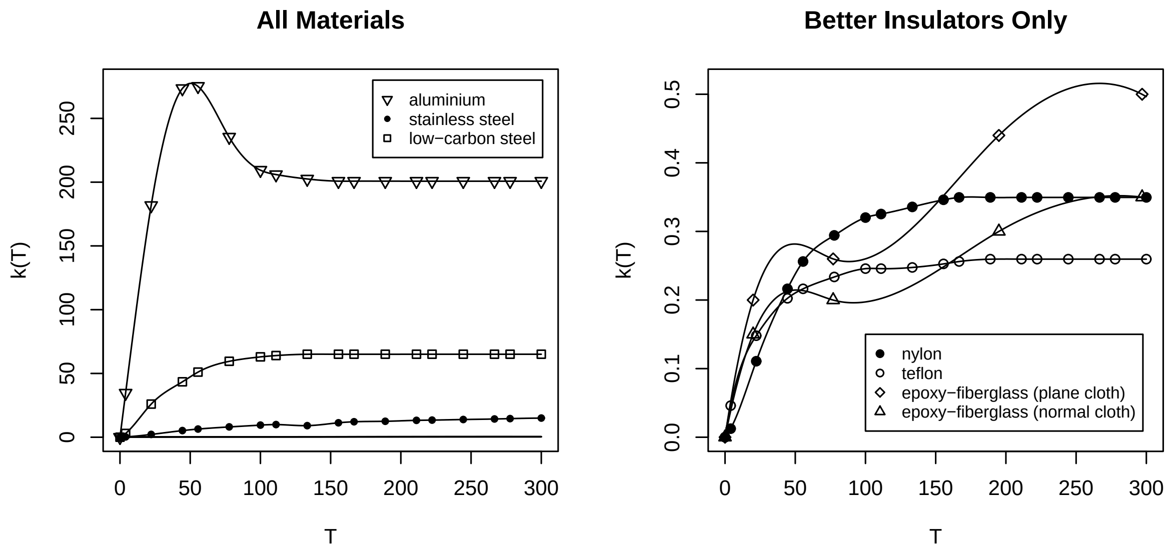


Figure 6.8: The effective thermal conductivity for the various materials in the function of temperature. Tabulated data fitted with cubic splines. The left plot presents all materials, and the right plot only the better insulators (visible on the left plot as the lowest line).

Certainly, this set may be divided intuitively into two groups. The better insulators would include the teflon, nylon, and both epoxy-fiberglass ones. The worse ones would be the aluminum and both types of steel. The effective thermal conductivity k of all these insulators varies heavily with the temperature. Figure 6.8 shows respectively all of the insulators on the left plot, and only the group of the better ones in the right plot. The tabulated data comes from [Barron, 1966]. Since the ACO$_{\mathbb{R}}$ is implemented in R, the tabulated data has been then fitted with cubic splines directly in R.

The plots of the effective thermal conductivity for the available materials clearly present, why the variables describing the materials are *categorical* and may not be easily ordered. While for some temperature ranges ΔT some materials have better insulation properties, for other temperature ranges, it changes. Hence, one cannot say that one material is always better than another, and thus it is impossible to define a single proper ordering of the materials.

The effective thermal conductivity k of all these insulators varies heavily with the temperature and does so differently for different materials. Hence, which material is better depends on the temperature and it is impossible to define a temperature-independent ordering of the insulation effectiveness of the materials.

The tabulated data of the effective thermal conductivity k that we use in this work comes from [Barron, 1966]. Since ACO$_{\mathbb{R}}$ is implemented in R, the tabulated data has been fitted with cubic splines directly in R for the purpose of calculating the integrals in the objective function given in Equation 6.10.

Experimental Setup

Certainly, the problem of choosing the right materials for insulators presented here is not overly complex, as it may be easily noted that only three out of the seven materials presented are indeed best in some temperature ranges (i.e., nylon, teflon, and epoxy-fiberglass in normal cloth). Further, these temperature ranges are reasonably well defined, and an optimization algorithm could be designed that uses this information effectively.

However, this information has not been used in previous studies in this way, and in order to obtain reasonably comparable results, we have also refrained from making an active use of it. Hence, for our ACO$_{\mathbb{R}}$ algorithms, the characteristic of the materials is not known. The integral over the thermal conductivity is only computed when calculating the objective function value for a given solution that has been created by the ants.

The constraints defined in Table 6.13 are met through the use of either a penalty or a repair function. First, constraint g_3 is met through normalizing the Δx values. The constraint g_2 is met through the design choice of using ΔT as a variable and ensuring that no ΔT may be negative. The latter is ensured together with meeting the constraint g_1 by checking if any Δx of ΔT chosen is negative. If it is, the objective function returns infinity as the solution quality.

The problem may be defined for a different number of intercepts. It has been shown by Kokkolaras *et al.* [Kokkolaras et al., 2001] that generally adding more intercepts allows to obtain better results. However, due to practical reasons, too many intercepts cannot be used in a real thermal insulation system. Hence, we decided to limit ourselves to $n = 10$ intercepts in our experiments (that is, our TISD instance has $3(n + 1) = 33$ decision variables).

Further, in order to obtain results comparable to those reported in the literature, we set to 2,350 the maximum number of objective function evaluations for our algorithms. In this way, the results obtained by ACO$_{\mathbb{R}}$ can be compared to those reported by Kokkolaras *et al.* [Kokkolaras et al., 2001].

All the results reported were obtained using 100 independent runs of both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on each of the problem instance tackled.

Parameter Tuning

Similarly to the problems presented earlier, we have used the F-RACE method to choose the parameters. The parameters chosen this way for both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ are summarized in Table 6.14.

Table 6.14: Summary of the parameters chosen for ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ for the TISD problem.

Parameter	Symbol	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
number of ants	m	2	2
speed of convergence	ξ	0.8	0.9
locality of the search	q	0.01	0.025
archive size	k	50	50

Results

Contrary to the two earlier mixed-variable optimization problems that we tackled, this one clearly contains categorical variables. The materials that may be used for insulation show different qualities depending on the temperature. This makes them difficult to be ordered *a priori*. TISD makes a perfect benchmark problem to test our hypothesis that in case of problems containing categorical variables, ACO$_{\mathbf{MV}}$ should outperform ACO$_{\mathbb{R}}$.

It could be seen in the earlier two example problems that if the discrete variables could be easily ordered, ACO$_{\mathbb{R}}$ was able to exploit this fact and converge faster to good results. The TISD problem is different, the categorical variables may not be easily ordered, and hence we would expect ACO$_{\mathbf{MV}}$ to perform better this time.

Our motivation was not only to compare ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$, but also to see how they perform comparing to other algorithms used to tackle this problem.

Because of the experimental setup that we chose, the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ could only be compared to the results obtained using mixed-variable programming

(MVP) by Kokkolaras *et al.* [Kokkolaras et al., 2001].[5]

Table 6.15: The results obtained by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on the TISD problem using $n = 10$ intercepts (i.e., 33 decision variables) and 2,350 function evaluations, compared to those reported in the literature.

	MVP	ACO$_{\mathbb{R}}$	ACO$_{\mathbf{MV}}$
min	25.36336	27.91392	25.27676
median	-	36.26266	26.79740
max	-	198.90930	29.92718

The results obtained by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$, as well as the result of MVP algorithm, are summarized in Table 6.15. As mentioned earlier, we followed the experimental setup used by Kokkolaras *et al.* [Kokkolaras et al., 2001] for one of their experiments. We used the TISD problem instance with $n = 10$ intercepts (i.e., $3(n+1) = 33$ decision variables) and only 2,350 function evaluations.

It may be observed that indeed, this time ACO$_{\mathbf{MV}}$ significantly outperforms ACO$_{\mathbb{R}}$. This clearly supports our initial hypothesis. Similarly to the initial benchmark problem we used, also on the TISD problem containing categorical variables, the native mixed-variable ACO$_{\mathbf{MV}}$ approach outperforms ACO$_{\mathbb{R}}$.

Comparing ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ performance to MVP is more complicate. While we have done 100 independent runs of both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$, Kokkolaras *et al.* reports only one single result. Also, Kokkolaras *et al.* used a Matlab implementation to fit the tabulated data and compute the integrals required to calculate the value of the objective function, while we used R. Furthermore, the MVP results for $n = 10$ intercepts were obtained with MVP using additional information about the problem. Due to all these reasons, it is not possible to compare very precisely the results obtained by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ to those of MVP. However, it may be concluded that the best results obtained by ACO$_{\mathbf{MV}}$ and MVP under similar conditions are comparable.

[5] A similarly defined problem was also tackled by Hilal and Boom [Hilal and Boom, 1977], but they only considered very simple cases.

6.4 Discussion

In this chapter, we have considered two approaches to the approximate solution of mixed-variable optimization problems. The first is based on ACO$_{\mathbb{R}}$ and uses a continuous-relaxation approach. The second is based on ACO$_{\mathbf{MV}}$, an extension of the ACO meta-heuristic, which uses a native mixed-variable approach.

The two approaches have advantages and disadvantages. Our initial hypothesis was that while ACO$_{\mathbb{R}}$ should perform better on problems containing discrete variables that can be *ordered*, ACO$_{\mathbf{MV}}$ should perform better on problems where a proper ordering is not possible, or unknown. We have proposed a new benchmark problem based on a rotated ellipsoid function in order to evaluate the difference in performance between ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$.

We have also run tests applying ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ to the solution of three representative mixed-variable engineering test problems. For each of the problems, we compared the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ with the results reported in the literature.

Our initial hypothesis was supported by the results of our experiments. While ACO$_{\mathbb{R}}$ performed better on the pressure vessel design and on the coil spring design problems which did not contain any categorical variables, ACO$_{\mathbf{MV}}$ was significantly better on the thermal insulation system design problem which on the contrary contained categorical variables. Also, the results obtained by ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ were usually as good as the best results reported in the literature for the considered test problems. Hence, it may be concluded that both ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ are competitive approaches for mixed-variable problems (in their versions without and with categorical variables respectively).

Further studies of the performance of ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ on various mixed-variable optimization problems are needed in order to better understand their respective advantages and disadvantages. Also, the current R-based implementation is not particularly efficient, nor it has been properly optimized. In order to use it for practical purposes, it should be re-implemented in C, or other compiled language and properly optimized.

Chapter 7

Conclusions

Approximate methods for solving optimization problems have been around for quite many years now. Nevertheless, the research in this area is still very active, with new findings appearing every year. The results reported in the literature show a constant increase in quality and robustness of the algorithms. This demonstrates that there is still room for improvement: better, more efficient, and more robust methods can still be developed.

Ant colony optimization is a relatively new metaheuristic, which means that there is certainly still a large potential for improvement and development. While the performance of ACO on static combinatorial problems has been already reasonably well understood, its application to other types of problems and its use in parallel environments is not that well studied. Current research concerning ACO concentrates on adapting it to different kinds of problems, which includes continuous and mixed-variable problems, dynamic problems, or problems involving uncertainty and noise. Also the application of ACO in parallel environments is given significant attention.

Our work fits neatly into this general trend of research by presenting a way to apply ACO to continuous and mixed-variable optimization problems. The main contributions of this thesis may be summarized in the following list:

- **Formal and coherent definition of the combinatorial, continuous, and mixed-variable optimization problems:** We provide a coherent definition of combinatorial, continuous, and mixed-variable problems based on the way the combinatorial problems are usually defined. This allows demonstrating how ACO algorithms may also tackle the continuous and mixed-variable optimization problems by emphasizing the crucial differences and similarities between these types of problems.

- **Ant colony optimization algorithm for continuous domains ($ACO_{\mathbb{R}}$):** One of the most significant contributions of this thesis. We present a novel idea on how ACO may be extended to continuous domains with the pheromone modeled by probability density functions instead of a table. We describe the underlying idea; we present a fully functional algorithm—$ACO_{\mathbb{R}}$—and we evaluate its performance on a large number of benchmark problems.

- **Application of $ACO_{\mathbb{R}}$ to training neural networks for pattern classification in the medical field:** We evaluate the performance of $ACO_{\mathbb{R}}$ on a real-world problem. We present how $ACO_{\mathbb{R}}$ may be applied to the problem of training neural networks for pattern classification in the medical field. We demonstrate that the performance of $ACO_{\mathbb{R}}$ is competitive, when compared to genetic algorithms, and that a hybridized version of $ACO_{\mathbb{R}}$ with a derivative-based method outperforms typical methods used for neural network training.

- **A benchmark mixed-variable optimization problem with well-controlled characteristics:** We propose a new mixed-variable benchmark problem, which provides a well-controlled environment for testing algorithms. It allows for adjusting its characteristics and difficulty level, and hence provides an excellent testing ground for mixed-variable optimization algorithms.

- **Ant colony optimization algorithm for mixed-variable domains ($ACO_{\mathbf{MV}}$):** Following the general idea of $ACO_{\mathbb{R}}$, we propose a further extended version of this algorithm—$ACO_{\mathbf{MV}}$—that is able to handle both continuous and discrete decision variables. We investigate its basic performance using the benchmark problems that we proposed.

- **Application of $ACO_{\mathbb{R}}$ and $ACO_{\mathbf{MV}}$ algorithms to real-world engineering mixed-variable optimization problems:** We apply our mixed-variable $ACO_{\mathbf{MV}}$ algorithm along with the $ACO_{\mathbb{R}}$ algorithm mentioned earlier to three engineering real-world mixed-variable optimization problems. We demonstrate how, as the function of a problem formulation, each of them has certain advantages. We compare the results obtained with those that have been obtained using other methods found in the literature.

While the use of probability density functions for modeling continuous search spaces seems quite obvious *a posteriori*, it has not been proposed in the context of ant colony optimization before our research began. It is therefore our pioneering and original contribution to the ant colony optimization research field.

A significant portion of the ideas presented in this thesis has been already published in conference proceedings, book chapters, and international journals. Also, numerous requests have led to making the code developed as part of this thesis available online under the GPL license. This attention shows that the topic tackled is interesting and useful, and that the quality of results obtained is encouraging for others to use and possibly improve it.

Building on the findings presented in this work, it is possible to further improve the performance and the range of applications of ACO algorithms. Hence, the algorithms developed in the course of this research, while being useful and comparable to the state of the art in many cases, may be also the stepping stone to developing even more efficient and robust algorithms for many new optimization problems. This also means that the research on this topic is hardly finished. We hope to continue it, so that both the theoretical and practical aspects of the proposed solutions could be further analyzed and understood. Practical implementations need to be found, which are more robust and efficient. Also, it will be extremely satisfying, if the $\mathrm{ACO}_{\mathbb{R}}$ or $\mathrm{ACO}_{\mathbf{MV}}$ algorithms could be used for solving practical real-world problems with good results.

As part of future work plans, we aim to develop a more efficient, optimized versions of the proposed algorithms. As such, they will able to tackle also more complex problems, which due to the interpreted nature of the R language could not be handled by the existing version at this time. We have already identified some very interesting real-world problems coming from the medical field, which we could not tackle using the existing implementation due to too long run time. An improved implementation (for instance in C) could allow to tackle such problems.

Also, as mentioned in the thesis, there are several design decisions and several possible parameters that can be used to fine tune the performance of the algorithm. More research is needed to understand better their interactions and how each of them can have influence on the algorithm performance. This is particularly interesting, if this may be related to the characteristics of the problem. If so, some automated way of tuning the algorithm based on the problem at hand could be proposed.

Finally, the application of our algorithm to mixed-variable problems is only the tip of the iceberg. Although the mixed-variable problems are encountered quite frequently in the real world, there are few benchmarks available in the literature. This is due to the fact that until recently there were very few algorithms that could natively handle these problems. Hence the real-world mixed-variable problems were reformulated or divided so that they could have been solved by existing algorithms. With more and more native mixed-variable optimization algorithms appearing, there is an increasing number of problems being evaluated. We plan to continue experimentation with these new algorithms in order to first gain better understanding of their operation, and eventually further improve their performance.

Appendix A

Variable Correlation Handling

ACO algorithms in general do not exploit correlation information between different decision variables (or components). In $\text{ACO}_{\mathbb{R}}$, due to the specific way the pheromone is represented (i.e., as the solution archive), it is in fact possible to take into account the correlation between the decision variables. Consider Fig. A.1, where the same Ellipsoid test function defined as:

$$f_{EL}(\vec{x}) = \sum_{i=1}^{n} (5^{\frac{i-1}{n-1}} x_i)^2, \tag{A.1}$$

is presented—not rotated (left), and then randomly rotated (right). The test function is presented as the $\text{ACO}_{\mathbb{R}}$ algorithm sees it—as a set of points representing different solutions found by the ants and stored in the solution archive. The darker the point, the higher the quality of the solution (and the higher its rank). While on the left plot the variables are not correlated (i.e., for good solutions, the value of one coordinate does not depend on the value of the other coordinate), on the right plot they are highly correlated.

The default coordinate system that corresponds to the set of the original decision variables $x_i, i = 1, 2$, is marked in bold. It is clear that the axes of that coordinate system align well with the scaling of the test function in the left plot. The example Gaussian kernel PDFs for both of the dimensions are indicated on the right and above the plot. Clearly a new solution generated using them would fall somewhere in the promising region. In contrast, in the case of the rotated Ellipsoid function presented on the right

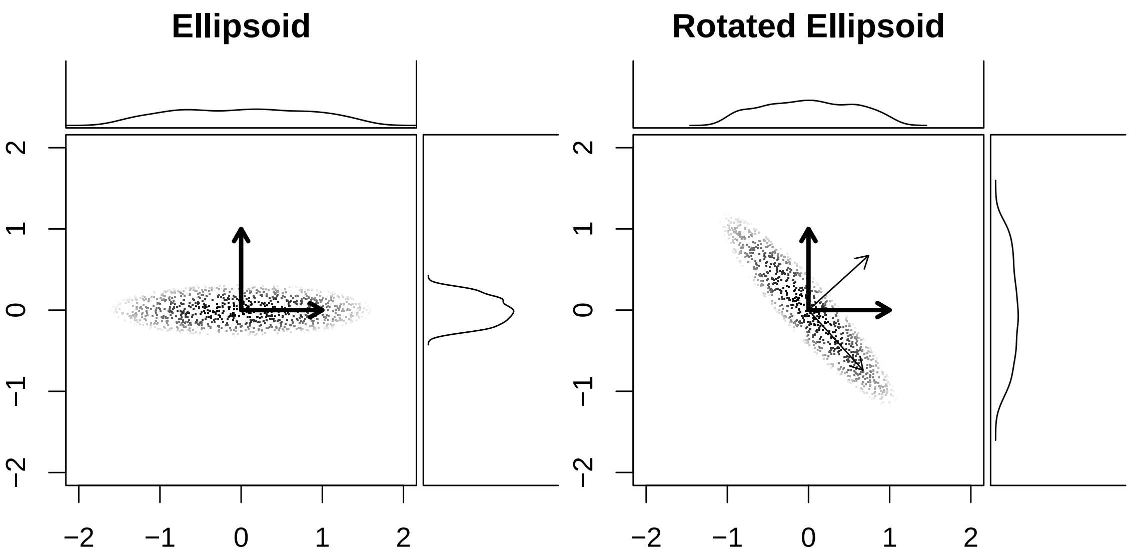


Figure A.1: Example of the Ellipsoid function not rotated (left) and rotated by 45° (right). It illustrated by 10000 points—of which only the best 1000 are visible (the darker the point the higher rank). The original coordinate system has been marked in bold, and the optimal one is also indicated on the right plot. Also, the examples of the Gaussian kernel PDFs as generated using the default coordinate system, are given on the right and above.

plot, the PDFs created with the default coordinate system cover roughly the whole search space (here $\mathbf{D} = [-2, 2]^2$). If sampling was done based on the original coordinate system, the points sampled would most likely be far from being good. The optimal coordinate system that should be used by the ants in this case is also indicated on the right plot. If ants used that coordinate system instead of the original one, the sampling would be as efficient as in the case of the non-rotated Ellipsoid function.

Our ACO$_\mathbb{R}$ algorithm dynamically adapts the coordinate system used by each ant in order to minimize the correlation between different decision variables. The adaptation of the coordinate system is actually accomplished by expressing the set of decision variables $\mathbf{X}$ with temporary variables $Z_i, i = 1, ..., n$ that are linear combinations of $X_i, i = 1, ..., n$. Although a popular method for accomplishing it is the Principle Component Analysis (PCA), we have implemented another method—a non-deterministic adaptive one. In the following sections, we shortly present PCA as well as the method chosen, and we provide the reasoning for our decision.

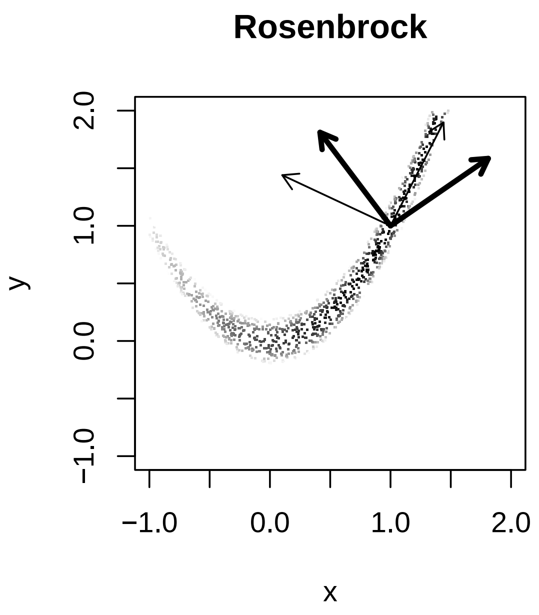


Figure A.2: Example of the Rosenbrock function illustrated by 10000 points—of which only the best 1000 are visible (the darker the point the higher rank). The coordinate system chosen by PCA is marked in bold, and it is different than the optimal one at this stage of the search process.

A.1 Principal Component Analysis

An obvious choice for adapting the coordinate system to the distribution of the solutions in the archive is the well-known technique of principal component analysis (PCA). It is based on a statistical analysis of the solutions in the archive in order to distinguish the principal components. This is usually done through the calculation of eigenvectors and eigenvalues in the process called *eigen decomposition*. For details we refer an interested reader for instance to [Hastie et al., 2001].

Although, this technique is quite efficient in case of reasonably easy and regular functions, such as the Ellipsoid function, its performance proved to be no so interesting in case of more difficult problems such as the Rosenbrock function.[1]

We have observed some drawbacks of PCA, when used for adapting the coordinate system in $ACO_{\mathbb{R}}$. For more irregular functions, such as the Rosenbrock function, the PCA adapts the coordinate system by considering all the points in the archive, what

[1]See Tab. 4.1 for the definition of the Rosenbrock function.

often places too much emphasis on points that are not so good. This results frequently in having a sub-optimal coordinate system chosen. Such situation is for instance presented in Figure A.2, where the optimal coordinate system is indicated, as well as the one chosen by PCA (in bold).

The choice of sub-optimal coordinate system in during one of the iterations of the algorithm is not a major problem. However, due to the fact that PCA is a deterministic algorithm, this often leads to stagnation and is difficult to overcome. In order to solve this issue, we have tried to define a *local PCA* that would only take into consideration a certain percentage of the best best points in the solution archive. The goal was, to ensure that only the important points are taken into account, and by that reduce the probability of the algorithm stagnation.

We have used the F-RACE method [Birattari et al., 2002; Birattari, 2005] for choosing the right configuration of $\text{ACO}_{\mathbb{R}}$ parameters and the percentage of best solutions to be used by PCA. This method has been already presented in some detail in Section 5.2.2. We have considered using 20%, 40%, 60%, 80%, and 100% of the best solutions from the archive by PCA. Together with other parameters of $\text{ACO}_{\mathbb{R}}$, we defined 660 different parameter configurations in total. As a test function, we used the Rosenbrock function ($n = 10$):

$$f_{Rn}(\vec{x}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2. \tag{A.2}$$

The best performing candidate used 100% of the solutions from the archive. This clearly indicates that the idea of using *local PCA* was not efficient. In fact, decreasing the percentage of the solutions used by the PCA degraded performance of the algorithm. Also, the best performing candidate required still two times more function evaluations on this test function, than the algorithm using *non-deterministic adaptive method* presented in the next section.

A.2 Non-Deterministic Adaptive Method

The mechanism that we designed instead of PCA, is relatively simple. Its main difference from PCA is the fact that it is not deterministic. This means that for a given set of solutions present in the solution archive, each ant uses a slightly different coordinate system. This is accomplished as follows.

Each ant at each step of the construction process chooses a *direction* in the search space. The direction is chosen by randomly selecting a solution s_u that is reasonably far away from the solution s_l chosen earlier as the mean of the PDF. Then, the vector $\vec{s_l s_u}$ becomes the chosen direction. The probability of choosing solution s_u at step i (having chosen earlier solution s_l as the mean of the PDF) is the following:

$$p(s_u|s_l)_i = \frac{d(s_u, s_l)_i^4}{\sum_{r=1}^{k} d(s_r, s_l)_i^4} , \tag{A.3}$$

where the function $d(\cdot, \cdot)_i$ returns the Euclidean distance in the $(n-i+1)$-dimensional search sub-space[2] between two members of the solution archive T. Once this vector is chosen, the new orthogonal basis for the ant's coordinate system is created using the Gram-Schmidt process [Golub and van Loan, 1989]. It takes as input all the (already orthogonal) directions chosen in earlier ant's steps and the newly chosen vector. The remaining missing vectors (for the remaining dimensions) are chosen randomly. Then, all the current coordinates of all the solutions in the archive are rotated and recalculated according to this new orthogonal base resulting in the set of new temporary variables $Z_i, i = 1, ..., n$.

At the end of the solution construction process, the chosen values of the temporary variables $Z_i, i = 1, ..., n$ are converted back into the original coordinate system, giving rise to a set of values for the original decision variables $X_i, i = 1, ..., n$.

This simple, non-deterministic mechanism makes the ACO$_{\mathbb{R}}$ algorithm much more robust. This is especially visible in case of irregular functions, such as the Rosenbrock function, for which (for $n = 10$), ACO$_{\mathbb{R}}$ with our non-deterministic method performs twice as good as with PCA. It has to be noted of course, that for very regular functions, such as for instance the Ellipsoid function, the performance of ACO with PCA is better than the performance of ACO$_{\mathbb{R}}$ with our proposed method. However, since the real-world problems are usually not regular, we chose not to use PCA, as it was less robust than our non-deterministic method.

[2]At step i, only dimensions i through n are used.

Appendix B

Alternative Description of ACO$_\mathbb{R}$

In some initial publications on the subject, a different description of the ACO$_\mathbb{R}$ algorithm was used than the one provided in Chapter 4. We found the new one to be more clear and elegant, while being functionally identical. For the sake of complitness, we also provide the initial description in this appendix.

B.1 Probability Density Function (PDF)

In principle, a *probability density function* may be any function $P(x) : \mathbb{R} \ni x \rightarrow P(x) \in \mathbb{R}$ such that:

$$\int_{-\infty}^{\infty} P(x)dx = 1. \tag{B.1}$$

For a given probability density function $P(x)$, an associated *cumulative distribution function* (CDF) $D(x)$ may be defined, which is often useful when sampling the corresponding PDF. The CDF $D(x)$ associated with PDF $P(x)$ is defined as follows:

$$D(x) = \int_{-\infty}^{x} P(t)dt. \tag{B.2}$$

The general approach to sampling PDF $P(x)$ is to use the *inverse* of its CDF, $D^{-1}(x)$.

When using the inverse of the CDF, it is sufficient to have a pseudo-random number generator that produces uniformly distributed real numbers.[1] However, it is important to note that for an arbitrarily chosen PDF $P(x)$, it is not always straightforward to find $D^{-1}(x)$.

One of the most popular functions that is used as a PDF is the Gaussian function. It has some clear advantages, such as a reasonably easy way of sampling—e.g., the Box-Muller method [Box and Muller, 1958]—but it also has some disadvantages. A single Gaussian function is not able to describe a situation where two disjoint areas of the search space are promising, as it only has one maximum. Due to this fact, we use a PDF based on Gaussian functions, but slightly enhanced—a Gaussian kernel PDF. Similar constructs have been used before [Bosman and Thierens, 2000], but not exactly in the same way. We define a Gaussian kernel as a weighted sum of several one-dimensional Gaussian functions $g_l^i(x)$, and denote it as $G^i(x)$:

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{-\frac{(x-\mu_l^i)^2}{2{\sigma_l^i}^2}} \quad . \tag{B.3}$$

Since we use as many Gaussian kernel PDFs as there is number of dimensions of the problem, $i = 1, ..., n$ identifies a single such PDF. The Gaussian kernel $G^i(x)$ is parametrized with three vectors of parameters: $\boldsymbol{\omega}$ is the vector of weights associated with the individual Gaussian functions, $\boldsymbol{\mu}^i$ is the vector of means, and $\boldsymbol{\sigma}^i$ is the vector of standard deviations. The cardinality of all these vectors is equal to the number of Gaussian functions constituting the Gaussian kernel. For convenience, we will use the parameter k to describe it, hence $|\boldsymbol{\omega}| = |\boldsymbol{\mu}^i| = |\boldsymbol{\sigma}^i| = k$.

Such a PDF allows a reasonably easy sampling, and yet provides a much increased flexibility in the possible shape, in comparison to a single Gaussian function. An example of how such a Gaussian kernel PDF may look like is presented in Figure B.1.

B.2 Pheromone Representation in ACO$_{\mathbb{R}}$

In ACO for combinatorial optimization, pheromone information is stored as a table. At each iteration, when choosing a component to be added to the current partial solution

[1]Such pseudo-random number generators are routinely available for most programming languages.

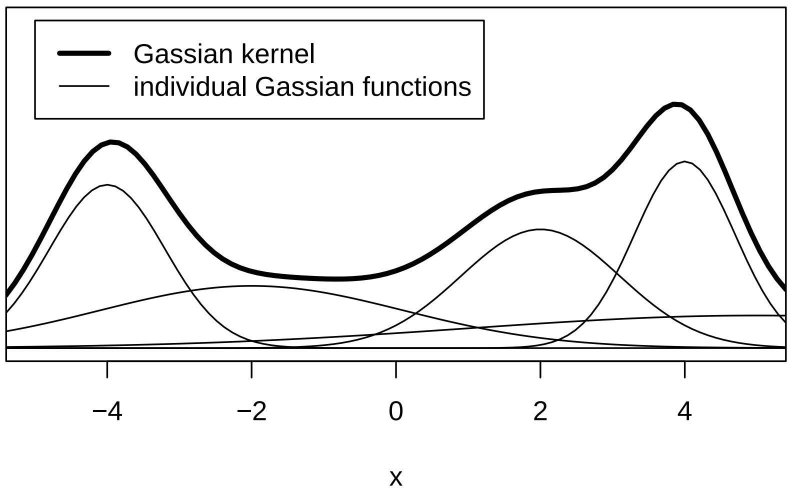


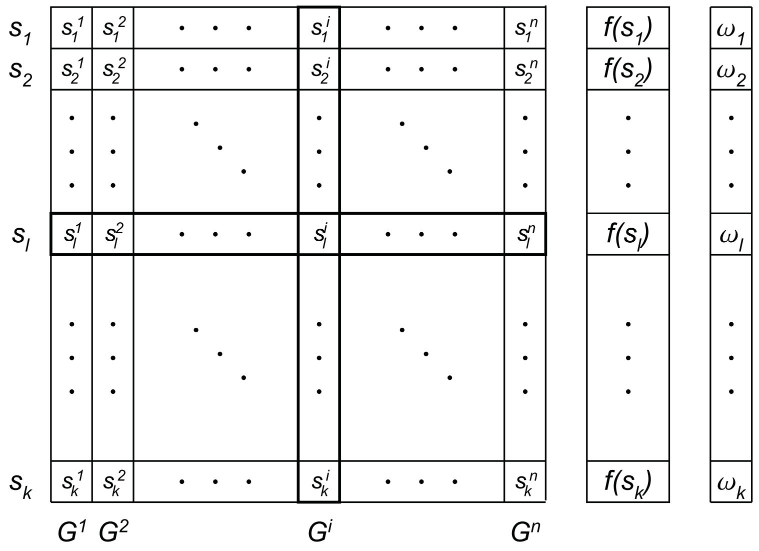
Figure B.1: Example of five Gaussian functions and their superposition–the resulting Gaussian kernel (illustration limited to the range $x \in [-5, 5]$).

(according to Eq. 3.2), an ant uses some of the values from that table as a discrete probability distribution (Figure 4.1a). In the case of continuous optimization, the choice an ant makes is not restricted to a finite set (Figure 4.1b). Hence, it is impossible to represent the pheromone in the form of a table. A different approach has to be adopted.

We use an idea similar to that proposed by Guntsch and Middendorf in *Population-Based ACO* (PB-ACO) [Guntsch and Middendorf, 2002b]. In PB-ACO the pheromone table is updated based on the components of good solutions found—just like in regular ACO. However, in regular ACO, the actual solutions found by the ants are discarded once the pheromone table has been updated. In contrast, PB-ACO keeps track of a certain number of the solutions used to update the pheromone table. Instead of using pheromone evaporation, the pheromone associated with the oldest solutions is eventually removed by performing a negative update on the pheromone table—thus canceling its influence.

In ACO$_\mathbb{R}$ we also keep track of a number of solutions in a *solution archive* T. For each solution s_l to an n-dimensional problem, ACO$_\mathbb{R}$ stores in T the values of its n variables and the value of the objective function $f(s)$. The i-th variable of the l-th solution is hereby denoted by s_l^i. The structure of the solution archive T is presented in Figure B.2.

While in case of PB-ACO the components of the solutions are used directly to modify the pheromone table, in the continuous case we use them to dynamically generate probability density functions. In order to accomplish this, a method for generating a PDF based



s_1
s_2
s_l
s_k
s_1^1 s_1^2 s_1^i s_1^n
s_2^1 s_2^2 s_2^i s_2^n
s_l^1 s_l^2 s_l^i s_l^n
s_k^1 s_k^2 s_k^i s_k^n
$f(s_1)$
$f(s_2)$
$f(s_l)$
$f(s_k)$
ω_1
ω_2
ω_l
ω_k
G^1 G^2 G^i G^n

Figure B.2: The archive of solutions kept by ACO$_{\mathbb{R}}$. The solutions are ordered in the archive according to their quality—i.e., for a minimization problem: $f(s_1) \leq f(s_2) \leq ... \leq f(s_l) \leq ... \leq f(s_k)$. Each solution has an associated weight ω that depends on the solution quality. Therefore, $\omega_1 \geq \omega_2 \geq ... \geq \omega_l \geq ... \geq \omega_k$. The PDF G^i is constructed using only the i-th coordinates of all k solutions from the archive.

on a set of memorized solutions is defined. As indicated in Section B.1 (Eq. B.3), the Gaussian kernel PDF is parametrized by three vectors $\boldsymbol{\omega}$, $\boldsymbol{\mu}^i$, and $\boldsymbol{\sigma}^i$ (each of cardinality k). The solutions in the archive are used to calculate the values of these parameters, and hence shape the Gaussian kernel PDF used to guide the ants in their search process.

The number of solutions memorized in the archive is set to k and this parameter determines therefore the complexity of the PDF: There are k separate Gaussian functions making up the Gaussian kernel PDF. For each dimension $i = 1, ..., n$ of the problem, there is a different Gaussian kernel PDF G^i defined (see Figure B.2). For each such G^i, the values of the i-th variable of all the solutions in the archive become the elements of the vector $\boldsymbol{\mu^i}$:

$$\boldsymbol{\mu^i} = \{\mu_1^i, ..., \mu_k^i\} = \{s_1^i, ..., s_k^i\}. \tag{B.4}$$

The vector of weights $\boldsymbol{\omega}$ is created in the following way. Each solution that is added to the archive T is evaluated and ranked (ties are broken randomly). The solutions in the archive are sorted according to their rank—i.e., solution s_l has rank l. The weight ω_l of the solution s_l is calculated according to the following formula:

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}}, \tag{B.5}$$

which essentially defines the weight to be a value of the Gaussian function with argument l, mean 1.0, and standard deviation qk, where q is a parameter of the algorithm. When q is small, the best-ranked solutions are strongly preferred, and when it is large, the probability becomes more uniform. The influence of this parameter on ACO$_{\mathbb{R}}$ is similar to adjusting the balance between the *iteration-best* and the *best-so-far* pheromone updates used in ACO. A more detailed analysis of the influence of the parameter q is presented in Section 4.3.4.

In order to find the final shape of each Gaussian kernel PDF G^i, the vector $\boldsymbol{\sigma}^i$ of the standard deviations must still be defiend. The detailed description of how this is accomplished is presented in the following section as part of the description of the solution construction process.

B.3 ACO$_{\mathbb{R}}$ Metaheuristic Framework

In this section, we outline the ACO$_{\mathbb{R}}$ version of the three major algorithmic components of the ACO metaheuristic as presented in Algorithm 1.

AntBasedSolutionConstruction(): Given decision variables $X_i, i = 1, ..., n$, an ant constructs a solution by performing n construction steps. At construction step i an ant chooses a value for variable X_i. As mentioned earlier, the Gaussian kernel PDF is composed of a number of regular Gaussian functions. The number of functions used is equal to the size k of the solution archive T. At construction step i, only the information about the i-th dimension (i.e., decision variable X_i) is used. In this way, at each step i the resulting Gaussian kernel PDF G^i is a different one.

Following Eq. B.3, in order to define the PDF G^i, the values of vectors $\boldsymbol{\mu}^i$, $\boldsymbol{\sigma}^i$, and $\boldsymbol{\omega}$ must be defined. While the creation of $\boldsymbol{\mu}^i$ and $\boldsymbol{\omega}$ has been discussed in Section B.2, the computation of the standard deviation vector $\boldsymbol{\sigma}^i$ is the most complex issue. Before presenting how this is done in detail, we explain the practical implementation of Eq. B.3.

In practice, the sampling process is accomplished as follows. First, the elements of the weight vector $\boldsymbol{\omega}$ are computed following Eq. B.5. Then, the sampling is done in two phases. Phase one consists of choosing one of the Gaussian functions that compose the Gaussian kernel. The probability p_l of choosing the l-th Gaussian function is given by:

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r} \, . \tag{B.6}$$

Phase two consists of sampling the chosen Gaussian function (i.e., at step i—function g_l^i). This may be done using a random number generator that is able to generate random numbers according to a parametrized normal distribution, or by using a uniform random generator in conjunction with, for instance, the Box-Muller method [Box and Muller, 1958]. This two-phase sampling is equivalent to sampling the Gaussian kernel PDF G^i as defined in Eq. B.3.

It is clear that at step i, the standard deviation needs only to be known for the single Gaussian function $g_l^i(x)$ chosen in phase one. Hence, we do not calculate the whole vector of standard deviations $\boldsymbol{\sigma}^i$, but only the σ_l^i that is needed.

The choice of the l-th Gaussian function is done only once per ant, per iteration. This means that an ant uses the Gaussian functions associated with the single chosen solution s_l—i.e. functions $g_l^i, i = 1, ..., n$, for constructing the whole solution in a given iteration. This allows exploiting the correlation between the variables, which is explained in detail in Appendix A. Of course, the actual Gaussian function sampled differs at each construction step, as for step i, $\mu_l^i = s_l^i$, and σ_l^i is calculated dynamically, as follows.

In order to establish the value of the standard deviation σ_l^i at construction step i, we calculate the average distance from the chosen solution s_l to other solutions in the archive, and we multiply it by a parameter ξ:

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{|x_e^i - x_l^i|}{k-1}. \tag{B.7}$$

The parameter $\xi > 0$, which is the same for all dimensions, has an effect similar to that of the pheromone evaporation rate in ACO. The higher the value of ξ, the lower the convergence speed of the algorithm. The pheromone evaporation rate in ACO influences the long term memory—i.e., the higher is the evaporation rate, the faster the algorithm focuses on the good solutions found so far by faster *forgetting* the bad ones. ξ in ACO$_{\mathbb{R}}$ influences the way the long term memory is used—i.e., the lower its value, the closer to the current good solutions is the search space sampled. The effect is similar—the search focuses on good solutions found so far.

As we said, this whole process is done for each dimension $i = 1, ..., n$ in turn, and each time the standard deviation σ_l^i is calculated only with the use of the single dimension i. This ensures that the algorithm is able to adapt to a linear transformation of the considered problem (e.g., moving from a sphare model to an ellipsoid, or rotating an ellipsoid).

PheromoneUpdate(): As mentioned earlier, in case of ACO$_{\mathbb{R}}$, the pheromone information is stored as a *solution archive*. This implies that the pheromone update procedure has to perform some form of update on this archive.

The size k of the archive T is a parameter of the algorithm. However, k may not be smaller than the number of dimensions of the problem being solved.[2] At the start of the algorithm, the solution archive T is initialized generating k solutions by uniform random sampling.

Pheromone update is accomplished by adding the set of newly generated solutions to the solution archive T and then removing the same number of worst solutions, so that the total size of the archive does not change. This process ensures that only the best solutions are kept in the archive, so that they effectively guide the ants in the search process.

DaemonActions(): As part of this algorithmic block, the best solution found is updated, so that it may be returned once the termination condition is met. We do not apply any local search heuristics, though this could be easily done to improve the algorithm performance.

[2]This is due to the explicit handling of correlation among variables as explained in Appendix A. In order to be able to rotate the coordinate system properly, the number of points available has to be at least equal to the number of dimensions.

Appendix C

Source Code

The algorithms developed as part of this work are free software; you can redistribute them and/or modify them under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

The source code (in R) is available online from:

`http://iridia.ulb.ac.be/supp/IridiaSupp2008-001`

The code has been tested with R up to version 2.5.1. Subsequent modifications of the R language may require adaptation of the code. The source code is divided into a few seperate files, which we describe in some detail below.

C.1 ACO$_{\mathbb{R}}$ and ACO$_{\mathbf{MV}}$ Algorithms

C.1.1 ACO.R

`ACO.R` file contains is the R implementation of both ACO$_{\mathbf{MV}}$ and ACO$_{\mathbb{R}}$ aglrotithms. In fact, when the problem tackled only contains continuous variables, ACO$_{\mathbb{R}}$$\equiv$ACO$_{\mathbf{MV}}$. Although, the implementation evloved slightly in time, the one presented here was essentially used to obtain the results in this thesis. Note that also routines presented in Section C.1.2 make an integral part of the algorithms.

The main function implementing the ACO$_{\mathbb{R}}$$\equiv$ACO$_{\mathbf{MV}}$ algorithms is as follows:

```
ACO <- function(f, e, n.of.ants, k, q, xi, max.eval, seed, ls)
```

and it takes the following parameters:

- *f* - objective function object - see functions.R for details and examples,
- *e* - max. error value - used when solution quality used as stopping criterion,
- *n.of.ants* - number of ants used in each iteration (≥ 2),
- *k* - size of the solution archive,
- *q* - locality of the search $(0, 1)$,
- *xi* - convergence pressure (0,Inf) - suggested: (0,1),
- *max.eval* - maximal number of function evaluations when used as the stopping criterion (0 means no limit),
- *seed* - random seed (0 means no random seed set),
- *ls* - local search function (if used, otherwise NULL).

C.1.2 routines.R

Additional routines required by both $\text{ACO}_{\mathbb{R}}$ and $\text{ACO}_{\mathbf{MV}}$ are provided in `routines.R` file. Some of them are simple helper functions, while some are cruscial to the algorithms' operation. In particular `gen.X()` and `gen.C()` functions are the functions generating new values for respectively continuous and discrete variables.

C.2 Test Functions

Additionally to the actual algorithms, we make available also the R implementation of all the test functions that we have used for evaluating performance of our algorithms. They may be used by interested researchers to verify the results reported or to perform different type of comparisons or investigations. Also, the presented implementation of the objective functions should serve as an excellent guide and example for anyone wishing to try the algorithms on also other objective functions.

C.2.1 functions.R

Each test function `foo` is defined as a set of objects described below. Each function is considered to be a maximization problem (any optimization function may be easily translated to such problem).

First object (`foo.d`) is a list that is the domain definition - for each variable the variable type must be defined as one of the following:

- **u** - unordered discrete variable,
- **o** - ordered discrete variable,
- **x** - continuous variable.

Following the variable type, the domain must be defined for each variable. For **u** and **o** variables, all possible values must be specified. For **x** variable the lower and upper bound must be given.

NOTE: All the discrete variables MUST be defined before any continuous variables!

Second object (`foo.tf`) is a function performing the mapping from the values generated by the algorithm to the values from the defined earlier domain. Usually, for continuous and ordered variables there is a direct mapping, and for unordered - categorical - variables there is a mapping from the index number to the actual value. There may be some additional translations or repair mechanisms implemented as well - an option left for the designer. See example test functions for details how this may be done.

Third object (`foo.f`) is the actual objective function. The first function that is called must be the foo.tf function in order to map the values generated by the algorithm to the actual values from the respective variables' domains. Only then the actual objective function value may be calculated and returned.

Finally the last object is a collection of all the previous objects of the form:

`foo <- c(f=foo.f,d=list(foo.d),tf=foo.tf,opt=<optimal value>)`,

where the `<optimal value>` is the known a priori optimal value of the objective function. If the optimal value is not known, it is sufficient to specify a high enough value (reminder: all functions are considered to be maximization problems).

Below, a template of a function is given. Additional insight into designing test functions may be obtained through the analysis of the (numerous) examples provided.

```
#########################################################################
# Test Function Template (FOO)
#
# foo.d <- list(
# list("<variable type>",<domain>),
# .
# .
# .
# list("<variable type>",<domain>))
#
# foo.tf <- function(u,o,x) {
# <mapping: X <- f(u,o,x)>
# return(X)
# }
#
# foo.f <- function(u,o,x) {
# X <- foo.tf(u,o,x)
# <function body: y <- f(X)>
# return(y)
# }
#
# foo <- c(f=foo.f,d=list(foo.d),tf=foo.tf,opt=<optimal value>)
```

C.2.2 utilities.R

The additional functions contained in this file provide the additional helper functions used, when defining the objective functions. These inlcude creation of a matrix of random values, or predefining required global variables.

C.2.3 nn.R

The code available in `nn.R` file is related in principle to Chapter 5, where the ACO$_{\mathbb{R}}$ algorithm is used for training feed-forward neural networks (NNs). The implementation of suitable objective functioins is provided along with the NN evaluation function and a couple of other algorithms, such as Backpropagation and Levenberg-Marquardt. For details see Chapter 5.

Bibliography

Ackley, D. H. [1987]. *A connectionist machine for genetic hillclimbing*, Kluwer Academic Publishers, Boston, MA.

Alba, E. and Chicano, J. F. [2004]. Training neural networks with GA hybrid algorithms, *in* K. Deb *et al.* (ed.), *Proceedings of the Genetic and Evolutionary Computation Conference—GECCO 2004*, Vol. 3102 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 852–863.

Arfken, G. [1985]. The method of steepest descents, *Mathemetical Methods for Physicists*, 3rd edn, Academic Press, Orlando, FL, pp. 428–436.

Audet, C. and Dennis Jr., J. [2001]. Pattern search algorithms for mixed variable programming, *SIAM Journal on Optimization* **11**(3): 573–594.

Baluja, S. and Caruana, R. [1995]. Removing the genetics from the standard genetic algorithm, *in* A. Prieditis and S. Russel (eds), *Twelfth International Conference on Machine Learning*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 38–46.

Barron, R. [1966]. *Cryogenic Systems*, McGraw-Hill, New York, NY.

Beyer, H.-G. and Schwefel, H.-P. [2002]. Evolution strategies: A comprehensive introduction, *Journal of Natural Computing* **1**(1): 3–52.

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O. and Schiavinotto, T. [2004]. Metaheuristics for the vehicle routing problem with stochastic demands, *in* X. Yao *et al.* (ed.), *Proceedings of Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference*, Vol. 3242 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 450–460.

Bilchev, G. and Parmee, I. [1995]. The ant colony metaphor for searching continuous design spaces, *in* T. C. Fogarty (ed.), *Proceedings of the AISB Workshop on Evolutionary Computation*, Vol. 993 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 25–39.

Birattari, M. [2005]. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective. PhD thesis*, Vol. 292 of *Dissertationen zur Künstlichen Intelligenz*, Akademische Verlagsgesellschaft Aka GmbH, Berlin, Germany.

Birattari, M., Stützle, T., Paquete, L. and Varrentrapp, K. [2002]. A racing algorithm for configuring metaheuristics, *in* W. B. Langdon *et al.* (ed.), *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufman, San Francisco, CA, pp. 11–18.

Bishop, C. M. [2005]. *Neural Networks for Pattern Recognition*, MIT Press.

Björkman, M. and Holmström, K. [1999]. Global optimization using the DIRECT algorithm in Matlab, *Advanced Modeling and Optimization* **1**(2): 17–37.

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

Blum, C. and Dorigo, M. [2004]. The hyper-cube framework for ant colony optimization, *IEEE Transactions on Systems, Man, and Cybernetics – Part B* **34**(2): 1161–1172.

Bonabeau, E., Dorigo, M. and Theraulaz, G. [1999]. *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NY.

Bosman, P. and Thierens, D. [1999]. An algorithmic framework for density estimation based evolutioinary algorithms, *Technical Report Technical Report UU-CS-1999-46*, Utrecht University, Netherlands.

Bosman, P. and Thierens, D. [2000]. Continuous iterated density estimation evolutionary algorithms within the IDEA framework, *in* M. Pelikan, H. Mühlenbein and A. O. Rodriguez (eds), *Proceedings of OBUPM Workshop at GECCO-2000*, Morgan-Kaufmann Publishers, San Francisco, CA, pp. 197–200.

Box, G. E. P. and Muller, M. E. [1958]. A note on the generation of random normal deviates., *Annals of Mathematical Statistics* **29**(2): 610–611.

Boyd, S. and Vandenberghe, L. [2004]. *Convex Optimization*, Cambridge University Press.

Brojden, C. [1967]. Quasi-Newton methods and their application to function mimimization, *Mathematical Computation* **21**: 368–381.

Bulirsch, R. and Stoer, J. [1991]. The conjugate-gradient method of Hestenes and Stiefel, *in* J. Stoer (ed.), *Introduction to Numerical Analysis*, Sprinver-Verlag, New York, NY, pp. 658–666.

Bullnheimer, B., Hartl, R. F. and Strauss, C. [1998]. Applying the Ant System to the vehicle routing problem, *in* I. H. Osman, S. Voß, S. Martello and C. Roucairol (eds), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Boston, MA, pp. 109–120.

Bullnheimer, B., Hartl, R. F. and Strauss, C. [1999]. An improved Ant System algorithm for the vehicle routing problem, *Annals of Operations Research* **89**: 312–328.

Bullnheimer, B., Kotsis, G. and Strauß, G. [1997]. Parallelization strategies for the Ant System, *Technical Report 8*, Vienna University of Economics and Business Administration, Vienna, Austria.

Camazine, S., Deneubourg, J.-L., Franks, N., Sneyd, J., Theraulaz, G. and Bonabeau, E. [2003]. *Self-Organization in Biological Systems*, Princeton University Press, Princeton, NJ.

Cao, Y. and Wu, Q. [1997]. Mechanical design optimization by mixed-variable evolutionary programming, *Proceedings of the Fourth IEEE Conference on Evolutionary Computation*, IEEE Press, pp. 443–446.

Cemy, V. [1985]. Thermodynamical approach to the traveling salesemen problem: Efficient simulation algorithm, *Journal of Optimization Theory and Applications* **45**: 41–51.

Cerny, V. [1985]. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, *Journal of Optimization Theory and Applications* **45**: 41–51.

Chellapilla, K. and Fogel, D. B. [1999]. Fitness distribution in evolutionary computation: Analysis of local extrema in the continuous domain, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-1999)*, IEEE Press, Piscataway, NJ, pp. 1885–1892.

Chelouah, R. and Siarry, P. [1999]. Enhanced continuous tabu search, *in* S. Voss, S. Martello, I. H. Osman and C. Roucairol (eds), *Meta-Heuristics Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Boston, MA, chapter 4, pp. 49–61.

Chelouah, R. and Siarry, P. [2000]. A continuous genetic algorithm designed for the global optimization of multimodal functions, *Journal of Heuristics* **6**: 191–213.

Chipman, H., George, E. and McCulloch, R. [1998]. Bayesian cart model search, *Journal of the American Statistical Associacion* **93**(443): 935–948.

Church, A. [1936]. An unresolvable problem of elementary number theory, *American Journal of Mathematics* **58**: 345–363.

Colorni, A., Dorigo, M., Maniezzo, V. and Trubian, M. [1994]. Ant System for job-shop scheduling, *JORBEL — Belgian Journal of Operations Research, Statistics and Computer Science* **34**(1): 39–53.

Cook, W., Cunningham, W., Pulleyblank, W. and Schrijver, A. [1998]. *Combinatorial Optimization*, John Wiley and Sons, New York, NY.

Cung, V.-D., Martins, S. L., Ribeiro, C. C. and Roucairol, C. [2001]. Strategies for the parallel implementation of metaheuristics, *in* C. C. Ribeiro and P. Hansen (eds), *Essays and Surveys in Metaheuristics*, Vol. 15 of *Operations Research/Computer Science Interfaces*, Kluwer Academic Publishers, Amsterdam, The Netherlands, chapter 13, pp. 263–308.

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

Deb, K. and Goyal, M. [1998]. A flexible optimiztion procedure for mechanical component design based on genetic adaptive search, *Journal of Mechanical Design* **120**(2): 162–164.

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

Dennis Jr., J. and More, J. [1977]. Quasi-Newton methods, motivation and theory, *SIAM Review* **19**: 46–89.

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

Doerner, K., Gutjahr, W., Hartl, R., Strauss, C. and Stummer, C. [2004]. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection, *Annals of Operations Research* **131**(1–4): 79–99.

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

Dorigo, M. and Di Caro, G. [1999]. The ant colony optimization meta-heuristic, *in* D. Corne, M. Dorigo and F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, pp. 11–32.

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

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

Dorigo, M., Maniezzo, V. and Colorni, A. [1991]. Positive feedback as a search strategy, *Technical Report 91-016*, Dipartimento di Elettronica, Politecnico di Milano, Italy.

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

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

Dréo, J. and Siarry, P. [2002]. A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions, *in* M. Dorigo, G. Di Caro and M. Sampels (eds), *Proceedings of the Third International Workshop on Ant Algorithms (ANTS'2002)*, Vol. 2463 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 216–221.

Dréo, J. and Siarry, P. [2004]. Continuous interacting ant colony algorithm based on dense heterarchy., *Future Generation Computer Systems* **20**(5): 841–856.

Eiben, A. E. and Bäck, T. [1997]. Empirical investigation of multiparent recombination operators in evolution strategies, *Evolutionary Computation* **3**(5): 347–365.

Fisher, M. [1981]. The lagrangian method for solving integer programming problems, *Management Science* **27**: 1–18.

Fogel, D. [1995]. *Evolutionary Computation*, IEEE Press, Piscataway, NJ.

Fogel, D. B. and Bayer, H. G. [1995]. A note on the empirical evaluation of intermediate recombination, *Evolutionary Computation* **4**(3): 491–495.

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

Fu, J.-F., Fenton, R. and Cleghorn, W. [1991]. A mixed integer-discrete-continuous programming method and its application to engineering design optimization, *Engineering Optimization* **17**(4): 263–280.

Fujita, H. and Yamaguti, M. [1981]. *The Newton Method and Related Topics*, Kinokuniya, Tokyo, Japan.

Gambardella, L. M. and Dorigo, M. [1996]. Solving symmetric and asymmetric TSPs by ant colonies, *in* T. Bäck, T. Fukuda and Z. Michalewicz (eds), *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, IEEE Press, Piscataway, NJ, pp. 622–627.

Gambardella, L. M. and Dorigo, M. [2000]. Ant Colony System hybridized with a new local search for the sequential ordering problem, *INFORMS Journal on Computing* **12**(3): 237–255.

Gambardella, L. M., Taillard, E. and Agazzi, G. [1999]. MACS-VRPTW: A multiple Ant Colony System for vehicle routing problems with time windows, *in* D. Corne, M. Dorigo and F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, pp. 63–76.

Glover, F. [1989]. Tabu search - part I, *ORSA Journal on Computing* **1**(3): 190–206.

Glover, F. [1990]. Tabu search - part II, *ORSA Journal on Computing* **2**(1): 4–32.

Glover, F. and Kochenberger, G. [2003]. *Handbook of Metaheuristics*, Kluwer Academic Publishers.

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

Golub, G. and van Loan, C. [1989]. *Matrix Computations*, 2nd edn, The John Hopkins University Press, Baltimore, MD.

Gomory, R. [1958]. Outline of an algorithm for integer solutions to linear programs, *Bulletin of the American Mathematical Society* **64**: 275–278.

Goss, S., Aron, S., Deneubourg, J.-L. and Pasteels, J.-M. [1989]. Self-organized shortcuts in the Argentine ant, *Naturwissenschaften* **76**: 579–581.

Grassé, P. P. [1946]. Les insectes dans leur univers, *Ed. du Palais de la découverte*, Paris.

Grassé, P. P. [1959]. La reconstruction du nid et les coordinations interindividuelles chez *bellicositermes natalensis* et *cubitermes* sp. la théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs., *Insectes Sociaux* **6**: 41–81.

Guignard, M. and Kim, S. [1987]. Lagrangian decomposition: a model yielding stronger lagrangian bounds, *Mathematical Programming* **39**: 215–228.

Guntsch, M. and Middendorf, M. [2002a]. Applying population based ACO to dynamic optimization problems, *in* M. Dorigo, G. Di Caro and M. Sampels (eds), *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, Vol. 2463 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 111–122.

Guntsch, M. and Middendorf, M. [2002b]. A population based approach for ACO, *in* S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf and G. Raidl (eds), *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTim*, Vol. 2279 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 71–80.

Guntsch, M. and Middendorf, M. [2003]. Solving multi-criteria optimization problems with population-based ACO, *in* C. Fonseca, P. Fleming, E. Zitzler, K. Deb and L. Thiele (eds), *Proceedings of Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003*, Vol. 2632 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 464–478.

Guntsch, M., Middendorf, M. and Schmeck, H. [2001]. An ant colony optimization approach to dynamic TSP, *in* L. Spector *et al.* (ed.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 860–867.

Guo, C., Hu, J., Ye, B. and Cao, Y. [2004]. Swarm intelligence for mixed-variable design optimization, *Journal of Zhejiang University SCIENCE* **5**(7): 851–860.

Gutjahr, W. J. [2004]. S-ACO: An ant-based approach to combinatorial optimization under uncertainty, *in* M. Dorigo, L. Gambardella, F. Mondada, T. Stützle, M. Birratari and C. Blum (eds), *ANTS'2004, Fourth Internatinal Workshop on Ant Algorithms and Swarm Intelligence*, Vol. 3172 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 238–249.

Hagan, M. and Menhaj, M. [1994]. Training feedforward networks with the marquardt algorithm, *IEEE Transactions on Neural Networks* **5**(6): 989–993.

Hansen, N. and Ostermeier, A. [2001]. Completly derandomized self-adaptation in evolution strategies, *Evolutionary Computation* **2**(9): 159–195.

Hastie, T., Tibshirani, R. and Friedman, J. [2001]. *The Elements of Statistical Learning*, Springer-Verlag, Berlin, Germany.

Hilal, M. and Boom, R. [1977]. Optmization of mechanical supports for large superconductive magnets, *Advances in Cryogenic Engineering* **22**: 224–232.

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

Iredi, S., Merkle, D. and Middendorf, M. [2001]. Bi-criterion optimization with multi colony ant algorithms, *in* E. Zitzler *et al.* (ed.), *Proceedings of the Evolutionary Multi-Criterion Optimization, First International Conference (EMO'01)*, Vol. 1993 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 359–372.

Kennedy, J. and Eberhart, R. C. [1995]. Particle Swarm Optimization, *Proceedings of IEEE International Conference on Neural Networks*, Vol. 4, IEEE Press, Piscataway, NJ, pp. 1942–1948.

Kern, S., Müller, S. D., Hansen, N., Büche, D., Očenášek, J. and Koumoutsakos, P. [2004]. Learning probability distributions in continuous evolutionary algorithms - A comparative review, *Natural Computing* **3**(1): 77–112.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. [1983]. Optimization by simulated annealing, *Science* **220**: 671–680.

Kokkolaras, M., Audet, C. and Dennis Jr., J. [2001]. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system, *Optimization and Engineering* **2**(1): 5–29.

Koza, J. [1992]. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.

Lampinen, J. and Zelinka, I. [1999a]. Mechanical engineering design optimization by differential evolution, *in* D. Corne, M. Dorigo and F. Glover (eds), *New Ideas in Optimization*, McGraw-Hill, London, UK, pp. 127–146.

Lampinen, J. and Zelinka, I. [1999b]. Mixed integer-discrete-continuous optimization by differential evolution, part 1: the optimization method, *in* P. Ošmera (ed.), *Proceedigns of MENDEL'99, 5th International Mendel Conference of Soft Computing*, Brno University of Technology, Brno, Czech Repuplic, pp. 71–76.

Lampinen, J. and Zelinka, I. [1999c]. Mixed integer-discrete-continuous optimization by differential evolution, part 2: a practical example, *in* P. Ošmera (ed.), *Proceedigns of MENDEL'99, 5th International Mendel Conference of Soft Computing*, Brno University of Technology, Brno, Czech Repuplic, pp. 77–81.

Land, A. and Doig, A. [1960]. An automatic method for solving discrete programming problems, *Econometrica* **28**: 97–520.

Lawler, E. L., Lenstra, J. K., Rinnooy-Kan, A. H. G. and Shmoys, D. B. [1985]. *The Travelling Salesman Problem*, John Wiley & Sons, New York, NY.

Lessing, L., Dumitrescu, I. and Stützle, T. [2004]. A comparison between ACO algorithms for the set covering problem, *in* M. Dorigo, L. Gambardella, F. Mondada, T. Stützle, M. Birratari and C. Blum (eds), *ANTS'2004, Fourth Internatinal Workshop on Ant Algorithms and Swarm Intelligence*, Vol. 3172 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 1–12.

Li, H.-L. and Chou, C.-T. [1994]. A global approach for nonlinear mixed discrete programing in design optimization, *Engineering Optimization* **22**: 109–122.

Li, Q., Li, X., McIntosh, G. and Boom, R. [1989]. Minimization of total refrigiration power of liquid neon and nitrogen cooled intercepts for SMES magnets, *Advances in Cryogenic Engineering* **35**: 833–840.

Loh, H. and Papalambros, P. [1991]. Computation implementation and test of a sequential linearization approach for solving mixed-discrete nonlinear design optimization, *Journal of Mechanical Design* **113**(3): 335–345.

Maniezzo, V. and Colorni, A. [1999]. The Ant System applied to the quadratic assignment problem, *IEEE Transactions on Knowledge and Data Engineering* **11**(5): 769–778.

Maniezzo, V., Colorni, A. and Dorigo, M. [1994]. The Ant System applied to the quadratic assignment problem, *Technical Report IRIDIA/94-28*, IRIDIA, Université Libre de Bruxelles, Belgium.

Mathur, M., Karale, S. B., Priye, S., Jyaraman, V. K. and Kulkarni, B. D. [2000]. Ant colony approach to continuous function optimization, *Ind. Eng. Chem. Res.* **39**: 3814–3822.

McGill, R., Tukey, J. and Larsen, W. [1978]. Variations of box plots, *The American Statistica* **32**: 12–16.

Merkle, D. and Middendorf, M. [2002]. Fast ant colony optimization on runtime reconfigurable processor arrays, *Genetic Programming and Evolvable Machines* **3**(4): 345–361.

Merkle, D., Middendorf, M. and Schmeck, H. [2002]. Ant colony optimization for resource-constrained project scheduling, *IEEE Transactions on Evolutionary Computation* **6**(4): 333–346.

Michel, R. and Middendorf, M. [1998]. An island model based Ant System with lookahead for the shortest supersequence problem, *in* A. E. Eiben, T. Back, M. Schoenauer and H.-P. Schwefel (eds), *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, Germany, pp. 692–701.

Michel, R. and Middendorf, M. [1999]. An ACO algorithm for the shortest common supersequence problem, *in* D. Corne, M. Dorigo and F. Glover (eds), *New Ideas in Optimisation*, McGraw-Hill, Boston, MA, pp. 51–61.

Monmarché, N., Venturini, G. and Slimane, M. [2000]. On how *Pachycondyla apicalis* ants suggest a new search algorithm, *Future Generation Computer Systems* **16**: 937–946.

Mühlenbein, H. and Paaß, G. [1996]. From recombination of genes to the estimation of distributions: I. binary parameters, *Proceedings of Parallel Problem Solving from Nature-PPSN IV*, Vol. 1411 of *LNCS*, pp. 178–187.

Musicki, Z., Hilal, M. and McIntosh, G. [1989]. Optimization of cryogenic and heat removal system of space borne magnets, *Advances in Cryogenic Engineering* **35**: 975–982.

Nelder, J. A. and Mead, R. [1965]. A simplex method for function minimization, *Computer Journal* **7**: 308–313.

Očenášek, J. and Schwarz, J. [2002]. Estimation distribution algorithm for mixed continuous-discrete optimization problems, *Proceedings of the 2nd Euro-International Symposium on Computational Intelligence*, IOS Press, Amsterdam, Netherlands, pp. 227–232.

Ostermeier, A., Gawelczyk, A. and Hansen, N. [1994]. Step-size adaptation based on non-local use of selection information, *in* Y. Davidor, H.-P. Schwefel and R. Männer (eds), *Parallel Problem Solving from Nature – PPSN III*, Vol. 866 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 189–198.

Padberg, M. and Rinaldi, G. [1991]. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, *SIAM Review* **33**: 60–100.

Pandia Raj, R. and Kalyanaraman, V. [2005]. GA based optimal design of steel truss bridge, *in* J. Herskovits, S. Mazorche and A. Canelas (eds), *Proceedigns of 6th World Congress of Structural and Multidisciplinary Optimization*, pp. CD–ROM proceedings.

Pasteels, J. M., Deneubourg, J.-L. and Goss, S. [1987]. Self-organization mechanisms in ant societies (i): Trail recruitment to newly discovered food sources., *Experientia Supplementum* **54**: 155–175.

Peitgen, H.-O. [1989]. *Newton's Method and Dynamical Systems*, Kluwer Academic Pulishers, Dordrecht, The Netherlands.

Pelikan, M., Goldberg, D. and Sastry, K. [2000]. Bayesian optimization algorithm, decision graphs, and Occam's razor, *Technical Report IlliGAL Report No. 2000020*, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL.

Powell, M. J. D. [1964]. An efficient method for finding the minimum of a function of several variables without calculating derivatives, *Computer Journal* **7**: 155–162.

Praharaj, S. and Azarm, S. [1992]. Two level nonlinear mixed discrete continuous optimization-based design: An application to printed circuit board assemblies, *Journal of Electronic Packaging* **114**(4): 425–435.

Prechelt, L. [1994]. PROBEN1—a set of neural network benchmark problems and benchmarking rules, *Technical Report Tech. Rep. 21*, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany.

Rahoual, M., Hadji, R. and Bachelet, V. [2002]. Parallel Ant System for the set covering problem, *in* M. Dorigo, G. D. Caro and M. Sampels (eds), *Proceedings of Ant Algorithms - Third International Workshop, ANTS 2002*, Vol. 2463 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 262–267.

Ralston, A. and Rabinowitz, P. [1978]. *A First Course in Numerical Analysis*, 2nd edn, McGraw-Hill, New York, NY.

Ramalhinho-Lourenço, H., Martin, O. and Stützle, T. [2002]. Iterated local search, *in* F. Glover and G. K. (Herausgeber) (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, Norwell, MA, pp. 321–353.

Ramalhinho-Lourenço, H. and Serra, D. [1998]. Adaptive approach heuristics for the generalized assignment problem, *Technical Report Economic Working Papers Series No.304*, Universitat Pompeu Fabra, Dept. of Economics and Management, Barcelona, Spain.

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

Rechenberg, I. [1973]. *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, Germany.

Rumelhart, D., Hinton, G. and Williams, R. [1986]. Learning representations by back-propagation errors, *Nature* **323**: 533–536.

Sandgren, E. [1990]. Nonlinear integer and discrete programming in mechanical design optimization, *Journal of Mechanical Design* **112**: 223–229.

Schmidt, H. and Thierauf, G. [2005]. A combined heuristic optimization technique, *Advances in Engineering Software* **36**: 11–19.

Schwefel, H.-P. [1981]. *Numerical Optimization of Computer Models*, John Wiley and Sons, New York, NY.

Sellar, R., Batill, S. and Renaud, J. [1994]. Optimization of mixed discrete/continuous design variable systems using neural networks, *Proceedings of AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA, Reston, VA.

Siarry, P., Berthiau, G., Durbin, F. and Haussy, J. [1997]. Enhanced simulated annealing for globally minimizing functions of many continuous variables, *ACM Transactions on Mathematical Software* **23**(2): 209–228.

Socha, K. [2003]. The influence of run-time limits on choosing Ant System parameters, *in* E. Cantu-Paz *et al.* (ed.), *Proceedings of GECCO 2003 – Genetic and Evolutionary Computation Conference*, Vol. 2723 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 49–60.

Socha, K. [2004]. ACO for continuous and mixed-variable optimization, *in* M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada and T. Stützle (eds), *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, Vol. 3172 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 25–36.

Socha, K. and Blum, C. [2006]. Ant colony optimiztion, *in* E. Alba and R. Martí (eds), *Metaheuristic Procedures for Training Neural Networks*, Computer Science Interfaces Series, Springer-Verlag, Berlin, Germany, chapter 8, pp. 153–180.

Socha, K. and Blum, C. [2007]. Hybrid ant algorithms applied to feed-forward neural network training: An application to medical pattern classification, *Neural Computing and Applications* **16**(3): 235–248.

Socha, K. and Dorigo, M. [2008]. Ant colony optimization for continuous domains, *European Journal of Operations Research* **185**(3): 1155–1173.

Socha, K., Knowles, J. and Sampels, M. [2002]. A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the university timetabling problem, *in* M. Dorigo, G. Di Caro and M. Sampels (eds), *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, Vol. 2463 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 1–13.

Socha, K., Sampels, M. and Manfrin, M. [2003]. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art, *in* G. Raidl *et al.* (ed.), *Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, Vol. 2611 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 334–345.

Spiser, M. [2005]. *Introduction to the Theory of Computation*, 2nd edn, Course Technology, Boston, MA.

Stelmack, M. and Batill, S. [1997]. Concurrent subspace optimization of mixed continuous/discrete systems, *Proceedings of AIAA/ASME/ASCE/AHS/ASC 38th Structures, Structural Dynamic and Materials Conference*, AIAA, Reston, VA.

Storn, R. and Price, K. [1995]. Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces, *Technical Report TR-95-012*, International Computer Science Institute, Berkeley, CA.

Storn, R. and Price, K. [1997]. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* **11**: 341–359.

Stützle, T. [1998]. Parallelization strategies for ant colony optimization, *in* A. E. Eiben, T. Bäck, M. Schoenauer and H.-P. Schwefel (eds), *Proceedings of Parallel Problem Solving from Nature - PPSN V: 5th International Conference*, Vol. 1498 of *LNCS*, Springer-Verlag, Berlin, Germany, pp. 722–731.

Stützle, T. and Dorigo, M. [1999]. ACO algorithms for the traveling salesman problem, *in* K. Miettinen, M. M. Mäkelä, P. Neittaanmäki and J. Périaux (eds), *Evolutionary Algorithms in Engineering and Computer Science*, John Wiley & Sons, Chichester, UK, pp. 163–183.

Stützle, T. and Hoos, H. [1998]. The $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and local search for combinatorial optimization problems: Towards adaptive tools for combinatorial global optimisation, *in* S. Voss, S.Martello, I.H.Osman and C.Roucairol (eds), *Meta-Heuristic, Advances and Trends in Local Search Paradigma for Optimization*, Kluwer Academic Publishers, Boston, MA, pp. 313–329.

Stützle, T. and Hoos, H. H. [2000]. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System, *Future Generation Computer Systems* **16**(8): 889–914.

Talbi, E.-G., Roux, O., Fonlupt, C. and Robillard, D. [1999]. Parallel ant colonies for combinatorial optimization problems, *Proceedings of the 11 IPPS/SPDP'99 Workshops held in conjunction with the 13th International Parallel Processing Symposium and the 10th Symposium on Parallel and Distributed Processing*, Springer-Verlag, Berlin, Germany, pp. 239–247.

Thierauf, G. and Cai, J. [2000]. Evolution strategies—parallelization and application in engineering optimization, *in* B. Topping (ed.), *Parallel and distributed processing for computational mechanics: systems and tools*, Saxe-Coburg Publications, Edinburgh, UK, pp. 329–349.

Torczon, V. [1997]. On the convergence of pattern search algorithms, *SIAM Journal on Optimization* **7**: 1–25.

Turing, A. [1936]. On computable numbers, with an application to the entscheidungsproblem, *Proceedings of the London Mathematical Society* **42**(2).

Turkkan, N. [2003]. Discrete optimization of structures using a floating point genetic algorithm, *Proceedings of Annual Conference of the Canadian Society for Civil Engineering*, pp. CD–ROM proceedings.

Wodrich, M. and Bilchev, G. [1997]. Cooperative distributed search: The ant's way, *Control & Cybernetics* **26**(3): 413–446.

Wu, S.-J. and Chow, P.-T. [1995]. Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization, *Engineering Optimization* **24**(2): 137–159.

Yamaguchi, M., Ohmori, T. and Yamamoto, A. [1991]. Design optimization of a vaporcooled radiation shield for LHe cryostat in space use, *Advances in Cryogenic Engineering* **37**: 1367–1375.

Yuan, B. and Gallagher, M. [2003]. Playing in continuous spaces: Some analysis and extension of population-based incremental learning, *in* Sarker, R. *et al.* (ed.), *Proceedings of Congress of Evolutionary Computation (CEC)*, IEEE Press, Piscataway, NJ, pp. 443–450.

Index